Rowan University

## Rowan Digital Works

---

Theses and Dissertations

---

5-18-2020

# Prediction of drug-drug interaction potential using machine learning approaches

Joseph Scavetta
*Rowan University*

---

# PREDICTION OF DRUG-DRUG INTERACTION POTENTIAL USING MACHINE LEARNING APPROACHES

by

Joseph Scavetta

A Thesis

Submitted to the
Department of Computer Science
College of Science and Mathematics
In partial fulfillment of the requirement
For the degree of
Master of Science in Computer Science
at
Rowan University
May 1, 2020

Thesis Chair: Serhiy Y. Hnatyshyn, Ph.D.

**Dedications**

I dedicate this thesis to my mother, Valerie Scavetta, the hardest working person I have ever known. Through hard times, she has never given up and has always given life her all, inspiring me to do the same. Without her constant support and inspiration, this thesis would not have been possible.

I also dedicate this thesis to Sarah Senula for her love and emotional support. I could not have completed this thesis without her.

# Abstract

Joseph Scavetta
PREDICTION OF DRUG-DRUG INTERACTION POTENTIAL USING MACHINE
LEARNING APPROACHES
2019-2020
Serhiy Y. Hnatyshyn, Ph.D.
Master of Science in Computer Science

Drug discovery is a long, expensive, and complex, yet crucial process for the benefit of society. Selecting potential drug candidates requires an understanding of how well a compound will perform at its task, and more importantly, how safe the compound will act in patients. A key safety insight is understanding a molecule's potential for drug-drug interactions. The metabolism of many drugs is mediated by members of the cytochrome P450 superfamily, notably, the CYP3A4 enzyme. Inhibition of these enzymes can alter the bioavailability of other drugs, potentially increasing their levels to toxic amounts. Four models were developed to predict CYP3A4 inhibition: logistic regression, random forests, support vector machine, and neural network. Two novel convolutional approaches were explored for data featurization: SMILES string auto-extraction and 2D structure auto-extraction. The logistic regression model achieved an accuracy of 83.2%, the random forests model, 83.4%, the support vector machine model, 81.9%, and the neural network model, 82.3%. Additionally, the model built with SMILE string auto-extraction had an accuracy of 82.3%, and the model with 2D structure auto-extraction, 76.4%. The advantages of the novel featurization methods are their ability to learn relevant features from compound SMILE strings, eliminating feature engineering. The developed methodologies can be extended towards predicting any structure-activity relationship and fitted for other areas of drug discovery and development.

# Table of Contents

**Table of Contents (Continued)**

**Table of Contents (Continued)**

# List of Figures

**List of Tables**

# Chapter 1

## Introduction to Drug Discovery

The origins of contemporary drug discovery can be traced back to the 1800s, when scholars and scientists made many fundamental developments in chemistry: Dmitri Mendeleev published the periodic table, Svante Arrhenius began theorizing about acids and bases, and August Kekulé explored aromatic organic molecules, just to name a few [1]. These advances shook the pharmacology field giving birth to a new area of chemistry driven pharmacology. Much later, combinatorial chemistry and high-throughput screening led to a new paradigm in drug discovery: parsing a plethora of data and compounds to find those that will be successful. However, we can only realize the significance of the findings when we are able to read, extract, and apply the data. Unfortunately, data analytics in drug discovery was slow to come as shown by the lack of improvement in the number of drugs reaching markets [1].

The next evolution in drug discovery, similar to the establishment of chemistry driven pharmacology, involves an alliance between pharmaceutical disciplines, bioinformatics, and computer science. Novel developments in computer science research, especially in the area of machine learning, has led to algorithms that allow scientists to use historic data for making predictions on new data, while minimizing cost and the number of errors. Computer aided drug discovery has and will continue to increase the productivity, speed, and efficiency of drug selection and development [2].

**Drug Discovery Overview**

The use of compounds for medicinal purposes has been a vital aspect of human society. Evidence of drug use dates to the prehistoric period, with written evidence appearing in ancient Egypt, China, Rome, Greece, and many other civilizations [3]. In more recent times, pharmaceuticals have increased life expectancy and significantly reduced the effects of disease and sickness. Diseases that had a devastating effects on society in the past, such as bacterial infections, smallpox, and tuberculosis, are now generally non-lethal or have a low-chance to contract [4]. Even diseases that are harder to control or cure, such as cancer or HIV, have recently began to shift from fatal to chronic but manageable [4]. In modern society, quick and efficient drug discovery is becoming more important as the population continues to increase, and people tend to live longer.

Bringing new and improved drugs to the market is important for the health and safety of society. Though much has improved from the birth of pharmaceutical sciences, the lifespan of a drug is still lengthy, and the process is costly. On average, it takes over ten years for a new candidate drug to be approved [5]. Many new candidate compounds never make it to clinical trials due to a prohibitive cost of failure: from 2015 to 2016 the median cost of pivotal clinical trial was estimated at $19 million [6]. Less than 1% of synthesized compounds enter trials [4]. For the drugs that do make it to clinical trials, the probability of success is also low: the highest three success rates were

- 32.6% for ophthalmology drug candidates,
- 25.5% for cardiovascular drug candidates, and
- 25.2% for infectious disease products;

the lowest success rate came in at just 3.4% for oncology trials [7]. Drugs that do succeed through pre-clinical and clinical trials have an average cost of $2.6 billion per compound from start to approval [8]. Many candidate compounds will never result in a new marketed drug as the candidates will fail, while for those that do succeed, both the duration and costs of the discovery and development are significant.

Following Ator et al.'s overview of drug discovery and development, summarized in Figure 1, this process starts with target identification where the receptors that the drug should act upon are selected [4]. Next, the compounds that can act on the target must be found, which often requires high-throughput screening (HTS) and structure-based drug design methods [4]. HTS allows us to select compounds that are active against targets. Compounds from the selection pool are optimized to obtain better drug-like properties [4]. Optimization is focused on careful examination of candidate drug properties such as absorption, distribution, metabolism, excretion, and toxicity (ADMET) [9]. After a drug candidate completes preclinical review and achieves its safety and efficacy goals, it then can enter clinical development. Clinical development consists of four phases:

- Phase I trials test the drug candidate for safety on $10 - 100$ healthy human volunteers,

- Phase II trials continue to test safety while also testing efficacy in $50 - 500$ of those with the targeted disease,

- Phase III trials test the drug in full-scale with diverse patients and several medical centers, and

- Finally, Phase IV trials monitor the drug's adverse effects post-approval [4].

3
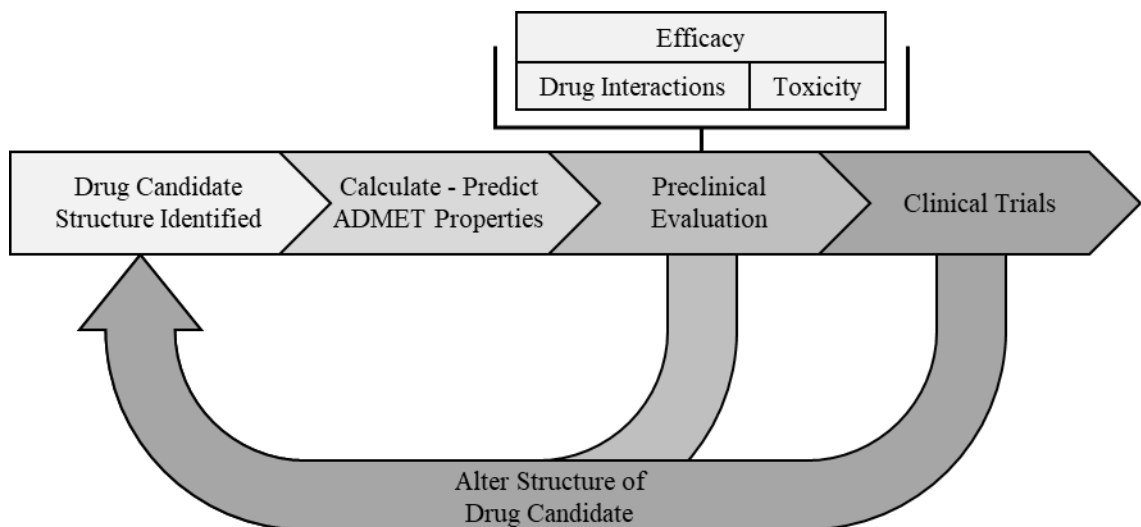
*Figure 1*. The standard workflow within the drug development pipeline.

**Why Drugs Fail**

For a drug to get approved, it must enter and succeed in clinical trials. In Phase I trials most drugs fail due to toxicity. During the Phase II and Phase III trials, drugs often fail due to efficacy problems, although, toxicity still plays a large role in drug failure [10]. Because toxicity and efficacy are the two most common causes of drug failure, ADMET properties are crucial for differentiating between successful drugs and those that will fail.

Failure during the late stages of the clinical trials is not only very costly in terms of money, time, and labor, but it is also potentially dangerous to patients participating in the trials. Clinical trial failures can either be unavoidable due to inadequate scientific advances or could be prevented through scientific rigor, curiosity, and discipline [11]. Unavoidable failures often lack well performing models due to an insufficient knowledge

base of the underlying chemistry and/or biology [11]. Preventable failures are explained by a lack of optimal study designs, dosages, and safety data [11]. Overall, learning from earlier mistakes, specifically, collecting, sharing, and analyzing data from both successful and failed drug tests can help reduce the number of avoidable failures. Furthermore, developing new models from these data can also help improve the conditions of the currently unavoidable failures.

**Drug Candidate Optimization**

The future improvements in the pharmaceutical industry are likely to focus on methods that will significantly reduce failures in clinical trials. Focusing on an early evaluation of new drug candidates will help reduce the associated cost and time of the overall process. Improving pharmacological properties of drug candidates in early stages will reduce the burden on managing compounds that will eventually fail in clinical trials [12]. Selecting successful drug candidates is a complex process that requires an understanding of drug-like properties, relying on the analyses of overwhelming amounts of data [13]. Various drug candidate selection techniques are discussed below.

**Rule-based drug discovery.** Rule-based generalizations act as guidelines as to which physiochemical properties one should expect from a successful drug, as compared to a less effective drug candidate. Defining common generalizations for the desired characteristics of drug-like properties is complicated by variations of a drug's targets and routes of transmission. The Lipinski's rule of 5 is well known approach to determining which properties make a successful drug candidate. This approach specifies upper bounds on molecular properties such as hydrogen bond donors and acceptors, molecular mass,

and lipophilicity [14]. While drugs that satisfy Lipinski's rule of 5 are often more successful than those that do not, many exceptions exist where the model would disqualify a successful drug and vice-versa [15]. The Lipinski's ruleset focuses on the absorption properties of a drug, which is only one of many factors that determines a successful drug candidate. Furthermore, this approach is primarily expressive of permeability potential; while solubility and dosage may also play a role in absorption [14]. Also, the bounds apply to oral drugs that do not act as substrates for naturally occurring transporters [14]. While there are limitations to this approach, the Lipinski's rule of 5 is a useful starting point in selecting important drug-like properties in future drug-prediction models, specifically for absorption models.

Other rule-based methods have been created as an extension to Lipinski's rule of 5 [16]. While the rule-based methods provide a simplified approach for determining drug success, they are limited in substantial ways. Having a strict cutoff points implies that these properties are discrete, rather than continuous [17]. Such assumptions can result in many missed opportunities. Furthermore, these rules are generated only from properties that successful drugs have in common. However, if the property distributions explored in successful drug candidates are similar to failed drug candidates, then these properties are ultimately uninformative [17].

To replace cutoffs with a continuous scale, Bickerton et al. developed the quantitative estimate of drug-likeness (QED) [18]. This approach performs quite well, however, it still does not consider whether a property is truly predictive, i.e. has a different distribution from failed drug candidates. To address both shortcomings of the

6

rule-based approaches, the relative drug likelihood (RDL) can be computed [17]. The RDL approach performs better than QED as it uses distributions from both successful drugs candidates and failed drug candidates. This allows RDL to identify properties that are important to identifying successful drug candidates. Yusof et al. expanded the idea of RDL and created a new algorithm based on the patient rule induction method (PRIM) [19]. PRIM differs from RDL by exploring all the properties simultaneously and identifying redundant properties.

While rule-based methods have improved, they may still lack the ability to generalize non-linear patterns in the data and may miss important relations. Current approaches in drug discovery have now shifted towards applying traditional and novel machine learning algorithms to chemical datasets with a promise of exploring non-linear patterns within data [2].

**Machine learning in drug discovery.** There have been numerous models and commercial software that can predict various ADMET properties for drug candidates with the help of machine learning techniques [9], [20]. For example, successful drugs typically have a solubility, denoted as *log S*, with the values ranging from -1 to -5. However, finding the solubility of a compound is difficult. Rather than directly measuring compound's solubility, the researchers apply machine learning techniques to construct solubility models using existing data on other compounds. This approach achieves high accuracy of predicting compound solubility, performing as well as experimental measurements [21]. Finding favorable ADMET properties is one of the areas were large sets of data already exist, thus, machine learning can be applied. Other

properties, such as a compound's pharmacokinetics, can also be modeled using machine learning techniques, but experimentally obtained data sets are less common.

Modeling pharmacokinetic properties of a potential drug is important for predicting success in clinical trials. Pharmacokinetic parameters can be modeled based on the potential drug's physiochemical properties and through experimental assays [20]. For example, the random forests, a popular machine learning technique, was used to model the volume of distribution, one of many important pharmacokinetic parameters [20], [22].

Quantitative structure–activity relationship (QSAR) modeling for ligand-based visual screening, has been benefiting from machine learning techniques. Specifically, researchers used machine learning algorithms to determine which drugs match a certain query in a database of potential compounds [23]. QSAR modeling relies on the idea of structural similarity: compounds with similar structures have similar bioactivities [24]. In general, QSAR models employ data from the molecular structure of ligands and examines physiochemical properties, therapeutic activities, and pharmacokinetic parameters to predict the best molecules for a target [24]. Overall, machine learning based QSAR modeling for visual screening has advanced many aspects of drug discovery and development, specifically, by taking advantage of big data to predict complex biological phenomena [2].

# Chapter 2

## Machine Learning Overview

Simulating human intelligence has been an active area of research in mathematics as far back as the 1700s, when Thomas Bayes developed mathematical fundamentals of what has become Bayes' Theorem [25]. Machine learning aims to create algorithms that can learn from given data to solve problems without specific instructions. Machine learning is often viewed as a subset of artificial intelligence, though it differs from common artificial intelligence algorithms in that machine learning emphasizes data rather than pure domain expertise to complete a task. One important sub-section of machine learning is a family of methods that devise models to classify or predict a value for an unknown sample, given a sample of data that has already been classified or has established values. These methods are commonly categorized as supervised machine learning [26].

Supervised machine learning employs statistical methods with fast and efficient algorithms to predict some target function $f: X \rightarrow y$. The target function is unknown, and may always be unknown, however, using some set of input data $x \in \mathbb{R}^d = X$, and some known output data $y$, a function $g$ can be created to closely approximate $f$. The combination of input data $X$ and output data $y$ is called a training set and is denoted as $(x_1, y_1), (x_2, y_2) \dots, (x_N, y_N)$. Using the training examples and a set of hypothesis functions $H$, often an infinite set, a learning algorithm can select $g$ from $H$ that best approximates $f$ [27].

9

## Machine Learning Approaches

**Linear models.** Many of the fundamental machine learning algorithms stems from the statistical work of linearly modeling relationships between two or more variables. This type of modeling is often referred to as linear regression, and it was presented as far back as 1886 by geneticist Francis Galton when he was investigating the difference in height between parents and children [28]. The overall goal of linear regression is to predict some dependent variable $y$ given some independent variable(s) $x_m$ using weighted relationships in the form:

$$y_i = \beta_0 + \beta_1 x_1 + \cdots + \beta_m x_m + \epsilon \qquad \text{Eq. 1}$$

where $\beta_j$ acts as corresponding weights of the variable $x_j$, and $\epsilon$ is the random error [29]. Using known data for the variable(s) $x_m$, a prediction of y, denoted as $\hat{y}$, can be computed using a modification of equation 1 as follows:

$$\hat{y}_i = \beta_0 + \beta_1 x_1 + \cdots + \beta_m x_m \qquad \text{Eq. 2}$$

To find the best predictive function, the weights $\beta_j$ can be adjusted to minimize the error between the predicted $\hat{y}_i$ and a known $y_i$. The least squares method is employed to minimize the error on each known sample $\boldsymbol{y} = y_i \in \mathbb{R}^n$ and the samples' corresponding prediction from equation 2 [29]. The least squares error can be calculated using the form:

$$\sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \qquad \text{Eq. 3}$$

There are two common approaches for minimizing the error: (1) using the normal equations for a closed form solution and (2) using an iterative method such as gradient descent. Assuming linear independence between the variables, a unique solution can be found following the closed form:

$$\boldsymbol{\beta} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y} \qquad\qquad \text{Eq. 4}$$

where $\boldsymbol{\beta}$ is an optimal weight vector given a matrix of independent variables $\boldsymbol{X} = x_{ij} \in$ $\mathbb{R}^{m\times n}$ and a vector of corresponding dependent variables $\boldsymbol{y} = y_i \in \mathbb{R}^n$ [27]. A general solution to the minimization problem is to use gradient descent, which iteratively adjust weights and recomputes the global minimum of the error function, until the best solution is found [30]. Linear regression is a popular statistical method for finding optimal linear relationships between independent and dependent variables. It is often a good starting point for solving a prediction task. However, linear regression has limitations as data can often be non-linear.

A close cousin to linear regression is logistic regression, one of the earliest and most commonly utilized machine learning algorithms for discrete classification [31]. The logistic regression learning algorithm approximates a target function. However, rather than predicting a functional relationship $f(\boldsymbol{X}) = \boldsymbol{y}$, logistic regression models the probability $P(\boldsymbol{y}|\boldsymbol{X})$ of the dependent variables $\boldsymbol{y} = y_i \in [0,1]$ belonging to a certain class, given a set of independent variables $\boldsymbol{X} = x_{ij} \in \mathbb{R}^{m\times n}$ [32]. The base form for logistic regression is similar to linear regression in equation 1. However, logistic regression applies a soft threshold to equation 1 to achieve the form:

$$y_i = \theta(\beta_0 + \beta_1 x_1 + \cdots + \beta_m x_m + \epsilon) \qquad\qquad \text{Eq. 5}$$

where $\theta$ is the logistic function:

$$\theta(s) = e^s/1 + e^s \qquad\qquad \text{Eq. 6}$$

[27]. A common error measure for logistic regression is maximum likelihood, which can be minimized in the form:

$$\frac{1}{n}\sum_{i=1}^{n}\ln(1 + e^{-y_i\boldsymbol{\beta}^T x_i}) \qquad\qquad \text{Eq. 7}$$

Unlike linear regression, logistic regression does not have a closed form solution. Instead, to minimize error, we can employ gradient descent method which can adjust the logistic model towards the steepest decrease in error [33]. Overall, logistic regression is equivalent to linear regression, aside from the addition of the soft threshold logistic function around the linear model to restrict the functional range to [0,1].

As more variables, or features, are added to linear models, their complexity increases. This can increase the temptation to fit the function too closely to the limited set of data points, or in other words, overfit the training data, which may lead to poor generalization of new data [34]. To mitigate the effects of overfitting, we can use methods that penalize models for having too large a dependency on any given feature. This method is often called regularization. Regularization adds another term to the error function, which penalizes large weights in the model. The two common methods for regularization are ridge ($L_2$) and lasso ($L_1$). Ridge regularization adds the term $\lambda\sum_{j=1}^{m}\beta_j^2$ to the error function, while lasso regularization adds the term $\lambda\sum_{j=1}^{m}|\beta_j|$ [35]. For example, with lasso regularization equation 7 can be extended to:

$$\frac{1}{n}\sum_{i=1}^{n}\ln(1 + e^{-y_i\boldsymbol{\beta}^T x_i}) + \lambda\sum_{j=1}^{m}|\beta_j| \qquad\qquad \text{Eq. 8}$$

The $\lambda$ acts as a parameter that controls the complexity of the model; i.e., a low $\lambda$ allows for a more complicated model while a high $\lambda$ reduces the complexity. The differences between the two approaches is that lasso regression allows for the optimal model to drop terms completely (i.e., setting the feature weights to 0), while ridge regularization does

12

not allow a 0 weight [35]. Traditionally, linear models are often the first sought after

model as they are fast, efficient, and support regularization methods to avoid overfitting.

Though for more complex problems, they may underfit the target function. The field of

machine learning also boasts a large number of other algorithms that are not tied to the

linear domain, which we will discuss next.

      **Decision trees.** Using a set of criteria or decision points, we can split and classify

samples based on the data values. This simple idea is the fundamental concept of decision

trees [32]. The decision tree consists of branch nodes that split paths and leaf nodes that

act as the outcome for a sample. A decision tree is created from top down; determining

which feature of the data will act as the first branch node, and so on. A range of values

can be considered for each branch of a feature node. For example, if some value for the

feature is less than 10, the tree may branch one way, otherwise, it will branch a separate

way. For this reason, ordering features with the most informative features at the top of the

tree is crucial. The more information a feature provides, the more likely it will split the

data into separable classifications down the tree. Feature importance is determined using

one of several feature selection methods.

      The first notable feature selection method, introduced by Breiman, Friedman,

Stone, and Olshen in 1984, is the *Gini index* or *Gini impurity*. The Gini index can be

interpreted as the estimated probability of misclassifying a random sample using the

selected feature, such that a Gini index of 0 implies a 0% chance that any sample will be

misclassified while a Gini index of 1 implies a 100% chance of misclassification [36].

13

Another widely used feature selection method is information gain. Information gain is based on information entropy, which is defined as the average production of novel information from randomly selected data. High entropy relates to a low probability of correctly predicting data, while low entropy relates to a high probability of correctly predicting information [37]. In short, the best feature to select for branching is the one that gains the most information, or in other words, select the feature that results in the greatest reduction of the overall entropy of the prediction task [38].

The strength of decision tree based approaches is accurate classification of data and also determination of the variable importance [32]. Though decision trees can often achieve high accuracy on the training data, they are prone to overfitting. This may lead to a lack of generalization and a decrease in prediction accuracy for new data [39].

Rather than performing classification using only one decision tree, an ensemble of decision trees can be used to significantly increase the performance of the predictive task [40]. Adaptive boosting combines multiple "weak learners" (i.e., models with low predictive effectiveness) into a single unified strong learner (i.e., a model that performs predictive task well). [41]. Adaptive boosting starts with several decision trees for the dataset, which may be weak learners. Next, the algorithm adjusts the weights of misclassified samples and selects a random set of data to create a new decision tree. Effectively, each new tree is dependent on the errors of the earlier trees. Along with this, each tree stores a weight of importance towards the final collective decision based on how well that individual tree performed.

Another popular ensemble method is bootstrap aggregation, or bagging. The bagging method constructs independent decision trees, each created from a bootstrapped sub-sample of the original dataset. This approach does sampling with replacement which allows duplication of observations [40]. The dataset is split into bootstrap samples many times, so that many varying trees are created. The resulting decision trees collectively perform prediction task by selecting the majority answer.

The random forests approach works similarly to the bagging method by creating multiple decision trees, however, created trees differ in their feature sub-space. Each tree is created by sampling the entire training set but with a randomly selected subset of available features [39]. Again, the consensus among individual trees is used to make the final prediction. In practice, the random forests approach relies on a combination of bagging with random feature sub-spacing [42]. Decision trees used collectively as random forests often provide some of the best average performance and are less sensitive to overfitting as compared to other methodologies [43], [44].

**Support vector machine.** The support vector machine (SVM) is a versatile machine learning approach that allows classification of linearly separable as well as non-linear data. SVMs are similar to linear models, in the sense that they try to fit a line that can split data into two classes. However, the SVM approach has the advantage of finding the optimal hyperplane which maximizes the distance of the data on both sides [45]. This approach allows for more robustness in generalizing towards new data after training, as there is more leeway for the data to trend towards hyperplane without crossing the class boundary. The SMV approach provides model stability by finding the optimal solution

15

every time, yielding consistent results for the same data [45]. This consistency makes support vector machine appealing for practical use. If data is not linearly separable, then the non-linear data can be projected into a high dimension, potentially allowing the data to become linearly separable [45]. SVM's versatility of handling linear and non-linear data makes it a practical solution to many varying prediction tasks that may differ widely in complexity of the data.

**Artificial neural network.** Recently, artificial neural networks (ANN) gained popularity and have reached the forefront of machine learning research and technology. Neural networks are based on the perceptron, a fundamental learning algorithm that was modeled from the biological neuron to learn a task [46]. A basic neural network consists of multiple layers of perceptrons, referred to as neurons, which are linked together in numerous ways to produce complex relationships from the input to the output data. Though research on neural networks was stagnant following the criticism of machine learning by Minsky and Papert in their book *Perceptrons: an introduction to computational geometry* [47], ANNs returned to the forefront with the creation of back-propagation methods, an increase in computational power, and availability of modern computers. Back-propagation is a technique that allows for the connections, or weights, between one neuron and another to be adjusted towards a global minimization of the output error for many neurons and the true output [48]. Using back-propagation, ANN models consisting of hundreds or more neurons have their connections adjusted in a way that the overall model can take an input and accurately output a response in a short amount of time and with little human effort. The greatest advantage of ANNs is the

possibility of modeling complicated non-linear relationships, as neural networks can often find hidden relationships between data that are difficult to detect.

In recent years, many machine-learning practitioners have used the term deep learning to describe a more complex neural networks that have many hidden layers. Deep learning succeeds at modeling even the most complicated relationships than a smaller standard ANN cannot achieve. For example, deep learning allowed computers to learn to play and to defeat professionals at the game of Go [49]. Convolutional neural networks (CNN) and recurrent neural networks (RNN) are examples of deep learning artificial neural networks. Similar to a standard ANN, a CNN has multiple layers of neurons. However, CNNs employ layers of convolution kernels. In contrast to a fully connected ANN that connects all neurons within two adjacent layers, convolution kernels only look at a sub-sample of input neurons to generate an output [50]. Overall, convolutional neural networks work very well with image recognition. The convolution kernels can learn image filters without extensive feature-engineering on the input, and with less computational overhead as compared to a fully connected network.

Recurrent neural networks differ from typical feed-forward networks in creation of connections across neurons based on time or sequence. This adjustment allows for a dynamic learning process where the neural network can remember earlier data in a sequence while training the next step in a sequence. Because of their notion of memory, recurrent networks can work well for sequential inputs such as text or speech [51]. Deep networks have many more variations that are well suited to specific problems.

**Machine Learning Tools**

The data that is being analyzed is often not perfect and may suffer from such deficiencies as missing values, major differences in unit variance, unbalanced classes, or an overwhelming number of features. One must also consider how the models will be evaluated. Although the end goal is to minimize error in the models, the process of determining and comparing errors across models is not resolute. Finally, additional information about models such as the features that were most important or how changes in the features affect the models' predictions may be of interest.

**Data preprocessing.** It is uncommon for the data to be collected in a perfect form. Preprocessing is often employed to handle inconsistencies and issues within the collected datasets. There is no simple solution for handling all issues with missing data. Approaches to handle missing data may include removing the samples or features that have missing data or filling in the missing data with some estimated value. If a large portion of data is unavailable, or specific samples or features contain a significant amount of missing data, it may be worth removing samples or features altogether. However, to avoid removing useful data, missing data can be estimated by taking the mean of the feature values, or the mode, if the feature is categorical. Of course, a mixture of both removing and imputing the values can be used as well.

During the data preprocessing phase, the researchers should also consider whether the features should be standardized or normalized. It is possible that one feature may unfairly bias the importance of other features. While in some cases such side effect could be desirable, it is often the result of inconsistent selection of the feature units. This issue

is often resolved by normalizing all features to be between 0 and 1 or standardizing the features so that they all have a mean of 0 and a standard deviation of 1.

Another problem that is often ignored is the lack of class balance in the dataset. For example, if a dataset contains 90% of class A samples then the model can be 90% accurate by always predicting that a sample belongs to class A. This issue can be solved by adjusting the accuracy or error metric, but sometimes it is better to address it at the data level instead. There are two main approaches to balancing a dataset: (1) removing from the over-sampled class and (2) adding to the under-sampled class. Removing from the over-sampled class can be as trivial as randomly selecting and removing samples within that class. However, this method risks losing important information from the dataset.

Adding data to an under-sampled class could be achieved by either sampling with replacement or by generating synthetic data points. Sampling with replacement simply duplicates randomly selected samples of under-sampled class. The **S**ynthetic **M**inority **O**ver-**S**ampling **T**echnique (SMOTE) is a method for generating synthetic data points. This approach randomly creates samples with features that would exist within the boundaries of the true class data [52].

Sometimes, there are too many features present in a given dataset. An overabundance of bad features or many correlated features could hurt the performance of a model by allowing it to learn based on misguided data. It can also lead to a significant slowdown of the algorithm. To mitigate the problem of too many features, the preprocessing can employ univariate feature selection or principal component analysis

(PCA). The univariate feature selection approach selects the $n$ best features based on certain statistical test or based on the amount of variance. The advantage to this approach is that the values of the remaining features will not change, those considered uninformative will simply be removed. However, eliminating features may lead to losing useful information. PCA is more robust against losing information as it transforms the original features into a few $(n)$ linear combinations of the features (principal components) with the highest amount of variance [53]. By using orthogonal transformations, PCA can convert correlated features into linearly uncorrelated principal components. Overall, the intentions of modeling and the amount of redundancy in the dataset may decide which, if any, approaches are taken.

**Metrics and workflows.** Model accuracy is an important metric for comparison of model performance. Assume that the given dataset consists of two classes: class positive and class negative. In this example, summarized in figure 2, the elements can be classified as follows:

- True positive $(TP)$ - number of positive elements classified as positive,

- False negative $(FN)$ - number of positive elements classified as negative,

- False Positive $(FP)$ - number of negative elements classified as positive, and

- True Negative $(TN)$ - number of negative elements classified as negative.

| | Positive Condition | Negative Condition |
|---|---|---|
| **Predicted Positive** | True Positive (TP) | False Positive (FP) |
| **Predicted Negative** | False Negative (FN) | True Negative (TN) |
| Accuracy $= \frac{TP+TN}{TP+FN+FP+TN}$ | | |

*Figure 2*. The possible outcomes for a binary classification prediction.

Using the above notation, we can define model accuracy as

$(TP + TN)/(TP + FN + FP + TN)$. The model accuracy specifies the proportion of

predictions that are correct. Other considered evaluation metrics are model's precision

$TP/(TP + FP)$, recall $TP/(TP + FN)$, false discovery rate $FP/(FP + TP)$ and/or an F1

score $2TP/(2TP + FP + FN)$. Model accuracy, along with other metrics, may help truly

understand how well a model is performing. For example, the F1 score can be thought of

as the harmonic mean of the model's precision and recall, creating a more robust metric.

The usage of the overall dataset in training and testing models has an important

impact on the observed performance metrics. The training and testing models can be

viewed as how well the model is able to train on a given dataset and how well the model

is able to classify data it has never seen, respectively. Model bias defines how well the

model is trained: a high bias means it did not train well to the dataset and a low bias

means it has trained well to the dataset. Model variance explains how well the model

performs a classification task: high variance means new data is not classified well and

low variance means new data is classified well. To determine model bias, we can simply

evaluate the model on the training data. To determine model variance, we must use a

dataset that the model was not trained on. To compute model's bias and variance the dataset could be randomly split into two subsets, creating a training set and a testing set.

The cross-validation approach trains and tests the model while rotating the data in the training and testing set such that all data will be used in the testing set once. Cross-validation is typically considered a better approach as it allows for a statistical comparison between models and gives a way to find performance anomalies on specific portions of the dataset. Please note that data-preprocessing should be computed only on the testing set and the results should be projected onto the testing set. For example, if missing data is estimated, they should only be estimated using data from the training set, otherwise, information may be leaked into the testing set. Information leaking can bias the scoring functions to overestimate a model's performance.

**Model analysis.** After the best performing model is selected, the details of a model's functionality may be of interest. Some model details that could be useful are the weighted importance of each variable, or how the different values a variable could obtain influence model predictions. Investigating variable or feature importance is trivial when working with linear based models because the absolute value of the weights associated with the features gives a clear indication as to which features change the prediction more heavily than others. The same is true for decision trees, where the Gini index or the information gain can show how important a feature is within the tree. To understand how a feature's value influences a model's sensitivity, partial dependence of various features can be plotted and analyzed. The researchers observe how the output changes across the feature domain when the model is fed the average values of all other features, or one

feature value is changed from certain minimum and maximum. The unified framework, SHAP (SHapley Additive exPlanations) assigns each feature an importance value for a given prediction. This is a model agnostic approach for performing exploration techniques [54], which allow us to better understand the model rather than leaving it a black box.

**Predictive Models**

There are many modeling approaches for predicting and evaluating potential drug activities [55]. Using empirical data, one could take a modeling approach that does not rely on fundamental knowledge of the system. This can be reliable as many drug phenomena are complex and not yet well understood. Two common empirical modeling methods are ligand-based, and target-based approaches [55]. In ligand-based approaches, shown in Figure 3, known active and inactive compounds can be used to detect important structural features for the activity in question. In target-based approaches, the structural features of the enzyme or protein in question can be used to detect potential ligand interactions. Both methods can benefit from machine learning techniques. Target-based approaches can use regression methods to learn and model various enzyme-ligand docking conformations. Ligand-based approaches can use regression or classification techniques to predict a ligand's activity towards the enzyme or the phenomena being modeled.
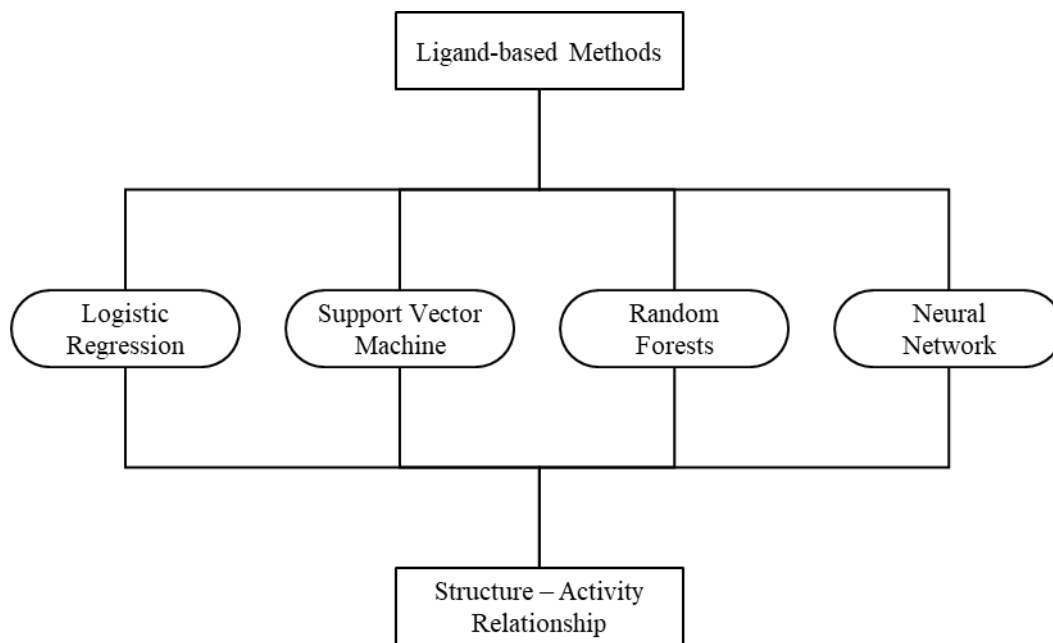
*Figure 3*. Ligand-based modeling approaches.

**Chapter 3**

**Biological Role of Cytochrome P450 Enzymes**

Cytochromes are an important class of proteins for all biological species.

Containing heme as a cofactor, they are involved in electron transport reactions and act as

enzymes in reduction and oxidation reactions. Cytochrome proteins were first discovered

in 1884 [56], but didn't receive the cytochrome name until the 1920s [57]. There are four

major types of cytochrome proteins, which can be distinguished by analytical chemistry

techniques. Various spectroscopy methods enable analyzing the exact structure of the

heme group, analyzing inhibitor sensitivity, and analyzing reduction potential [58]. This

chapter focuses on the cytochrome P450 superfamily of enzymes, which named after the

characteristic peak formed by absorbance of light at wavelengths near 450 nm, when the

heme iron is reduced to carbon monoxide. These enzymes, specifically the 3A4 variant,

play an important role in oxidative metabolism and are primarily involved in

steroidogenesis and detoxification [58].

The cytochrome P450 superfamily are categorized as cytochromes that act as

monooxygenases, reducing oxygen to a hydroxyl group for incorporation into substrates

[59]. Cytochrome P450 enzymes are found in all domains of life and consist of more than

2000 distinct proteins across different species [60]. There has been a total of 57 human

genes described within the P450 superfamily with substrates including sterols, fatty acids,

eicosanoids, vitamins, and xenobiotics [61]. The oxidation of xenobiotics is particularly

important for reducing toxicity that is involved with the incorporation of foreign

compounds, which may often be pharmaceuticals. The metabolism, of an estimated 75%

25

of drugs, is mediated by P450 enzymes, emphasizing the importance of this family of proteins in drug design [61]. According to their intracellular localization, P450 enzymes may be classified into: 1) *microsomal cytochrome P450,* which are present mainly in the microsomes of liver cells and represents about 14% of the microsomal fraction of liver cells or 2) *mitochondrial cytochrome P450,* which are present in mitochondria of many tissues but is particularly abundant in the liver and steroidogenic tissues such as adrenal cortex, testis, ovary, placenta, and kidney.

**Metabolic Reactions Mediated by CYP3A**

The P450 protein superfamily can be further broken down into families, subfamilies, and finally the specific gene products. The most common and the most versatile member of the cytochrome P450 family of oxidizing enzymes is CYP3A4. Like all other members of this family CYP3A4 is a hemoprotein which is involved in drug metabolism. In humans, the CYP3A4 protein is encoded by the *CYP3A4* gene [62]. This gene is part of a cluster of Cytochrome P450 genes and, as visible in Figure 4, is positioned at chromosome 7q22.1 [63].
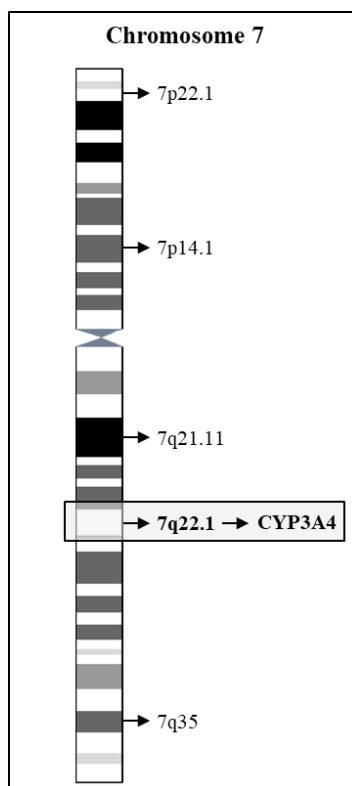
*Figure 4*. Human chromosome 7 with cytogenetic bands displayed.

According to Wienkers at al [64] the bulk of the metabolism of known drugs in humans is mediated by cytochrome P450 (CYP) enzymes. Among those, the majority of drug oxidations (46%) were carried out by members of the CYP3A family. CYP3A catalyze many reactions involved in drug metabolism as well as in synthesis of cholesterol, steroids, and other lipids components. Enzymes of CYP3A family mainly found in the liver and in the intestine. Their role is oxidation of small foreign organic molecules (xenobiotics), such as toxins or drugs, so that they can be removed from the body. This makes CYP3A enzymes remarkably important in drug design and pharmaceutical research.

The CYP3A subfamily consists of 4 genes: CYP3A4, CYP3A5, CYP3A7, and CYP3A43. Of the four, CYP3A4 tends to have the highest expression and the greatest affinity for metabolizing pharmaceutical drugs [65]. CYP3A4 has a large and flexible active site, and can bind with many large lipophilic compounds, allowing for substrates like immunosuppressants, antibiotics, antidepressants, opioids, and many others [65]. The CYP3A4 protein is an important enzyme in first-pass metabolism, where some amount of a drug may be metabolized before entering systemic circulation [66]. A snapshot of the KEGG entry for CYP3A4 is shown in Figure 5. A sample oxidative reaction catalyzed by CYP3A4 is shown in Figure 6, where a xenobiotic molecule binds with oxygen, creating a less toxic drug and/or a drug that is easier to metabolize downstream.

| Entry | EC 1.14.14.55           Enzyme |
|---|---|
| Name | quinine 3-monooxygenase;<br>CYP3A4 (gene name) |
| Class | Oxidoreductases;<br>Acting on paired donors, with incorporation or reduction of molecular oxygen;<br>With reduced flavin or flavoprotein as one donor, and incorporation of one atom of oxygen into the other donor<br>[BRITE hierarchy] |
| Sysname | quinine,[reduced NADPH---hemoprotein reductase]:oxygen oxidoreductase |
| Reaction(IUBMB) | quinine + [reduced NADPH---hemoprotein reductase] + O2 = 3-hydroxyquinine + [oxidized NADPH---hemoprotein reductase] + H2O [RN:R05727] |
| Reaction(KEGG) | R05727<br>[Reaction] |
| Substrate | quinine [CPD:C06526];<br>[reduced NADPH---hemoprotein reductase] [CPD:C03024];<br>O2 [CPD:C00007] |
| Product | 3-hydroxyquinine [CPD:C07344];<br>[oxidized NADPH---hemoprotein reductase] [CPD:C03161];<br>H2O [CPD:C00001] |
| Comment | A cytochrome P-450 (heme-thiolate) protein. |
| History | EC 1.14.14.55 created 2000 as EC 1.14.13.67, transferred 2017 to EC 1.14.14.55 |
| Orthology | K17689   cytochrome P450 family 3 subfamily A polypeptide 4 |
| Genes | HSA: 1576(CYP3A4)<br>PTR: 463582(CYP3A4)<br>PPS: 100976899<br>GGO: 101149513<br>PON: 100436782<br>NLE: 100593175<br>MCC: 678670(CYP3A4)<br>MCF: 102144258(CYP3A8)<br>CSAB: 103246766<br>RRO: 104673585<br>» show all<br>[Taxonomy] |

*Figure 5*. KEGG entry for the *CYP3A4* gene.
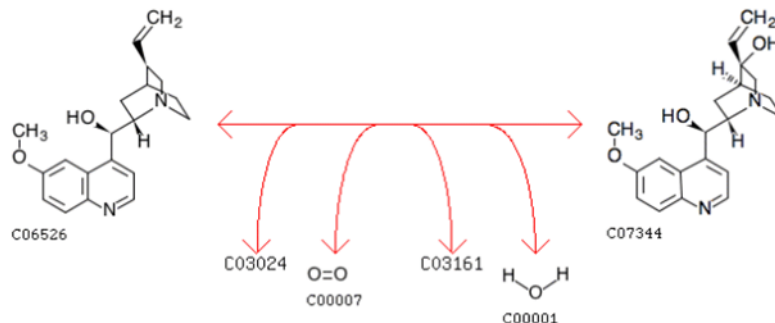
*Figure 6*. Oxidation reaction catalyzed by CYP3A4. Obtained from KEGG R05727.

**Potential for Drug-Drug Interactions**

The role of metabolic reactions catalyzed by CYPs is to transform the xenobiotic substances to harmless and excretable metabolites [55]. Issues arise in the situation when xenobiotic substances are transformed into a toxic metabolite or the speed of metabolic clearances is significantly altered. The reliance on P450 enzymes to oxidize and facilitate the removal of dugs opens the potential for issues in drug metabolism. Two major issues in drug metabolism are 1) the bioavailability of a drug, i.e., how quickly the drug is metabolized and eliminated, and 2) the accumulation of the drug in the system, which could cause toxic side effects. Issues in bioavailability may arise if the P450 enzymes are induced by another drug, while issues in drug toxicity could occur if the P450 enzymes are inhibited by another drug. The inhibition of P450 enzymes is often the cause of many adverse drug reaction as these drugs are metabolized at slower rates and can begin to accumulate at levels that are toxic [61]. The ability for one drug to affect another drug's metabolism is referred to as a drug-drug interaction. This interaction is often linked to the inhibition or induction of a P450 enzyme.

30

**Interferences with the Action of the CYP3A4 Enzyme**

While many drugs are metabolized by CYP3A4 mediated reactions, there are also some drugs which are *activated* by the enzyme. Some substances, such as grapefruit juice and some drugs, interfere with the action of CYP3A4. These substances will therefore either amplify or weaken the action of the drugs that are modified by CYP3A4, potentially causing them to exceed the minimum toxicity levels and create adverse side effects. Inhibition can happen in one of three ways, 1) competitive, in which the inhibitor competes with the substrate for the active site within the enzyme, 2) non-competitive, where the inhibitor reduces the activity of the enzyme, and 3) mechanism-based, where the inhibitor is metabolized into a reactive group that can form an irreversible bond with the enzyme. A well reported case of CYP3A4 inhibition was discovered by researchers who noticed an increase in bioavailability of many drugs in patients after consuming grapefruit juice. This effect is due to bergamottin, a furanocoumarin, found within grapefruit that acts as a mechanism-based inhibitor of CYP3A4 [67].

**Screening for CYP3A4 Inhibitors**

Understanding a drug's potential to inhibit CYP3A4 is a common step in drug development; many *in vitro* and some *in vivo* assays are conducted on potential drugs to determine if they have any inhibitory ability. The potential for a drug to inhibit CYP3A4 can be assessed *in vitro* by using a probe substrate. The probe must be a compound that is metabolized by the CYP3A4 enzyme. A commonly used probe is midazolam, a known substrate of CYP3A4, which performs similarly *in vitro* and *in vivo* [68]. The screening process works as follows. The probe is placed in a mixture of microsomes that contain

31

the CYP3A4 enzyme and other drugs that are being assessed. If the drug inhibits CYP3A4, the probe can no longer be metabolized at its standard rate. This will result in lower level of metabolites, which can be measured to determine if inhibition was taking place. To increase throughput, CYP3A4 inhibition assays typically use either a fluorescent probe or Liquid Chromatography Mass Spectrometry (LC-MS/MS) to monitor the rate of probe metabolism [64]. Recently, there has also been an effort in predicting CYP3A4 inhibition in silico, which could save time and resources in the lengthy and expensive drug development pipeline.

**Kinetics of CYP3A Meditated Reactions**

The kinetics of drug metabolism can be summarized by few computed values: Michaelis-Menten constants ($K_m$), maximal velocities ($V_{max}$), and intrinsic clearance ($CL_{int}$, $V_{max}/K_m$). Similarly, protein inhibition and kinetics can be understood through values such as the inhibition constant ($K_i$) and the inhibitory concentrations ($IC_{50}$). While this manuscript focuses on CYP3A4, many compounds that are substrates of CYP3A4 can also be metabolized by CYP3A5 [69]. The speed of metabolism varies by compound and drug class. For example, antifungals on average have a lower $CL_{int}$ than antivirals, with an exception for Itraconazole [69]. The strength of inhibition and induction also varies by compound, a list of compounds with varying degrees of inhibition or induction strength are displayed in Table 1.

32

Table 1
*Known Inhibitors and Inducers for CYP3A4*

| Compound Name | Strength | Type |
|---|---|---|
| Boceprevir | Strong | Inhibitor |
| Cobicistat | Strong | Inhibitor |
| Danoprevir and Ritonavir | Strong | Inhibitor |
| Elvitegravir and Ritonavir | Strong | Inhibitor |
| Grapefruit Juice | Strong | Inhibitor |
| Indinavir and Ritonavir | Strong | Inhibitor |
| Itraconazole | Strong | Inhibitor |
| Ketoconazole | Strong | Inhibitor |
| Lopinavir and Ritonavir | Strong | Inhibitor |
| Posaconazole | Strong | Inhibitor |
| Ritonavir | Strong | Inhibitor |
| Saquinavir and Ritonavir | Strong | Inhibitor |
| Telaprevir | Strong | Inhibitor |
| Tipranavir and Ritonavir | Strong | Inhibitor |
| Telithromycin | Strong | Inhibitor |
| Troleandomycin | Strong | Inhibitor |
| Voriconazole | Strong | Inhibitor |
| Clarithromycin | Strong | Inhibitor |
| Idelalisib | Strong | Inhibitor |
| Nefazodone | Strong | Inhibitor |
| Nelfinavir | Strong | Inhibitor |
| Aprepitant | Moderate | Inhibitor |
| Ciprofloxacin | Moderate | Inhibitor |
| Conivaptan | Moderate | Inhibitor |
| Crizotinib | Moderate | Inhibitor |
| Cyclosporine | Moderate | Inhibitor |
| Diltiazem | Moderate | Inhibitor |
| Dronedarone | Moderate | Inhibitor |
| Erythromycin | Moderate | Inhibitor |
| Fluconazole | Moderate | Inhibitor |
| Fluvoxamine | Moderate | Inhibitor |
| Imatinib | Moderate | Inhibitor |
| Tofisopam | Moderate | Inhibitor |
| Verapamil | Moderate | Inhibitor |
| Chlorzoxazone | Weak | Inhibitor |
| Cilostazol | Weak | Inhibitor |
| Cimetidine | Weak | Inhibitor |
| Clotrimazole | Weak | Inhibitor |

Table 1 (continued)

| Compound Name | Strength | Type |
|---|---|---|
| Fosaprepitant | Weak | Inhibitor |
| Istradefylline | Weak | Inhibitor |
| Ivacaftor | Weak | Inhibitor |
| Lomitapide | Weak | Inhibitor |
| Ranitidine | Weak | Inhibitor |
| Ranolazine | Weak | Inhibitor |
| Ticagrelor | Weak | Inhibitor |
| Apalutamide | Strong | Inducer |
| Carbamazepine | Strong | Inducer |
| Enzalutamide | Strong | Inducer |
| Mitotane | Strong | Inducer |
| Phenytoin | Strong | Inducer |
| Rifampin | Strong | Inducer |
| St. John's Wort | Strong | Inducer |
| Bosentan | Moderate | Inducer |
| Efavirenz | Moderate | Inducer |
| Etravirine | Moderate | Inducer |
| Phenobarbital | Moderate | Inducer |
| Primidone | Moderate | Inducer |
| Armodafinil | Weak | Inducer |
| Modafinil | Weak | Inducer |
| Rufinamide | Weak | Inducer |

**Bioanalytical Methods to Study CYP3A Mediated Metabolism**

The bioanalytical approaches for metabolism studies can be classified into three

main categories: 1) metabolite profiling and identification, which includes

biotransformation and structural analysis both *in vitro* and *in vivo*; 2) metabolic stability,

which includes profiling the kinetics both *in vitro* and *in vivo*, and 3) identification of

rate-limiting CYP enzymes *in vitro* only. The third approach is often used for drug

interaction studies which examine the influence of a drug substrate on CYP activity: 1)

CYP inhibition: single substrate or cocktail study in microsomes or in hepatocytes, and 2) CYP induction: nuclear receptor activation or cocktail studies in hepatocytes [70]–[74].

Traditional in vitro CYP450 inhibition assays typically target six isoforms of P450: CYP1A2, 2C9, 2C19, 2D6, 3A4, and 3A5, which are known to metabolize more than 90% of drugs. There are two major types of assays: 1) single-substrate assays using known P450 inhibitors and licorice root extract and 2) cocktail assays, also known as N-in-one assays [75].

Single-substrate assay typically evaluates the inhibition of a drug on one P450 isoform at a time. The cocktail inhibition assays can simultaneously evaluate the inhibition effects of drugs on up to 12 CYP450 isoforms. While cocktail assays are much more efficient than traditional single probe substrate approaches, they still have some disadvantages. They are much more complex and require significant investment into an assay's parameter optimization. Cocktail assay parameter optimization includes selection of enzyme protein concentration, minimization of probe substrate interactions, minimization of solvent effects, complicated detection of probe substrates and usage of a fast and sensitive ultrahigh pressure liquid chromatography (UHPLC) – tandem mass spectrometry (MS/MS) quantitative instrumentation. Typical experimental procedure requires potassium phosphate buffer (100 ml, 0.1 M, pH 7.4) containing 1.3 mM NADPH, 0.2 mg/ml human liver microsomes, and a cocktail of 10 probe substrates including midazolam, which is used for testing CYP3A4 activity [75]. Such experimental procedures are expensive and time-consuming, thus the development of accurate theoretical models to predict CYP450 activity is highly desirable.

**In Silico Prediction of CYP–Ligand Interactions**

       *In silico* models to predict characteristics of CYP–ligand interactions attempt to

link the structure and properties of ligand with the readouts from CYP mediated

metabolic transformations obtained during in vitro experiments. Such models enable the

researchers to exploit experimentally derived data and fill the gaps in understanding of

        (1) the affinity of ligand binding to specific CYPs;

        (2) predicting sites of metabolism;

        (3) prediction of inhibition characteristics of test molecules [55].

**Inhibition Model Review**

       The task of predicting CYP3A4 inhibition in silico has been examined before with

the help of various theoretical models. A summary of these studies is presented in Table

2. In 2011, Cheng et al. was able to develop a model based on the support vector machine

approach, which achieved a cross-validation accuracy of 0.775 [76]. The models utilized

MACCS fingerprints determined from the compound structures. In addition, Sun et al.

achieved a cross-validation accuracy of 0.811, also using a support vector machine model

in 2011. The primary difference between these approaches was in the feature set and the

use of custom atom type descriptors [77]. More recently, in 2018, Li et al. used an auto

encoder deep neural network that was able to achieve a cross-validation accuracy of

0.850. This study  used a set of features generated using the PaDEL software in addition

to the compounds' PubChem fingerprints [78]. In 2019, Wu et al. extended on previous

work  and developed a new XGBoost approach that achieved a cross-validation accuracy

of 0.860 [79].

Table 2

*CYP3A4 Inhibitor Prediction Models in Literature*

| Authors | Date | Model | Accuracy | # Training | Feature Set |
|---|---|---|---|---|---|
| **Wu et al.** | 2019 | XGBoost | 0.860 | (3070, 5985) | PaDEL1&2D, PubChem FP |
| **Wu et al.** | 2019 | Gradient Boosting DT | 0.851 | (3070, 5985) | PaDEL1&2D, PubChem FP |
| **Wu et al.** | 2019 | Deep NN | 0.839 | (3070, 5985) | PaDEL1&2D, PubChem FP |
| **Wu et al.** | 2019 | Convolutional NN | 0.833 | (3070, 5985) | PaDEL1&2D, PubChem FP |
| **Wu et al.** | 2019 | Random Forests | 0.828 | (3070, 5985) | PaDEL1&2D, PubChem FP |
| **Li et al..** | 2018 | Multitask AE-DNN | 0.850 | (3070, 5985) | PaDEL1&2D, PubChem FP |
| **Li et al.** | 2018 | Singletask AE-DNN | 0.845 | (3070, 5985) | PaDEL1&2D, PubChem FP |
| **Lee et al.** | 2017 | Laplacian Naïve Bayes | 0.799 | (1193, 2221) | VolSurf+ |
| **Su et al.** | 2015 | Support Vector Machine | 0.756 | (5177, 7456) | PaDEL3D, Mold |
| **Su et al.** | 2015 | C5.0 Decision Tree | 0.733 | (5177, 7456) | PaDEL3D, Mold |
| **Sun et al.** | 2011 | Support Vector Machine | 0.811 | (2334, 4466) | Custom Atom types |
| **Cheng et al.** | 2011 | Support Vector Machine | 0.775 | (4637, 6899) | MACCS |
| **Cheng et al.** | 2011 | Stack: SVM; K-NN | 0.767 | (4637, 6899) | MACCS |
| **Cheng et al.** | 2011 | Stack: SVM; Naïve Bayes | 0.752 | (4637, 6899) | FP4 |

# Chapter 4

## Representation, Modeling, and Featurization of Chemical Compounds

Much of chemical exploration and research relies on computational tools and techniques to represent and study compound structures. However, it is not trivial to efficiently represent a compound in its entirety. At a basic level, we can view a compound as a graph representing atoms as nodes and bonds as vertices. However, the graph must ensure that the differences in compound configuration are also clearly distinguished. For example, features such as single, double, triple, and ionic bonds, the stereochemistry of an atom, or aromaticity could be difficult to represent. Creating a graph-based representation that covers all of these aspects is not necessarily impossible. In fact, many of these features are often used to visualize compounds [80]. However, existing representations often lack the ability to store structural information or to gather complex information from of a compound in an efficient manner. As summarized in Table 3 there are other techniques that can be used to represent compounds, such as the simplified molecular-input line-entry system (SMILES), generating molecular fingerprints, or simply using their molecular descriptors. These techniques allow us to describe complex compounds numerically, so that a computer can easily and efficiently process them.

Table 3

*Featurization Methods Useful for Predictive Modeling in Drug Discovery*

| Featurization | Type | Description | Ref |
|---|---|---|---|
| SMILE | String | Alphanumeric characters representing the 2D structure | [2] |
| MACCS | Fingerprint | 960 or 166 structural keys for important substructures | [5] |
| PubChem FP | Fingerprint | 881 structural keys used by the PubChem database | [6] |
| Extended-Connectivity | Fingerprint | Sets bits based on the structure in a radius of focused atoms | [7] |
| PaDEL | Descriptor | Calculate 797 chemical features including 1D, 2D, and 3D | [8] |
| Mordred | Descriptor | Calculate more than 1800 2D and 3D descriptors | [9] |

**Simplified Molecular-Input Line-Entry System (SMILES)**

The simplified molecular-input line-entry system (SMILES) is a widely used representation of compounds. As its name suggests, this method attempts to model a particular compound using a single line of characters. SMILES relies on a set of parsing rules that allow for an unambiguous reconstruction of a compound's structure. This approach was first introduced in the 1980s [81]. While currently there are other formats, the fundamentals of a SMILE compound representation as a string remain unchanged. For example, atoms are denoted by an uppercase character abbreviation, e.g., carbon is represented as C, nitrogen as N, fluorine as F, chlorine as Cl, etc. Atoms can also be extended to include their charge, as applicable. For instance, a hydroxide may be depicted as [OH-] to indicated that the OH group contains a formal negative charge and a titanium(IV) atom may be depicted as [Ti+4] to indicate a formal positive charge of 4. To represent bonds, the characters ".", "-", "=", "#", "$", ":", can be used for a non-bond, single bond, double bond, triple bond, quadruple bond, and aromatic bond. Bond

stereochemistry can be denoted using "/" and "\" characters which represent single bonds

connected to a double bonded pair of atoms. For example, F/C=C/F shows the *trans*

configuration while F/C=C\F represents the *cis* configuration. Often, standard single

bonds are assumed and omitted, thus, it is rare to see "-" within a SMILE string.

To store ring structures in a SMILE string, one arbitrary bond within the ring is

broken and an integer value is placed instead. At the end of the ring, the integer value is

repeated to show that the ring has closed. For example, cyclohexane, a ring of 6 single

bonded carbons, would be represented as C1CCCCC1. Integers can be incremented to

demonstration additional rings in the compound. Aromatic rings can be shown in a

multitude of styles. However, the common syntax is to convert the atom symbols into

lowercase characters. For example, using this approach, benzene, a 6-carbon aromatic

ring, can be shown as c1ccccc1.

SMILE format also allows representing multiple branching points using

parenthesis. Specifically, the atom where the branch starts is followed by the branch

portion of the compound surrounded by an open and close parenthesis. For example,

isopropyl alcohol is denoted by CC(C)O, indicating that the third carbon branches off of

the second and is not bonded to the oxygen. Branch bonds that must be shown, such as a

double bond, are denoted within the parentheses. For example, acetone is represented as

CC(=O)C to indicate that the oxygen is double bonded to the second carbon, while the

third carbon is also bonded to the second carbon rather than the oxygen.

A final consideration in the structural representation of a compound is the chiral

configuration of a tetrahedral carbon, where the positional order of the four bonds may be

chemically important. There are a several ways to represent chirality. It is often denoted by a clockwise or counter-clockwise bond order with either @ (counter-clockwise) or @@ (clockwise) characters in a SMILE string. For example, the amino acid L-alanine and D-alanine are represented as N[C@@H](C)C(=O)O and N[C@H](C)C(=O)O, respectively.

CANGEN was the first standard algorithm for creating canonical SMILE strings to ensure that each compound has a unique SMILE representation [82]. Recently new algorithms that achieve better results were developed. For example, the public chemical database ChEMBL uses Accelrys's Pipeline Pilot software to generate their canonical SMILE strings [83]. Overall, a standardized canonical SMILE algorithm can ensure uniqueness of compound representation, which is vital when searching for specific compounds, similar compounds, or compounds that contain a particular sub-structure. Canonical SMILE strings can also provide a normalized structure placement when mining structural information from many SMILE strings.

**Molecular Fingerprints**

Another approach to representing a compound, especially for similarity scoring or for data mining, is *molecular fingerprints*. The essence of a fingerprint is that the compound is represented as a variable length string of bits. Each bit identifies the presence or lack of certain piece of information from of the compound. There are many approaches to generating fingerprints, notably, substructure keys-based, topological or path-based, and circular [84].

Substructure keys-based fingerprints are based on the idea of chemical motifs or functional groups. Each bit in the compound's notation indicates the presence or absence of a given substructure. This can typically be done with SMARTS patterns, where a SMILE string is parsed for a particular character sequence, similar to regular expressions. Examples of substructure keys-based fingerprints include MACCS fingerprints, which has 960 or 166 structural keys for important and popular substructures in drug discovery [85], and PubChem fingerprints, which contains 881 structural keys used by the PubChem database for similarity searching [86]. Structure based fingerprints are important for tasks that rely on the presence or absence of particular functional groups in a compound, such as, filtering compounds that may be reactive based on the presence of one or more reactive chemical motifs.

Topological or path-based fingerprints do not rely on predefined keys and a set number of bit positions, instead, they parse molecular fragments along bond paths. While moving along a bond path, these methods hash the path into a new bit position. The location of that bit within the bit string is determined by the hashing function. These methods can be customized to generate a string of any number of bits by adjusting the hashing function. An example of topological fingerprints is the Daylight fingerprints that contain 2048 bits. The downside to this approach is a possibility of collision, which may place different substructures at the same bit position, possibly losing information.

Circular fingerprints, also called extended-connectivity fingerprints (ECFPs), are similar to topological fingerprints in that they hash the chemical structures. However, bond paths are not traveled down the molecule, instead, the structure within a given

radius of an atom is searched. ECFPs have the advantage that they can be calculated quickly and can represent any number of molecular features, while also capturing stereochemical information [87]. While the bits may be harder to utilize in substructure searching, this approach could provide much more information about a compound for data mining applications.

Molecular fingerprints can be an efficient and highly versatile representation of compounds, allowing for computed similarity scores between compounds, searching databases with substructure queries, extracting chemical and structural information from compounds, and transforming compounds into useful features for data mining pipelines. Many of the fingerprint techniques can be used together with the SMILES representation as both approaches have their advantages. Therefore, the fingerprints approach is not necessarily a replacement of SMILE strings.

**Molecular Descriptors**

Representing a compound as a set of molecular descriptors is another approach that computes chemically relevant data. This approach can represent complex features such as a molecule's solubility or acidic properties and often formats features in ways that are easier to interpret compared to fingerprints. Molecular descriptors could range from something as simple as a molecular weight to structural information such as the molecular surface area and electrostatic properties such as polar surface area. There are many packages that can calculate numerous molecular features for a given compound. In this work we will focus on the following two popular software packages: PaDEL-Descriptor and Mordred.

PaDEL-Descriptor is an open source software that can generate 797 chemical features including 1D, 2D, and 3D descriptors [88]. The types of descriptors calculated by PaDEL include constitutional, topological, geometric, electrostatic, hydrophobic, steric, and quantum chemical. PaDEL can be regarded as one of the better options because it is free, open source, supports a majority of molecular file formats, and is fast. The Mordred software package is based on PaDEL-Descriptor and it can calculate more than 1800 2D and 3D descriptors. Mordred is open source and free just like PaDEL. However, it is almost twice as fast as PaDEL [89]. The Mordred also can handle large molecules which is often an issue for other packages. Overall, the PaDEL-Descriptor software and the Mordred software can compute a plethora of varied but relevant molecular descriptors from the structure of a compound.

**Numeric Featurization**

While humans can understand conceptual information, such as a compound being acidic or basic, a computer relies on raw numbers and therefore, molecular concepts must be converted to some numerical form [90]. This process, however, can be tricky when avoiding non-informative or redundant features that may act as noise. There are several ways to handle the procedure of feature selection: removing features that are highly correlated with others, dropping features with low variance, or using regularization techniques.

A conversion of categorical data into numerical values can arise another issue. A simple approach to assign a number to each possible category could bias the computer into seeing a non-existent order within the categories. For example, labeling acid as 0,

base as 1, and neutral as 2 may cause the computer to interpret that an acid is less than a base, which is less than a neutral compound. Of course, this ordered relationship does not exist in reality. Instead, categorical features can be one-hot encoded to create a series of bits that simply represent if a feature is present or not. A one-hot encoded feature is treated as multiple numerical features, one bit for each possible case in the category. If a particular case is present in the category, then that bit will become a 1, otherwise, it will become a 0. So, for the acid-base category, two numeric features will be generated, one bit (either 0 or 1) for an acid and another bit (either 0 or 1) for a base. If a compound is neutral, then both the acid and base bits will be set to 0, indicating the third category.

Representing compounds using featurization can be difficult and may require domain knowledge for the specific task. Fortunately, molecular fingerprints and the molecular descriptor approaches to compound representation has been curated and tested by many experts within the chemical and pharmaceutical fields and have been shown to work well.

# Chapter 5

## Cytochrome P450 3A4 Inhibitor Modeling

*In silico* methods for determining a compound's potential to inhibit CYP3A4 could greatly reduce the time and resources spent on drug candidate development. Inhibition prediction models can be built by selecting relevant features from compound structures and correlating them to experimentally observed inhibition outcomes using the data generated by *in vitro* assays. Machine learning techniques that analyze many data samples can discover patterns in the feature space and make adequate predictions of the inhibition potential for a new candidate molecule. The best model for separating already known samples and generalizing the unknown data can be selected from the comparison of various machine learning approaches and modeling techniques. The process of creating a CYP3A4 inhibition model must consider many aspects, including the choice of the dataset used for training, the procedure of molecular feature extraction from compounds, the selection of the machine learning algorithm for model generation, etc. Our workflow for building predictive models is presented in Figure 7.
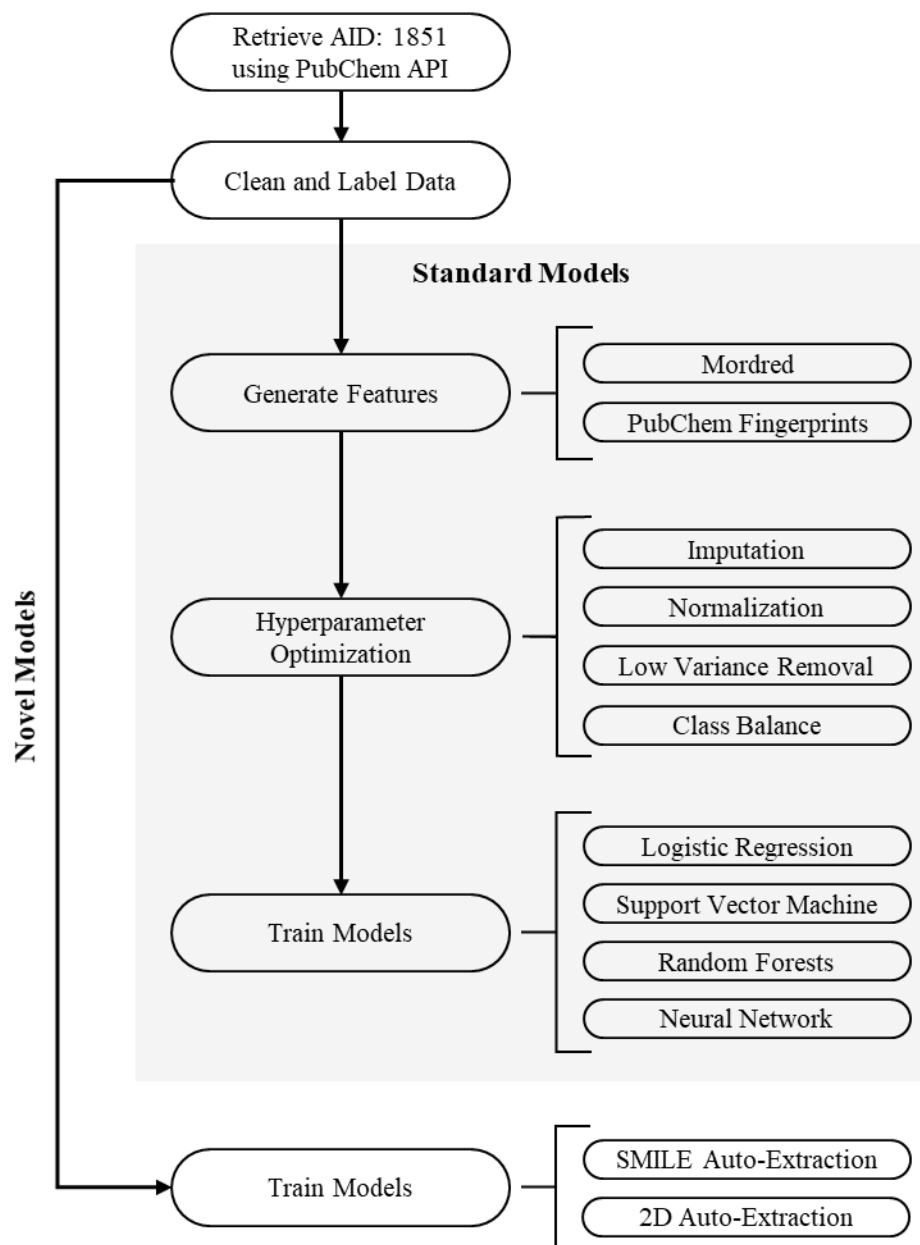
*Figure 7*. Methodological pipeline used to create predictive models.

## Dataset Curation

A vital part of any successful prediction model is the data used to train it. The

quality of training data directly influences the accuracy of model predictions. PubChem

AID: 1851 is a publicly available and frequently used dataset. It is often considered to be the standard for CYP3A4 inhibition modeling [91]. Many compounds (n=17,143) were studied *in vitro* using a luminescence-probe approach. CYP3A4, as well as many other CYP enzymes, were used to catalyze the dealkylation of pro-luciferin substrates to luciferin. Using luciferase detection reagent, luciferin could be measured with luminescence. Various concentrations of the compounds were tested to determine the compounds' IC50 for the studied CYP enzymes from measured concentration−response curves. In addition, the data on response curves contains curve classification that specifies the completeness of the reaction and efficacy of the compound. PubChem provides an activity score and an activity outcome, where the compounds are labeled as active, if their activity score is 40 or more. Inactive compounds have a score of 0, and inconclusive results if their activity score is less than 40 but greater than 0.

PubChem enables a publicly available RESTful API providing the access to data tables corresponding to a user-selected assay. Substance IDs and specific table features, like activity scores, can be retrieved over HTTPS protocol using the PubChem API. CYP3A4 data were downloaded in a JSON[1] format. A correspondent list of canonical SMILES strings were obtained by matching activity data records using compound IDs. All compounds, activity scores, and curve classifications in the AID: 1851 assay were retrieved from PubChem, matched with SMILES structures, and filtered using an

---

[1] JavaScript Object Notation

automated Python script. Please see Appendix A: *pubchem.py* and Appendix B for our source code. The compounds were classified as follows:

- inhibitors - compounds with an activity score $\geq 40$ and a curve classification of -1.1 (complete curve; high efficacy), -1.2(complete curve; partial efficacy), or -2.1 (partial curve; high efficacy),

- non-inhibitors - compounds with an activity score of 0 and a curve classification of 4 (undefined),

- inconclusive -- compounds that did not fall under either classification. Compounds classified as inconclusive were excluded from the dataset.

PubChem fingerprints were generated by the PaDEL software for all included compounds. The Mordred package was employed to generate molecular features. Our final feature set included both PubChem fingerprints and Mordred generated molecular features. In total, 10,832 compounds were selected for the training dataset. For each compound in the dataset 1,826 molecular features were calculated and 881 fingerprint bits were extracted. The source code for molecular descriptor generation is shown in Appendix A: *featurized.py*.

**Inhibitor Models**

Four machine learning approaches were evaluated for the task of building a prediction model for CYP3A4 inhibition: logistic regression, random forests, support vector machine, and neural network. A logistic regression approach has the advantages of being the fastest to train and use. Also, a logistic regression model is relatively easy to modify and interpret. In comparison, random forests and support vector machine are

more difficult to use and modify. However, these approaches often perform well on many different tasks, and can be better generalized to new data. A neural network approach offers the most potential in learning more complicated features in addition to the ones already present. While the neural network approach can be much harder to interpret and more complicated to use, it offers a significant increase in performance for many machine learning tasks.

**Model creation.** All four machine learning models were implemented in Python using the scikit-learn package [92], please see the details in Appendix C. The training data went through several pre-processing steps:

1) missing data was imputed using the mean value of the relevant feature,

2) features were normalized to fall within a range of 0 to 1,

3) features with a variance less than a certain threshold were removed, and

4) classes were either balanced using a SMOTE or an under-sampling approach, or classes were left unbalanced.

All pre-processing steps were conducted in a pipeline corresponding to a cross-validated train-test split. The pre-processors were constructed using training dataset and evaluated using the testing dataset.

The optimization of each model contained multiple hyperparameters. For example, the logistic regression algorithm's regularization factor can affect the weight of each feature. For the random forests model, the number of estimators in the ensemble and the fraction of features considered for each estimator can significantly alter the bias and variance of the final model. For the support vector machine model, the regularization

parameter (lambda) and gamma can alter the weights of the features and the amount of influence one sample has. Finally, for the neural network approach, the number of layers and the number of nodes in each layer can greatly impact the bias and variance of the model. Regularization can alter the weighting of each node, with high regularization corresponding to a lower amount of strongly weighted nodes. All models went through extensive tuning and optimization of the above model hyperparameters, as well as for various pre-processing steps, using the open source hyperparameter optimization framework Optuna [93].

**Model performance and comparisons.** To evaluate and compare the above models, a 3-fold cross-validation approach was conducted on each of them. Prediction accuracy was calculated and used to score each model. The best average cross-validation accuracies for each model approach were the following:

- logistic regression achieved an accuracy of 0.832,

- random forests achieved an accuracy of 0.834,

- support vector machine achieved an accuracy of 0.819, and

- neural network achieved an accuracy of 0.823.

Overall, all models achieved a similar performance on accuracy that was consistent with the results seen in literature. The cross-validated accuracies for each of the 3-folds are compared in Figure 8. Random forests and logistic regression have similar performances, though, random forests had a larger variance between the three cross validated folds. Support vector machine had the worst performance of the four approaches and a large variance between fold accuracies. The neural network approach

51

had the most consistent accuracy measures between the three folds; however, these accuracies were worse than those of both random forests and logistic regression. The models performed best when the variance threshold was set lower, allowing for more features to pass through to the model. The best class balancing technique for random forests, support vector machine, and neural network was SMOTE, while logistic regression preferred no balancing of classes. A more detailed overview of all hyperparameters for each modeling approach is shown in Table 4.
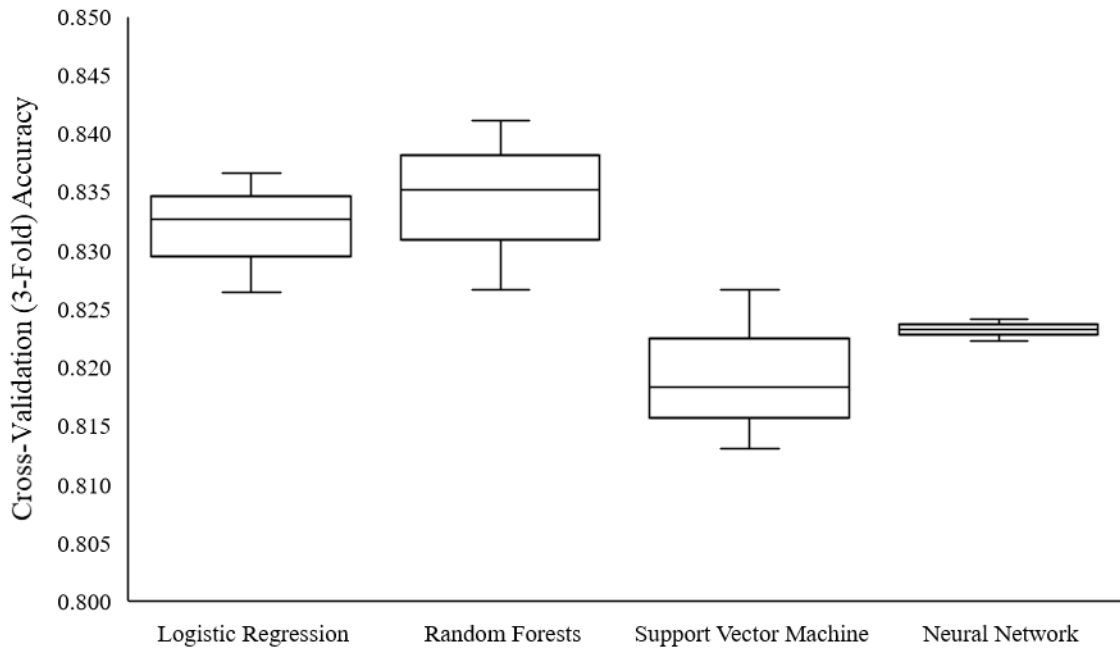


*Figure 8*. Cross-validated (3-fold) accuracies for all standard modeling approaches.

Table 4

*Hyperparameter Optimization for Standard Models*

| Model | Hyperparameter | Range Considered | Best Value |
|---|---|---|---|
| **Logistic Regression** | Balance | {None, Under-sample, SMOTE} | None |
| | Variance Threshold | [0, 0.25] | 0.00164 |
| | C = 1 / regularization | $[1\text{x}10^{-5}, 1\text{x}10^{5}]$ | 44772.5 |
| **Random Forests** | Balance | {None, Under-sample, SMOTE} | SMOTE |
| | Variance Threshold | [0, 0.25] | 0.02322 |
| | Max Features | [0.01, 1] | 0.36402 |
| | # Estimators | [1, 1000] | 968 |
| **Support Vector Machine** | Balance | {None, Under-sample, SMOTE} | SMOTE |
| | Variance Threshold | [0, 0.25] | 0.06889 |
| | C = 1 / regularization | $[1\text{x}10^{-5}, 1\text{x}10^{5}]$ | 691.286 |
| | Gamma | $[1\text{x}10^{-5}, 1\text{x}10^{5}]$ | 0.06045 |
| **Neural Network** | Balance | {None, Under-sample, SMOTE} | SMOTE |
| | Variance Threshold | [0, 0.25] | 0.01611 |
| | Alpha | $[1\text{x}10^{-10}, 1\text{x}10^{10}]$ | 2.28473 |
| | Layers | [1, 3] | 2 |
| | Units in Layer 1 | [1, 300] | 140 |
| | Units in Layer 2 | [1, 300] | 181 |

**Model Exploration**

Exploring which features were most important to a model can bring insight into the model's predictive power and can showcase patterns between specific features and labels. While some modeling approaches, such as logistic regression, can be easily explained by viewing the weights of each feature, other approaches are harder to comprehend by simply examining the weight values. The SHAP framework can be employed to identify important features in a model, observe how fluctuations in feature values effect the model output, and determine how features altered a model's prediction for a given sample. We implemented a series of model exploration tools, presented in Appendix D.

The top five features for each model are shown Figure 9. In all of the explored models the two most important features were the atom-bond connectivity index (ABC) [94] and the centered Moreau-Broto autocorrelation of lag 5 weighted by Allred-Rochow electronegativity (ATSC5are) [95]. Additionally, all models had Graovac-Ghorbani atom-bond connectivity index (ABCGG) [96] and centered Moreau-Broto autocorrelation of lag 6 weighted by Allred-Rochow electronegativity (ATSC6are) within their top five most important features. Logistic regression and random forests both shared the same top 5 features and also happen to be the best performing models. The support vector machine model had 5-membered ring count (n5Ring) as a unique most important feature, while the neural network model had shortest path diameter of adjacency matrix (SpDiam_A) as a unique most important feature. We observed that higher ABC values indicate a higher probability that the compound will be classified as an inhibitor in all of the explored models.
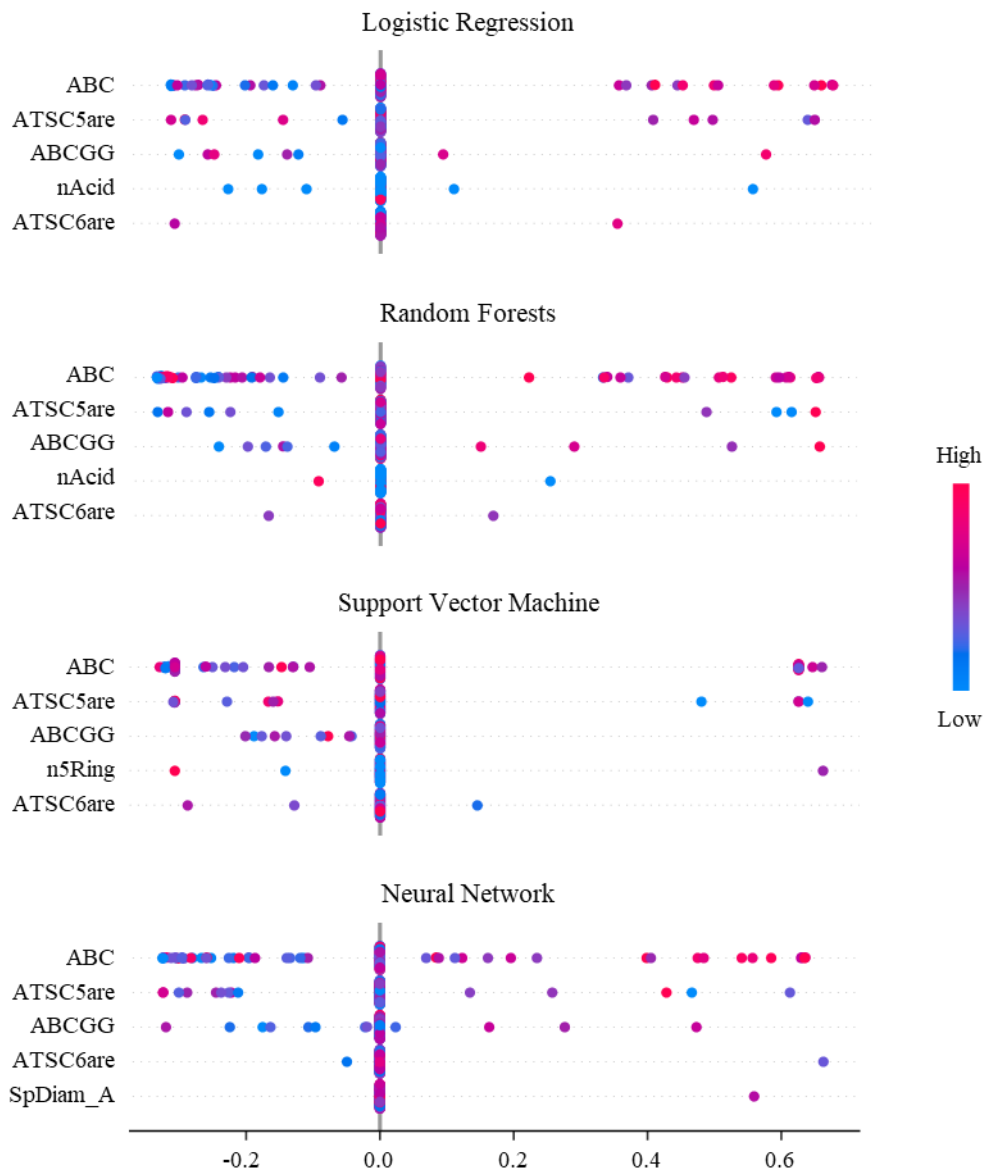
*Figure 9*. Most important features for standard modeling approaches.

## Novel Machine Learning Approaches

We examined several modeling approaches for predicting CYP3A4 inhibitors.

However, all investigated models relied on external featurization of compounds, i.e. with

PaDEL, Mordred, or fingerprint techniques. While these techniques have been widely

used in research, they all depend on tedious manual curation of the features. This may lead to imprecise representations of the molecular structures and lack of relevance to the prediction task at hand. Rather than rely on a human curated preprocessing step of extracting molecular features from a SMILES string, the machine learning algorithm can perform feature extraction instead. This removes the bias and dependence on a particular molecular descriptor tool in feature generation, allowing for the model to learn relevant features for a specific task without human intervention. The features extracted with this machine learning approach are correspondent to detected patterns in the training data, allowing the model to identify sub-structures within a compound that may be important for the prediction task.

Neural network architectures can allow for auto-featurization though connecting multiple nodes in different ways with tunable weights. Convolutional neural networks can extend featurization from a 1-dimensional vector to a 2-dimensional matrix, 3-dimensional matrix, or more [50]. A 1-dimensional convolutional layer can take a 2-dimensional matrix and scan across one dimension with varying kernel sizes, while filtering on the second dimension. This is often beneficial for finding temporal patterns in one-hot encoded strings. A 2-dimensional convolutional layer performs similarly, except it can scan across a 2-dimnesional matrix with an $x$ by $y$ kernel, while filtering on the third dimension. Scanning in multiple dimensions is valuable for detecting spatial patterns.

To investigate these opportunities of auto-feature extraction, we developed two novel approaches in CYP3A4 inhibition prediction based on work published by Hirohara et al. [97]. Specifically, we focused on answering the following two question:

- Can relevant molecular and structural features be extracted and used to accurately predict the potential for CYP3A4 inhibition based on SMILES strings?

- Can relevant molecular and structural features be extracted and used to accurately predict the potential for CYP3A4 inhibition based on a 2-dimensional conformation of the compound's atoms and bonds?

**SMILES string auto-extraction.** A molecule's SMILES string holds information about the structure of the compound, notating atoms that are bonded together, the bond orders, rings, branching structures, and of course the symbol of the atoms present. Treating a SMILES string as a temporal sequence of atoms and bonds can allow for substructures to be identified and featurized from the SMILES string itself. To setup a SMILES string into a numerical matrix that can be fed into a neural network, each character in the string must be one-hot encoded, such that a specific character will receive a specific bit in a vector, much like fingerprints. Through using one-hot encoding on all characters in a SMILES string, a 2-dimensional numeric matrix is created. All the dimension lengths of all SMILES matrices must be the same, thus, the matrices require zero-padding. An example SMILES matrix can be seen in Table 5. In this example, amifostine (NCCCNCCSP(=O)(O)O) was featurized using each character in the

compound's SMILES string. The columns represent each character in the SMILES string, while the rows represent which feature the bit or integer refers too.

Table 5
*SMILE Featurization Matrix for Amifostine*

| Amifostine | N | C | C | C | N | C | C | S | P | ( | = | O | ) | ( | O | ) | O |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| # Hydrogens | 2 | 2 | 2 | 2 | 1 | 2 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| Degree | 3 | 4 | 4 | 4 | 3 | 4 | 4 | 2 | 4 | 0 | 0 | 1 | 0 | 0 | 2 | 0 | 2 |
| Formal Charge | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Valence | 3 | 4 | 4 | 4 | 3 | 4 | 4 | 2 | 5 | 0 | 0 | 2 | 0 | 0 | 2 | 0 | 2 |
| In Ring | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Aromatic | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| C | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| N | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| O | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| Br | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Cl | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| F | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| P | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| S | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ring | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ( | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| ) | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| / | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| \ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| = | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| # | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Counterclockwise Chirality | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Clockwise Chirality | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| SP | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| SP2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| SP3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| Ring Start | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Ring End | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Using 1D convolutional layers, the SMILES matrix can be scanned across the first dimension representing SMILES characters, while pooling and filtering the second dimension representing the one-hot encoded bit vectors. These convolutional layers can be repeated, if beneficial, before inserting the extracted features into a dense layer of nodes. A dense layer is not always needed. Although, one or more dense layers are often added to the end of the model architecture. This architecture is visualized in Figure 10.
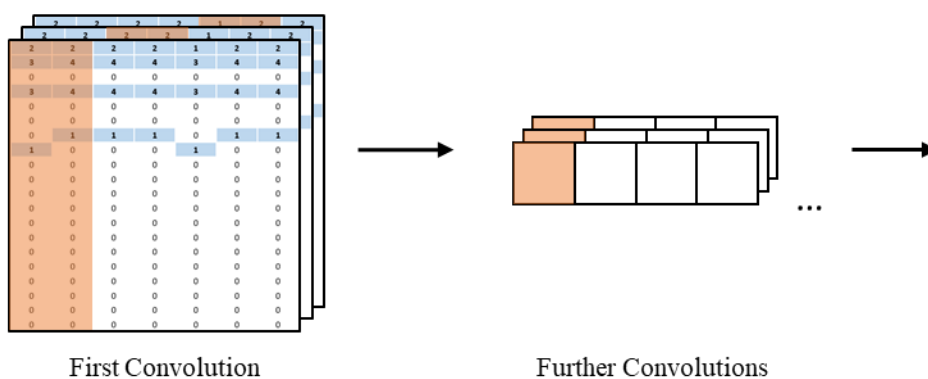


*Figure 10.* First two layers of the SMILE string auto-extraction architecture.

We achieved an accuracy of 0.823 after extensive tuning. The best model consisted of two 1D convolutional layers with 27 and 13 filters, a kernel size of 5 and 4, and a stride of 1 and 4. Dropout proportions of 0.35 and 0.47, respectively, were used to reduce overfitting. Both convolutional layers used a rectified linear unit (ReLU) activation function and were followed by batch normalization. The output layer consisted of one densely connected node using a sigmoid activation function. Hyperparameter details are available in Table 6. Overall, the SMILES string auto-extraction architecture

provided similar accuracy to the standard models, while displaying ability for auto-

featurization from SMILES strings.

Table 6

*Hyperparameter Optimization for Auto-Extractor Models*

| Model | Hyperparameter | Range Considered | Best Value |
|---|---|---|---|
| **SMILE Auto-Extraction** | Conv Layers | [1, 2] | 2 |
| | Conv Layer 1 Filters | [1, 50] | 27 |
| | Conv Layer 1 Kernel | [1, 5] | 5 |
| | Conv Layer 1 Stride | [1, 5] | 1 |
| | Conv Layer 1 Dropout | [0.2, 0.5] | 0.34915 |
| | Conv Layer 2 Filters | [1, 50] | 13 |
| | Conv Layer 2 Kernel | [1, 5] | 4 |
| | Conv Layer 2 Stride | [1, 5] | 4 |
| | Conv Layer 2 Dropout | [0.2, 0.5] | 0.46571 |
| | Dense Layers | [0, 1] | 0 |
| **Structure Auto-Extraction** | Conv Layers | [1, 6] | 4 |
| | Conv Layer 1 Filters | [1, 20] | 8 |
| | Conv Layer 1 Kernel | [1, 25] | 4 |
| | Conv Layer 1 Stride | [1, 5] | 1 |
| | Conv Layer 1 Dropout | [0.2, 0.7] | 0.59298 |
| | Conv Layer 2 Filters | [1, 20] | 3 |
| | Conv Layer 2 Kernel | [1, 25] | 25 |
| | Conv Layer 2 Stride | [1, 5] | 1 |
| | Conv Layer 2 Dropout | [0.2, 0.7] | 0.69967 |
| | Conv Layer 3 Filters | [1, 20] | 15 |
| | Conv Layer 3 Kernel | [1, 25] | 9 |
| | Conv Layer 3 Stride | [1, 5] | 5 |
| | Conv Layer 3 Dropout | [0.2, 0.7] | 0.26906 |
| | Conv Layer 4 Filters | [1, 20] | 4 |
| | Conv Layer 4 Kernel | [1, 25] | 17 |
| | Conv Layer 4 Stride | [1, 5] | 3 |
| | Conv Layer 4 Dropout | [0.2, 0.7] | 0.43041 |
| | Dense Layers | [0, 1] | 1 |
| | Dense Layer 1 Units | [75, 250] | 104 |
| | Dense Layer 1 Dropout | [0, 0,5] | 0.27760 |

**2D structure auto-extraction.** Using the RDKit package, 2D conformations for molecules can be computed from a SMILES string. The generated 2D conformations provide structural information in the form of atom positioning measured in ångströms as well as the bonds connecting atoms. Using the computed x, y coordinates for atoms, a 2D matrix can be created to represent the coordinates at a specified resolution and scale. For example, a 2D matrix consisting of 100x100 pixels can be created and the atom coordinates can be converted into a -15 to 15 ångström scale, so that each atom coordinate gets placed into a pixel within the matrix. The pixels to represent a bond can be calculated using Bresenham's line algorithm, where the bond pixels are determined as the best pixelated line from one atom pixel to the other. Atom's in the matrix can be encoded by their atomic mass, while bonds can be encoded by their bond order. To avoid potential overlapping coordinates of atoms and bonds, two different filters can be constructed for a given 2D matrix, one for atoms and one for bonds. Effectively, each SMILES will result in a [100x100x2] matrix representing the atomic structure. A flattened example of these pixel matrices for Chlorzoxazone and Voriconazole can be seen in Figure 11.
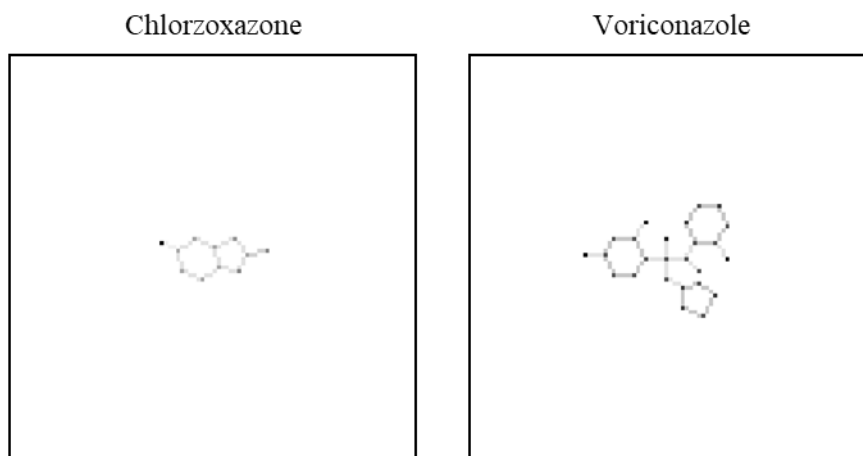
*Figure 11*. 100x100 pixel matrix displaying Chlorzoxazone and Voriconazole.

Using 2D convolutional layers, the structural matrix can be scanned across the x, y dimensions with a rectangular kernel, while pooling and filtering the third dimension of atom and bond filters. These convolutional layers can be repeated if beneficial, before inserting the extracted features into a dense layer of nodes.

The highest accuracy achieved for the best model based on the 2D structure extractor was 0.764, which falls short of the other models, but still shows the promising potential for this approach. The best model for the 2D structure extractor approach consisted of four 2D convolutional layers with 8, 3, 15 and 4 filters, a kernel size of 4, 25, 9, and 17, and a stride of 1, 1, 5, and 3. Following the convolutional layers, 1 dense layer was added with 104 units. Dropout was used on all layers with dropout proportions of 0.59, 0.70, 0.27, and 0.43 for the convolutional layers, respectively, and a dropout proportion of 0.28 for the dense layer. All layers used a rectified linear unit (ReLU) activation function and were followed by batch normalization. The output layer consisted

of one densely connected node using a sigmoid activation function. Hyperparameter details are summarized in Table 6.

Overall, the 2D structure auto-extraction architecture had a lower accuracy than other models. However, it performed reasonably well to be considered for further optimization. The performances of both auto-extractor models are compared in Figure 12. Random forests, the best performing model, was significantly better than the 2D structure auto-extraction approach. Although it was only marginally better than the SMILES string auto-extraction technique. Both novel approaches achieved accuracies greater than 75%.
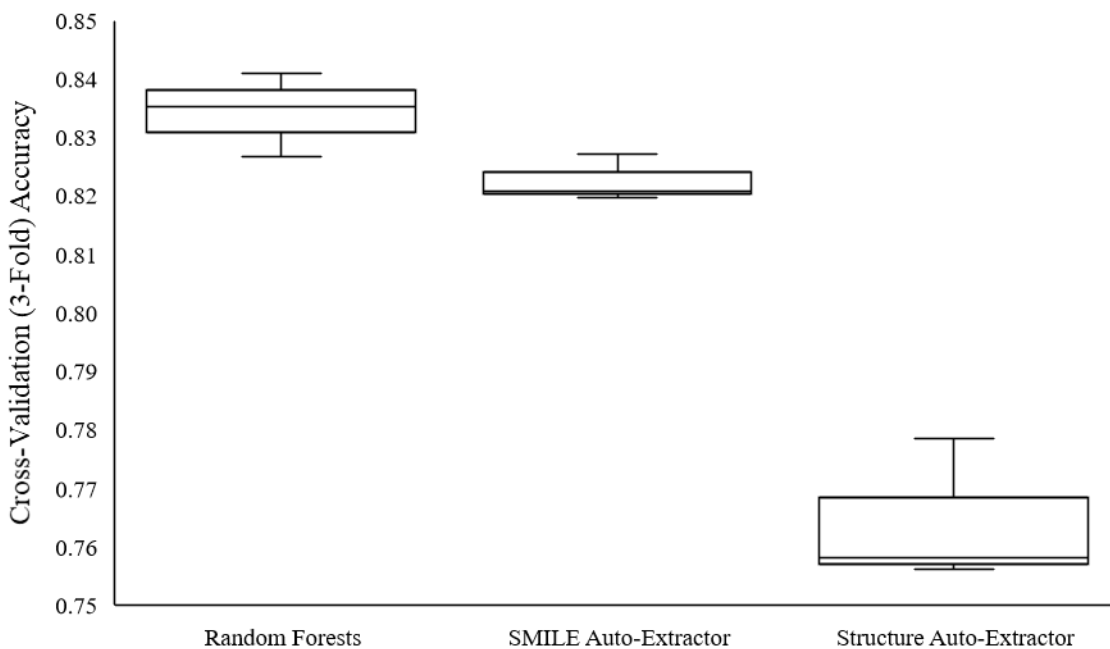


*Figure 12*. Cross-validated (3-fold) accuracies for auto-extraction approaches.

**Validation on Known CYP3A4 Inhibitors**

A set of compounds published by the FDA as either strong, moderate, or weak clinical inhibitors was passed into each model to determine its predictive performance. Inhibition probabilities were calculated for all compounds in each of the tested models. The summary of results for all the models is shown in Figure 13. Logistic regression performed the best with 24 out of 33 samples classified correctly as inhibitors, followed by both convolutional auto-extractor models correctly predicting 16. The Random forests model correctly predicted 13 inhibitors, the neural network model predicted 12, and the support vector machine model predicted only 7. We did not detect a trend in prediction confidence from strong to weak inhibitors for any model. This was anticipated because the strength of inhibition was not considered when we trained the models. Please note, that these predictions are not an indication of model accuracy as only inhibitors were included in this test set. Instead, these results are an indication of the models' recall on inhibitors, or the models' ability to correctly classify a true inhibitor.

| | Structure | SMILE | LR | RF | SVM | NN |
|---|---|---|---|---|---|---|
| **Boceprevir** | 0.76 | 0.82 | 0.33 | 0.40 | 0.06 | 0.38 |
| **Cobicistat** | 0.43 | 0.04 | 1.00 | 0.59 | 0.38 | 0.83 |
| **Ketoconazole** | 0.27 | 0.62 | 0.99 | 0.60 | 0.82 | 0.81 |
| **Posaconazole** | 0.58 | 0.06 | 0.70 | 0.45 | 0.23 | 0.23 |
| **Ritonavir** | 0.22 | 0.00 | 0.97 | 0.53 | 0.25 | 0.52 |
| **Telaprevir** | 0.36 | 0.01 | 0.62 | 0.45 | 0.07 | 0.64 |
| **Telithromycin** | 0.45 | 0.17 | 0.99 | 0.36 | 0.05 | 0.02 |
| **Troleandomycin** | 0.91 | 0.99 | 0.70 | 0.66 | 0.95 | 0.20 |
| **Voriconazole** | 0.13 | 0.41 | 0.67 | 0.40 | 0.06 | 0.11 |
| **Aprepitant** | 0.45 | 0.49 | 0.51 | 0.47 | 0.21 | 0.56 |
| **Ciprofloxacin** | 0.90 | 0.92 | 0.00 | 0.06 | 0.02 | 0.00 |
| **Conivaptan** | 0.68 | 0.42 | 0.95 | 0.63 | 0.04 | 0.74 |
| **Crizotinib** | 0.20 | 0.38 | 0.98 | 0.46 | 0.18 | 0.56 |
| **Cyclosporine** | 0.84 | 0.99 | 0.63 | 0.66 | 0.95 | 0.25 |
| **Diltiazem** | 0.11 | 0.59 | 0.97 | 0.63 | 0.87 | 0.56 |
| **Dronedarone** | 0.44 | 0.06 | 0.64 | 0.48 | 0.26 | 0.20 |
| **Erythromycin** | 0.89 | 0.99 | 0.20 | 0.13 | 0.21 | 0.01 |
| **Fluconazole** | 0.48 | 0.46 | 0.17 | 0.26 | 0.08 | 0.28 |
| **Fluvoxamine** | 0.94 | 0.90 | 0.77 | 0.63 | 0.93 | 0.27 |
| **Imatinib** | 0.35 | 0.28 | 0.58 | 0.47 | 0.02 | 0.18 |
| **Tofisopam** | 0.47 | 0.54 | 0.86 | 0.51 | 0.16 | 0.57 |
| **Verapamil** | 0.39 | 0.11 | 0.83 | 0.54 | 0.88 | 0.61 |
| **Chlorzoxazone** | 0.96 | 0.99 | 0.02 | 0.10 | 0.03 | 0.04 |
| **Cilostazol** | 0.82 | 0.35 | 0.82 | 0.53 | 0.37 | 0.26 |
| **Cimetidine** | 0.94 | 0.37 | 0.38 | 0.23 | 0.07 | 0.05 |
| **Clotrimazole** | 0.33 | 0.22 | 0.99 | 0.68 | 0.82 | 0.85 |
| **Fosaprepitant** | 0.24 | 0.60 | 0.16 | 0.43 | 0.17 | 0.44 |
| **Istradefylline** | 0.69 | 0.58 | 0.89 | 0.45 | 0.24 | 0.29 |
| **Ivacaftor** | 0.61 | 0.73 | 0.20 | 0.27 | 0.04 | 0.07 |
| **Lomitapide** | 0.39 | 0.14 | 0.93 | 0.57 | 0.27 | 0.80 |
| **Ranitidine** | 0.91 | 0.91 | 0.02 | 0.01 | 0.02 | 0.04 |
| **Ranolazine** | 0.71 | 0.87 | 0.61 | 0.34 | 0.09 | 0.18 |
| **Ticagrelor** | 0.65 | 0.93 | 0.70 | 0.36 | 0.07 | 0.03 |

*(Row groups labeled along the left margin: Strong (Boceprevir–Voriconazole), Moderate (Aprepitant–Verapamil), Weak (Chlorzoxazone–Ticagrelor).)*
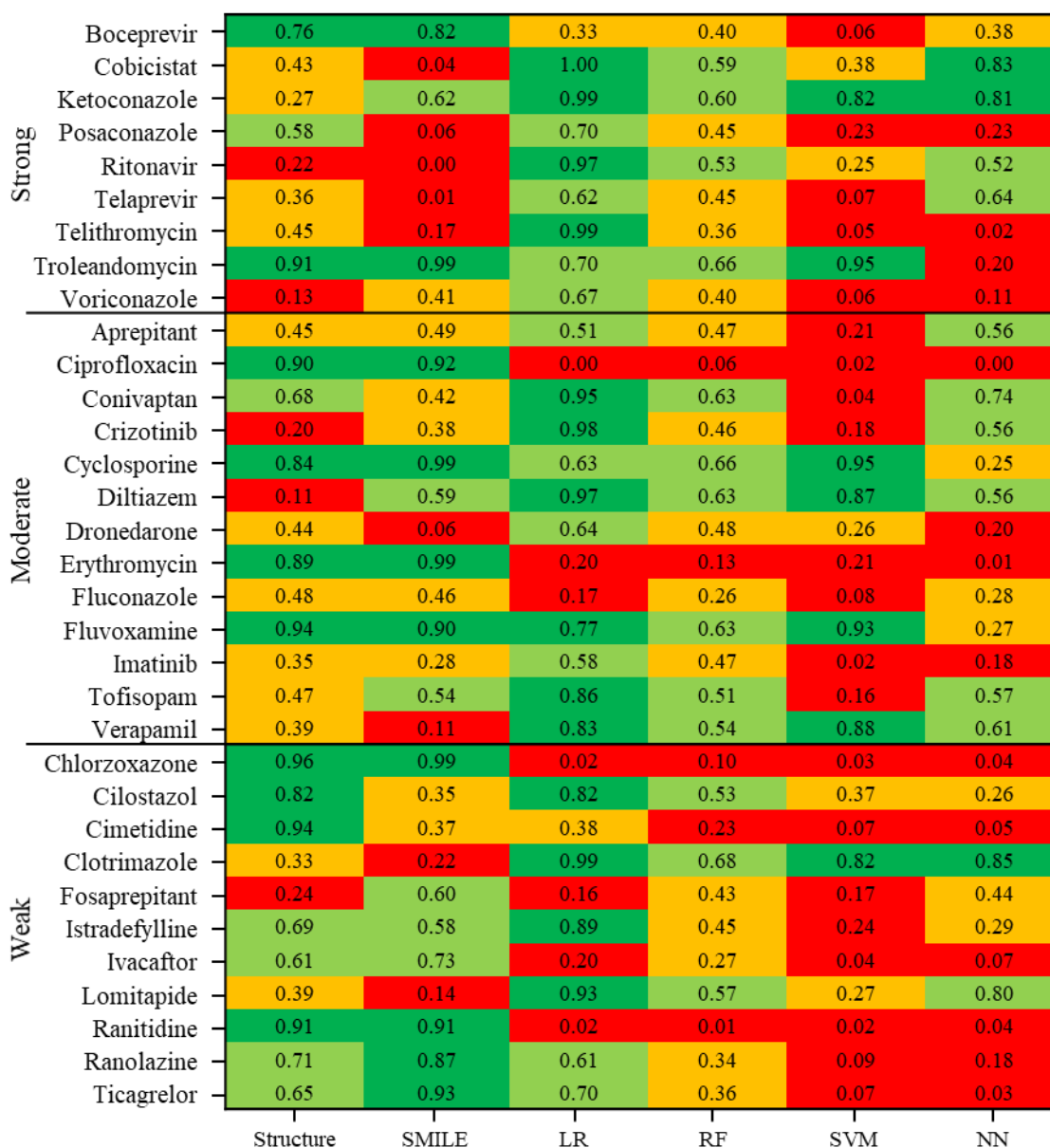
*Figure 13*. Inhibition prediction probabilities on FDA inhibitors for all models.

For each model, we created a SHAP force plot showing the best and the worst predictions as well as providing an insight as to which variables were most influential.

65

Figure 14 illustrates the generated SHAP force plots. For logistic regression, *cobicistat*[2] was correctly predicted as an inhibitor with the most confidence. The two features that were deemed most important to this decision were centered Moreau-Broto autocorrelation of lag 8 weighted by Van der Waals volume (ATSC8v) and Moreau-Broto autocorrelation of lag 8 weighted by sigma electrons (ATS8d) [95]. The inhibition prediction with the lowest confidence was for the compound *chlorzoxazone*[3]. The Geary coefficient of lag 4 weighted by ionization potential (GATS4i) [98] weighted the classification towards an inhibitor. However, the average Moreau-Broto autocorrelation of lag 0 weighted by Pauling electronegativity (AATS0pe), and the presence of PubChem fingerprint 669 (Cl-C:C-C=O) [86] confused the model and caused an incorrect prediction of non-inhibitor.

---

[2] Drug molecule for use in the treatment of human immunodeficiency virus infection (HIV/AIDS)
[3] Drug molecule for use as muscle relaxant to treat muscle spasm and the resulting pain or discomfort.
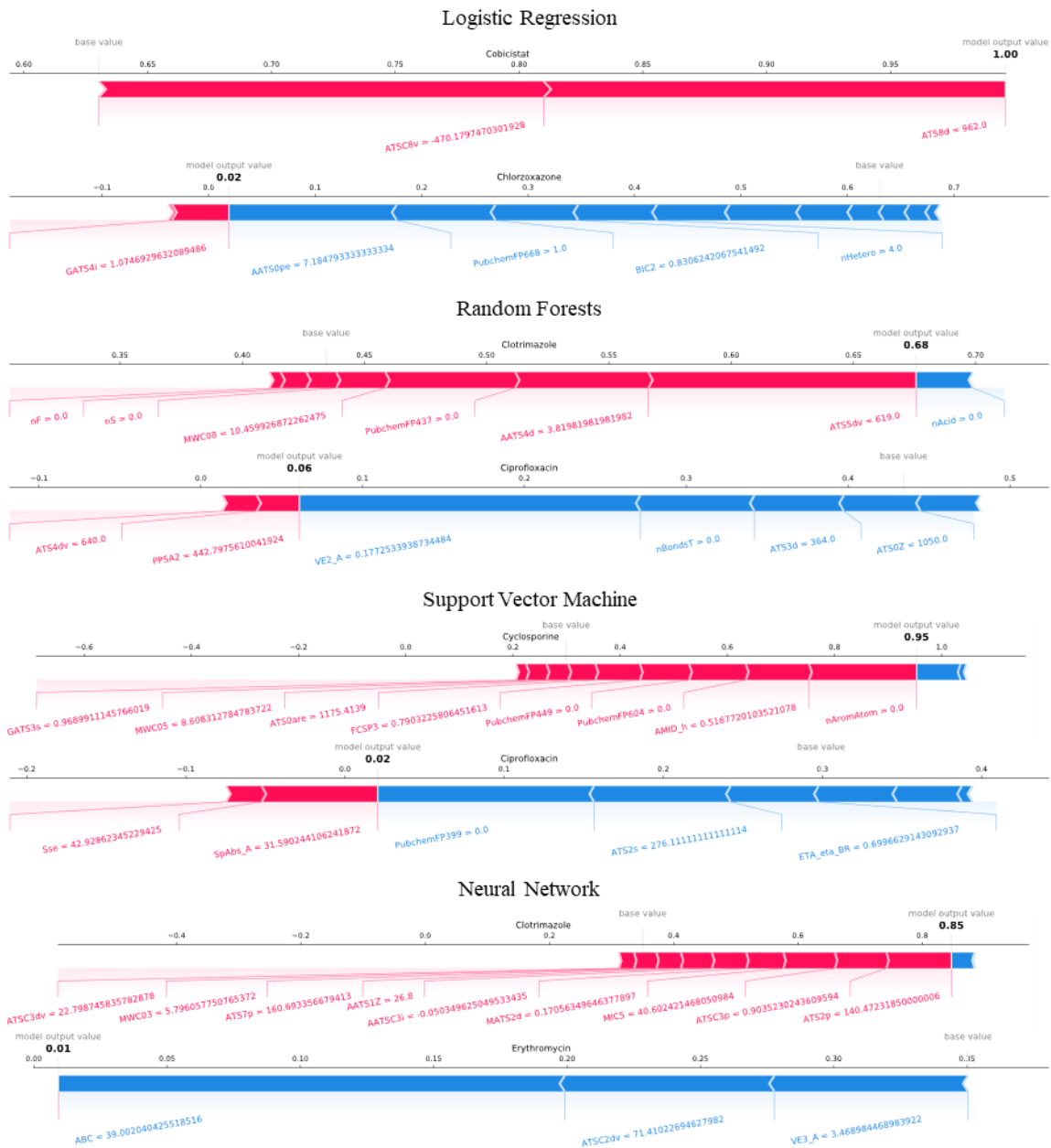
*Figure 14*. SHAP force plots for the best and worst FDA test predictions.

Specific to the 2D structure auto-extractor approach, we can explore the spatial areas and potentially the sub-structures within a compound that are most influential to the model using SHAP values. An example of this exploration is shown in Figure 15 with

*chlorzoxazone* and *voriconazole*[4]. Pixels that are shaded red contribute more towards a classification of inhibitor, while pixels shaded blue contribute more towards the non-inhibitor classification. The 2D structure model incorrectly classificed voriconazole as a non-inhibitor, likely because of the 3 ring structures present around the center of the molecule as indicated by the blue shading.



*Figure 15*. 2D SHAP values for predictions on Chlorzoxazone and Voriconazole.

---

[4] Drug molecule known as <u>antifungal medication</u>, used to treat a number of <u>fungal infections</u>.

## Chapter 6

## Conclusions and Future Opportunities

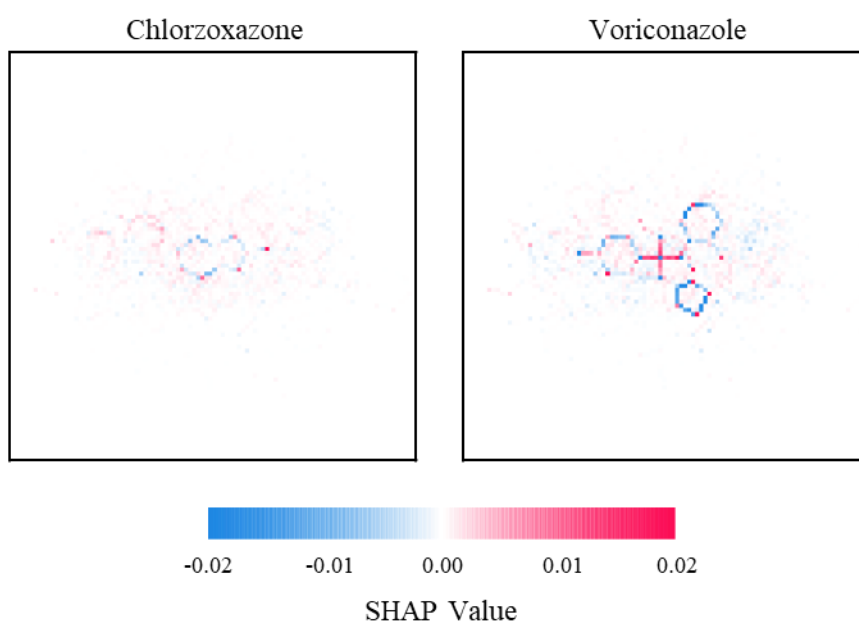Four machine learning methods, logistic regression, random forests, support vector machine, and artificial neural network were used to create classification models to link compound structures with their ability to inhibit CYP3A4 activity. All models shared the same training set derived from the PubChem AID: 1851 dataset. Two new approaches for extracting molecular features from compound structures were developed and investigated: 1) external featurization using Mordred, and 2) feature auto-extraction from the SMILES representation of the compounds' structure using CNNs. The results of the featurization approaches showed that the random forests models based on external featurization performed significantly better than the 2D structure auto-extraction approach. However, its performance was only marginally better than the SMILES string auto-extraction technique which requires farther study and optimization.

Two features: 1) the atom-bond connectivity index (ABC) and 2) the centered Moreau-Broto autocorrelation of lag 5 weighted by Allred-Rochow electronegativity (ATSC5are) were assigned the highest importance in all models based on external featurization. Additionally, all models based on external featurization share Graovac-Ghorbani atom-bond connectivity index (ABCGG) and centered Moreau-Broto autocorrelation of lag 6 weighted by Allred-Rochow electronegativity (ATSC6are) as part of their top five features.

All studied models performed well (greater than 75% accuracy) at classifying compounds as either inhibitors or non-inhibitors. Random forests and logistic regression

performed the best with accuracies of 0.834 and 0.832, respectively. Although, Random forests had a larger variance between the three cross validated folds. Support vector machine had the worst performance of the four approaches and a large variance between fold accuracies. The neural network approach had the most consistent accuracy measures between the three folds.

All developed models were validated on the set of compounds published by the FDA. Logistic regression performed the best with 24 out of 33 compounds correctly classified as inhibitors, followed by both convolutional auto-extractor models correctly predicting 16, the random forests model predicting 13, the neural network model predicting 12, and the support vector machine model predicting only 7.

The modeling methods implemented in this study are not limited to predicting only CYP3A4 inhibitors. Rather, they can learn to predict any activity for a given target. Possible model expansion can be achieved through adding more refined data labels and changing classification output from a binary case to a spectrum consisting of multiple labels, e.g. strong, medium, or weak inhibitors labels. These added classes could improve predictive power as compound activity may not be binary. Studied models can be expanded into predicting compound's activity on a continuous scale yielding a numerical measure of compound's potency towards a target. In addition to possible model improvements from refined labels, the number of compounds in the training dataset should be increased as well. It is important to ensure that new data is readily available for improved models as new compounds are tested *in vitro* and/or *in vivo*. Models could

continuously learn and improve their predictive power over time by connecting to a downstream data pipeline.

Research into model exploration tools could find important substructures within a compound. If the substructure is important for the binding task, it should be present more often in samples with high binding activities. Models such as the SMILES string auto-extractor or the 2D structure auto-extractor can assign high weights to nodes relating to these substructures. This allows for diagnostic tools to explore models and discover highly weighted substructures within the compounds.

Predicting a compound's potential to inhibit an enzyme such as CYP3A4 is important. However, adjusting the compound to remove inhibitory activity is another complicated but crucial step. A tool that generates potential compound derivates from a parent compound could be incorporated in predictive models. After generating all derivatives, each compound could have, for example, their CYP3A4 inhibition potential predicted. From these predictions, the tool can score each derivative and display them to a chemist for further considerations. A tool like this could assist with structural modification on drug candidates, allowing for chemists to find paths that remove or greatly diminish the inhibitory affects.

# References

[1]     J. Drews, "Drug discovery: a historical perspective.," *Science*, vol. 287, no. 5460, pp. 1960–4, Mar. 2000.

[2]     Y. C. Lo, S. E. Rensi, W. Torng, and R. B. Altman, "Machine learning in chemoinformatics and drug discovery," *Drug Discov. Today*, vol. 23, no. 8, pp. 1538–1546, 2018.

[3]     W. Sneader, *Drug Discovery: A History*, 1st ed. Hoboken, NJ, USA: John Wiley and Sons, 2005.

[4]     M. A. Ator, J. P. Mallamo, and M. Williams, "Overview of drug discovery and development.," *Curr. Protoc. Pharmacol.*, vol. Chapter 9, p. Unit9.9, Dec. 2006.

[5]     PhRMA, "Modernizing Drug Discovery, Development & Approval," 2016.

[6]     T. J. Moore, H. Zhang, G. Anderson, and G. C. Alexander, "Estimated Costs of Pivotal Trials for Novel Therapeutic Agents Approved by the US Food and Drug Administration, 2015-2016," *JAMA Intern. Med.*, vol. 178, no. 11, p. 1451, Nov. 2018.

[7]     C. H. Wong, K. W. Siah, and A. W. Lo, "Estimation of clinical trial success rates and related parameters," *Biostatistics*, vol. 20, no. 2, pp. 273–286, Apr. 2019.

[8]     J. A. DiMasi, H. G. Grabowski, and R. W. Hansen, "Innovation in the pharmaceutical industry: New estimates of R&D costs.," *J. Health Econ.*, vol. 47, pp. 20–33, May 2016.

[9]     H. van de Waterbeemd and E. Gifford, "ADMET in silico modelling: Towards prediction paradise?," *Nat. Rev. Drug Discov.*, vol. 2, no. 3, pp. 192–204, 2003.

[10]    D. Schuster, C. Laggner, and T. Langer, "Why Drugs Fail - A Study on Side Effects in New Chemical Entities," *Curr. Pharm. Des.*, vol. 11, no. 27, pp. 3545–3559, Oct. 2005.

[11]    D. A. Parasrampuria, L. Z. Benet, and A. Sharma, "Why Drugs Fail in Late Stages of Development: Case Study Analyses from the Last Decade and Recommendations.," *AAPS J.*, vol. 20, no. 3, p. 46, 2018.

[12]    S. Ekins, B. Boulanger, P. W. Swaan, and M. A. Z. Hupcey, "Towards a new age of virtual ADME/TOX and multidimensional drug discovery.," *Mol. Divers.*, vol. 5, no. 4, pp. 255–75, 2002.

[13] M. D. Segall, "Multi-Parameter Optimization: Identifying High Quality Compounds with a Balance of Properties," *Curr. Drug Metab.*, vol. 18, no. 9, pp. 1292–1310, Mar. 2012.

[14] C. A. Lipinski, F. Lombardo, B. W. Dominy, and P. J. Feeney, "Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings.," *Adv. Drug Deliv. Rev.*, vol. 46, no. 1–3, pp. 3–26, Mar. 2001.

[15] P. D. Leeson and B. Springthorpe, "The influence of drug-like concepts on decision-making in medicinal chemistry," *Nat. Rev. Drug Discov.*, vol. 6, no. 11, pp. 881–890, 2007.

[16] D. F. Veber, S. R. Johnson, H.-Y. Cheng, B. R. Smith, K. W. Ward, and K. D. Kopple, "Molecular properties that influence the oral bioavailability of drug candidates.," *J. Med. Chem.*, vol. 45, no. 12, pp. 2615–23, Jun. 2002.

[17] I. Yusof and M. D. Segall, "Considering the impact drug-like properties have on the chance of success," *Drug Discov. Today*, vol. 18, no. 13–14, pp. 659–666, 2013.

[18] G. R. Bickerton, G. V Paolini, J. Besnard, S. Muresan, and A. L. Hopkins, "Quantifying the chemical beauty of drugs," *Nat. Chem.*, vol. 4, no. 2, pp. 90–98, Feb. 2012.

[19] I. Yusof, F. Shah, T. Hashimoto, M. D. Segall, and N. Greene, "Finding the rules for successful drug optimisation," *Drug Discov. Today*, vol. 19, no. 5, pp. 680–687, 2014.

[20] J. Panteleev, H. Gao, and L. Jia, "Recent applications of machine learning in medicinal chemistry," *Bioorganic Med. Chem. Lett.*, vol. 28, no. 17, pp. 2807–2815, 2018.

[21] W. L. Jorgensen and E. M. Duffy, "Prediction of drug solubility from structure," *Adv. Drug Deliv. Rev.*, vol. 54, no. 3, pp. 355–366, Mar. 2002.

[22] F. Lombardo and Y. Jing, "In Silico Prediction of Volume of Distribution in Humans. Extensive Data Set and the Exploration of Linear and Nonlinear Methods Coupled with Molecular Interaction Fields Descriptors.," *J. Chem. Inf. Model.*, vol. 56, no. 10, pp. 2042–2052, 2016.

[23] A. Lavecchia, "Machine-learning approaches in drug discovery: Methods and applications," *Drug Discov. Today*, vol. 20, no. 3, pp. 318–331, 2015.

[24]    L. Zhang, J. Tan, D. Han, and H. Zhu, "From machine learning to deep learning: progress in machine intelligence for rational drug discovery," *Drug Discov. Today*, vol. 22, no. 11, pp. 1680–1685, 2017.

[25]    T. Bayes and Price, "LII. An essay towards solving a problem in the doctrine of chances. By the late Rev. Mr. Bayes, F. R. S. communicated by Mr. Price, in a letter to John Canton, A. M. F. R. S," *Philos. Trans. R. Soc. London*, vol. 53, pp. 370–418, Jan. 1763.

[26]    R. Caruana and A. Niculescu-Mizil, "An Empirical Comparison of Supervised Learning Algorithms," in *Proceedings of the 23rd International Conference on Machine Learning*, 2006, pp. 161–168.

[27]    Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin, *Learning from Data*. AMLBook, 2012.

[28]    F. Galton, "Regression Towards Mediocrity in Hereditary Stature.," *J. Anthropol. Inst. Gt. Britain Irel.*, vol. 15, pp. 246–263, 1886.

[29]    X. Yan and X. G. Su, *Linear Regression Analysis*. Singapore, UNITED STATES: WORLD SCIENTIFIC, 2009.

[30]    T. Zhang, "Solving large scale linear prediction problems using stochastic gradient descent algorithms," in *Twenty-first international conference on Machine learning - ICML '04*, 2004, p. 116.

[31]    J. S. Cramer, "The Origins of Logistic Regression," *SSRN Electron. J.*, Dec. 2003.

[32]    S. Dreiseitl and L. Ohno-Machado, "Logistic regression and artificial neural network classification models: a methodology review," *J. Biomed. Inform.*, vol. 35, no. 5–6, pp. 352–359, Oct. 2002.

[33]    L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization Methods for Large-Scale Machine Learning," Jun. 2016.

[34]    P. Baumann, D. S. Hochbaum, and Y. T. Yang, "A comparative study of the leading machine learning techniques and two new optimization algorithms," *Eur. J. Oper. Res.*, vol. 272, no. 3, pp. 1041–1057, Feb. 2019.

[35]    R. Tibshirani, "Regression Shrinkage and Selection Via the Lasso," *J. R. Stat. Soc. Ser. B*, 1996.

[36]    L. Breiman, J. H. Friedman, C. J. Stone, and R. A. Olshen, *Classification And Regression Trees*, 1st ed. Chapman and Hall/CRC, 1984.

[37] C. E. Shannon, "A Mathematical Theory of Communication," *Bell Syst. Tech. J.*, vol. 27, no. 3, pp. 379–423, Jul. 1948.

[38] J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, Mar. 1986.

[39] Tin Kam Ho, "Random decision forests," in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, 1995, vol. 1, pp. 278–282.

[40] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, Aug. 1996.

[41] Y. Freund and R. E. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, Aug. 1997.

[42] L. Breiman, "Random Forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, Oct. 2001.

[43] R. Caruana, N. Karampatziakis, and A. Yessenalina, "An empirical evaluation of supervised learning in high dimensions," in *Proceedings of the 25th international conference on Machine learning - ICML '08*, 2008, pp. 96–103.

[44] M. Fernández-Delgado, E. Cernadas, S. Barro, D. Amorim, and A. Fernández-Delgado, "Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?," 2014.

[45] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, Sep. 1995.

[46] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychol. Rev.*, vol. 65, no. 6, pp. 386–408, Nov. 1958.

[47] M. Minsky and S. Papert, *Perceptrons: an introduction to computational geometry*. MIT Press, 1969.

[48] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, 1986.

[49] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, 2016.

[50] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, "Object Recognition with Gradient-Based Learning," 1999, pp. 319–345.

[51] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, 1997.

[52]    N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *J. Artif. Intell. Res.*, vol. 16, no. 1, pp. 321–357, Jun. 2002.

[53]    A. S. Hess and J. R. Hess, "Principal component analysis," *Transfusion*, vol. 58, no. 7, pp. 1580–1582, Jul. 2018.

[54]    S. Lundberg and S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," *Adv. Neural Inf. Process. Syst.*, vol. 2017-Decem, no. Section 2, pp. 4766–4775, May 2017.

[55]    H. Raunio, M. Kuusisto, R. O. Juvonen, and O. T. Pentikäinen, "Modeling of interactions between xenobiotics and cytochrome P450 (CYP) enzymes," *Front. Pharmacol.*, vol. 6, no. MAY, pp. 1–14, Jun. 2015.

[56]    C. A. MacMunn, "VI. Researches on myohamatin and the histohæmatins," *Philos. Trans. R. Soc. London*, vol. 177, pp. 267–298, Jan. 1886.

[57]    D. Keilin, "On Cytochrome, a Respiratory Pigment, Common to Animals, Yeast, and Higher Plants," *Proc. R. Soc. B Biol. Sci.*, vol. 98, no. 690, pp. 312–339, Aug. 1925.

[58]    T. YAMANAKA and K. OKUNUKI, "CYTOCHROMES," in *Microbial Iron Metabolism*, vol. 10, no. 15, Elsevier, 1974, pp. 349–400.

[59]    F. J. Gonzalez and H. V Gelboin, "Human cytochromes P450: evolution and cDNA-directed expression.," *Environ. Health Perspect.*, vol. 98, pp. 81–5, Nov. 1992.

[60]    B. Meunier, S. P. de Visser, and S. Shaik, "Mechanism of Oxidation Reactions Catalyzed by Cytochrome P450 Enzymes," *Chem. Rev.*, vol. 104, no. 9, pp. 3947–3980, Sep. 2004.

[61]    F. P. Guengerich, "Cytochrome P450 and Chemical Toxicology," *Chem. Res. Toxicol.*, vol. 21, no. 1, pp. 70–83, Jan. 2008.

[62]    H. Hashimoto *et al.*, "Gene structure of CYP3A4, an adult-specific form of cytochrome P450 in human livers, and its transcriptional control.," *Eur. J. Biochem.*, vol. 218, no. 2, pp. 585–95, Dec. 1993.

[63]    K. Inoue *et al.*, "Assignment of the human cytochrome P-450 nifedipine oxidase gene (CYP3A4) to chromosome 7 at band q22.1 by fluorescencein situ hybridization," *Jpn. J. Hum. Genet.*, vol. 37, no. 2, pp. 133–138, Jun. 1992.

[64]  L. C. Wienkers and T. G. Heath, "Predicting in vivo drug interactions from in vitro drug discovery data," *Nat. Rev. Drug Discov.*, vol. 4, no. 10, pp. 825–833, Oct. 2005.

[65]  U. M. Zanger and M. Schwab, "Cytochrome P450 enzymes in drug metabolism: Regulation of gene expression, enzyme activities, and impact of genetic variation," *Pharmacol. Ther.*, vol. 138, no. 1, pp. 103–141, Apr. 2013.

[66]  R. G. Tirona and R. B. Kim, "Introduction to Clinical Pharmacology," in *Clinical and Translational Science*, Elsevier, 2017, pp. 365–388.

[67]  K. He, K. R. Iyer, R. N. Hayes, M. W. Sinz, T. F. Woolf, and P. F. Hollenberg, "Inactivation of Cytochrome P450 3A4 by Bergamottin, a Component of Grapefruit Juice," *Chem. Res. Toxicol.*, vol. 11, no. 4, pp. 252–259, Apr. 1998.

[68]  K. E. Thummel *et al.*, "Use of midazolam as a human cytochrome P450 3A probe: II. Characterization of inter- and intraindividual hepatic CYP3A variability after liver transplantation.," *J. Pharmacol. Exp. Ther.*, vol. 271, no. 1, pp. 557–66, Oct. 1994.

[69]  H. Yamazaki, T. Niwa, N. Murayama, and C. Emoto, "Comparison of Kinetic Parameters for Drug Oxidation Rates and Substrate Inhibition Potential Mediated by Cytochrome P450 3A4 and 3A5," *Curr. Drug Metab.*, vol. 9, no. 1, pp. 20–33, Jan. 2008.

[70]  W. C. Wright, J. Chenge, and T. Chen, "Structural perspectives of the CYP3A family and their small molecule modulators in drug metabolism," *Liver Res.*, vol. 3, no. 3–4, pp. 132–142, Dec. 2019.

[71]  E. Laille *et al.*, "Evaluation of CYP3A-mediated drug-drug interactions with romidepsin in patients with advanced cancer," *J. Clin. Pharmacol.*, vol. 55, no. 12, pp. 1378–1385, Dec. 2015.

[72]  S. A. Peters, C. R. Jones, A.-L. Ungell, and O. J. D. Hatley, "Predicting Drug Extraction in the Human Gut Wall: Assessing Contributions from Drug Metabolizing Enzymes and Transporter Proteins using Preclinical Models," *Clin. Pharmacokinet.*, vol. 55, no. 6, pp. 673–696, Jun. 2016.

[73]  S. Pandey, P. Pandey, G. Tiwari, and R. Tiwari, "Bioanalysis in drug discovery and development," *Pharm. Methods*, vol. 1, no. 1, p. 14, 2010.

[74]  J. H. Lin and A. Y. H. Lu, "Role of pharmacokinetics and metabolism in drug discovery and development.," *Pharmacol. Rev.*, vol. 49, no. 4, pp. 403–49, Dec. 1997.

[75]  G. Li, K. Huang, D. Nikolic, and R. B. van Breemen, "High-Throughput
      Cytochrome P450 Cocktail Inhibition Assay for Assessing Drug-Drug and Drug-
      Botanical Interactions," *Drug Metab. Dispos.*, vol. 43, no. 11, pp. 1670–1678,
      Nov. 2015.

[76]  F. Cheng *et al.*, "Classification of Cytochrome P450 Inhibitors and Noninhibitors
      Using Combined Classifiers," *J. Chem. Inf. Model.*, vol. 51, no. 5, pp. 996–1011,
      May 2011.

[77]  H. Sun, H. Veith, M. Xia, C. P. Austin, and R. Huang, "Predictive Models for
      Cytochrome P450 Isozymes Based on Quantitative High Throughput Screening
      Data," *J. Chem. Inf. Model.*, vol. 51, no. 10, pp. 2474–2481, Oct. 2011.

[78]  X. Li, Y. Xu, L. Lai, and J. Pei, "Prediction of Human Cytochrome P450
      Inhibition Using a Multitask Deep Autoencoder Neural Network," *Mol. Pharm.*,
      vol. 15, no. 10, pp. 4336–4345, Oct. 2018.

[79]  Z. Wu, T. Lei, C. Shen, Z. Wang, D. Cao, and T. Hou, "ADMET Evaluation in
      Drug Discovery. 19. Reliable Prediction of Human Cytochrome P450 Inhibition
      Using Artificial Intelligence Approaches," *J. Chem. Inf. Model.*, vol. 59, no. 11,
      pp. 4587–4601, Nov. 2019.

[80]  A. T. Balaban, "Applications of graph theory in chemistry," *J. Chem. Inf. Model.*,
      vol. 25, no. 3, pp. 334–343, Aug. 1985.

[81]  D. Weininger, "SMILES, a chemical language and information system. 1.
      Introduction to methodology and encoding rules," *J. Chem. Inf. Model.*, vol. 28,
      no. 1, pp. 31–36, Feb. 1988.

[82]  D. Weininger, A. Weininger, and J. L. Weininger, "SMILES. 2. Algorithm for
      generation of unique SMILES notation," *J. Chem. Inf. Model.*, vol. 29, no. 2, pp.
      97–101, May 1989.

[83]  G. Papadatos and J. P. Overington, "The ChEMBL database: a taster for medicinal
      chemists," *Future Med. Chem.*, vol. 6, no. 4, pp. 361–364, Mar. 2014.

[84]  A. Cereto-Massagué, M. J. Ojeda, C. Valls, M. Mulero, S. Garcia-Vallvé, and G.
      Pujadas, "Molecular fingerprint similarity search in virtual screening," *Methods*,
      vol. 71, no. C, pp. 58–63, Jan. 2015.

[85]  J. L. Durant, B. A. Leland, D. R. Henry, and J. G. Nourse, "Reoptimization of
      MDL Keys for Use in Drug Discovery," *J. Chem. Inf. Comput. Sci.*, vol. 42, no. 6,
      pp. 1273–1280, Nov. 2002.

[86]    E. E. Bolton, Y. Wang, P. A. Thiessen, and S. H. Bryant, "PubChem: Integrated Platform of Small Molecules and Biological Activities," in *Annual Reports in Computational Chemistry*, vol. 4, Elsevier BV, 2008, pp. 217–241.

[87]    D. Rogers and M. Hahn, "Extended-Connectivity Fingerprints," *J. Chem. Inf. Model.*, vol. 50, no. 5, pp. 742–754, May 2010.

[88]    C. W. Yap, "PaDEL-descriptor: An open source software to calculate molecular descriptors and fingerprints," *J. Comput. Chem.*, vol. 32, no. 7, pp. 1466–1474, May 2011.

[89]    H. Moriwaki, Y.-S. Tian, N. Kawashita, and T. Takagi, "Mordred: a molecular descriptor calculator," *J. Cheminform.*, vol. 10, no. 1, p. 4, Dec. 2018.

[90]    A. V Joshi, *Machine Learning and Artificial Intelligence*. Cham: Springer International Publishing, 2020.

[91]    H. Veith *et al.*, "Comprehensive characterization of cytochrome P450 isozyme selectivity across chemical libraries," *Nat. Biotechnol.*, vol. 27, no. 11, pp. 1050–1055, Nov. 2009.

[92]    F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[93]    T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A Next-generation Hyperparameter Optimization Framework," Jul. 2019.

[94]    K. C. Das, "Atom-bond connectivity index of graphs," *Discret. Appl. Math.*, vol. 158, no. 11, pp. 1181–1188, Jun. 2010.

[95]    G. Moreau and P. Broto, "Autocorrelation of a topological structure: A new molecular descriptor," *Nouv. J. Chim*, vol. 4, no. 6, pp. 359–360, 1980.

[96]    A. Graovac and M. Ghorbani, "A new version of atom-bond connectivity index.," *Acta Chim. Slov.*, vol. 57, no. 3, pp. 609–12, Sep. 2010.

[97]    M. Hirohara, Y. Saito, Y. Koda, K. Sato, and Y. Sakakibara, "Convolutional neural network based on SMILES representation of compounds for detecting chemical motif," *BMC Bioinformatics*, vol. 19, no. S19, p. 526, Dec. 2018.

[98]    Q. Luo, D. A. Griffith, and H. Wu, "The Moran Coefficient and the Geary Ratio: Some Mathematical and Numerical Comparisons," 2017, pp. 253–269.

## featurized.py

```python
import os
import re
import numpy as np
import pandas as pd
from mordred import Calculator, descriptors
from padelpy import padeldescriptor
from rdkit import Chem
from rdkit.Chem import AllChem
from sklearn.preprocessing import OneHotEncoder
from tqdm import notebook


def generate_descriptors(smiles):
    """
    Generate molecular descriptors using the Mordred package for a list of SMILE strings.
    :param smiles: An iterable collection of SMILE strings.
    :return: A pandas data frame of generated mordred descriptors by SMILE string.
    """
    descriptors_table = np.ndarray((len(smiles), 1826), dtype=object)
    print("Generating mordred descriptors:")
    for index in notebook.tqdm(range(descriptors_table.shape[0])):
        structure = smiles[index]
        mol = Chem.MolFromSmiles(structure)
        if mol is None:
            descriptors_table[index, :] = [None] * 1826
        else:
            AllChem.EmbedMolecule(mol, useExpTorsionAnglePrefs=True, useBasicKnowledge=True)
            descriptors_table[index, :] = Calculator(descriptors, ignore_3D=False)(mol).fill_missing()
    return pd.DataFrame(descriptors_table, columns=Calculator(descriptors, ignore_3D=False).descriptors)


def generate_fingerprints(smiles: pd.Series) -> pd.DataFrame:
    """
    Generate PubChem fingerprints for a list of SMILE strings using PaDEL.
    :param smiles: A pandas series of SMILE strings.
    :return: A pandas data frame of PubChem fingerprint bits by SMILE string.
    """
    print("Generating fingerprints:")
    smiles.to_csv("temp_smiles.smi", index=False, header=False)
    padeldescriptor(mol_dir="temp_smiles.smi",d_file="fingerprints.csv",fingerprints=True,retainorder=True)
    fingerprints_table = pd.read_csv("fingerprints.csv").drop("Name", axis="columns")
    os.remove("temp_smiles.smi")
    os.remove("fingerprints.csv")
    print("\tDone.\n")
    return fingerprints_table


def extract_smiles(smiles, max_length=250) -> np.ndarray:
    """
    Extract a stack of one-hot encoded 2D matrices from a list of SMILE strings.
    :param smiles: An iterable collection of SMILE strings.
    :param max_length: The length of the SMILE string dimension, those shorter than this are 0-padded to this length.
    :return: A 3-dimensional ndarray of (samples, smile positions, one-hot encoded features).
    """
```

```python
    smile_list = []
    print("Extracting SMILE matrices:")
    for smile in notebook.tqdm(smiles):
        smile_list.append(__extract_smile_features(smile, max_length))

    return np.stack(smile_list)

def extract_smile_structures(smiles, resolution=100, scale=(-15, 15)) -> np.ndarray:
    """
    Extract a stack of 2-dimensional matricies from a list of SMILE strings.
    :param smiles: An iterable collection of SMILE strings.
    :return: A 4-dimensional ndarray of (samples, x-coordinates, y-coordinates, filters).
    """
    mol_list = []
    print("Extracting 2D structures:")
    for smile in notebook.tqdm(smiles):
        matrix = None
        mol = Chem.MolFromSmiles(smile)
        if mol is not None:
            mol.Compute2DCoords()
            matrix = __extract_mol_structure(mol, 0, resolution, scale)
        if matrix is None:
            matrix = np.full((resolution, resolution, 2), -1, dtype='b')
        mol_list.append(matrix)

    return np.stack(mol_list)

def __extract_atom_features(molecule, atom_index):
    symbol_features = []
    atom = molecule.GetAtomWithIdx(atom_index)
    symbol_features.append(atom.GetSymbol())
    symbol_features.append(atom.GetTotalNumHs())
    symbol_features.append(atom.GetTotalDegree())
    symbol_features.append(atom.GetFormalCharge())
    symbol_features.append(atom.GetTotalValence())
    symbol_features.append(atom.IsInRing() * 1)
    symbol_features.append(atom.GetIsAromatic() * 1)
    symbol_features.append(str(atom.GetChiralTag()))
    symbol_features.append(str(atom.GetHybridization()))
    symbol_features.append(0)
    two_char_abbr_flag = True if len(atom.GetSymbol()) > 1 else False
    return symbol_features, two_char_abbr_flag

def __extract_smile_features(smile, max_length):
    molecule = Chem.MolFromSmiles(smile)
    ion_flag = False
    two_char_abbr_flag = False
    two_digit_ring_flag = False
    ring_first_digit = 0
    ring_indices = []
    atom_index = 0
    smile_array = []
    if molecule:
        for character in smile:
            if re.match(r'[a-gi-z]', character, re.IGNORECASE):
                if two_char_abbr_flag:
                    two_char_abbr_flag = False
                else:
```

81

```python
        symbol_features, two_char_abbr_flag = __extract_atom_features(molecule, atom_index)
        smile_array.append(symbol_features)
        atom_index += 1
    elif re.match(r'[\\/.=#)(]', character):
        symbol_features = [character, 0, 0, 0, 0, 0, 0, 'CHI_UNSPECIFIED', 'UNSPECIFIED', 0]
        smile_array.append(symbol_features)
    elif re.match(r'[+-]', character):
        ion_flag = True
    elif re.match(r']', character):
        ion_flag = False
    elif re.match(r'%', character):
        two_digit_ring_flag = True
    elif re.match(r'[0-9]', character):
        if two_digit_ring_flag:
            if ring_first_digit == 0:
                ring_first_digit = character
                continue
            else:
                character = ring_first_digit + character
                ring_first_digit = 0
                two_digit_ring_flag = False
        if not ion_flag:
            symbol_features = ['ring']
            if character not in ring_indices:
                # Ring start.
                symbol_features.extend([0, 0, 0, 0, 0, 0, 'CHI_UNSPECIFIED', 'UNSPECIFIED', 1])
                ring_indices.append(character)
            else:
                # Ring end.
                symbol_features.extend([0, 0, 0, 0, 0, 0, 'CHI_UNSPECIFIED', 'UNSPECIFIED', 2])
            smile_array.append(symbol_features)
# 0-Padding
smile_array.extend([[0] * 10] * (max_length - len(smile_array)))
smile_array = pd.DataFrame(smile_array)

encoder = OneHotEncoder([['C', 'N', 'O', 'Br', 'Cl', 'F', 'P', 'S', 'ring', '(', ')', '/', '\\', '=', '#'],
            ['CHI_TETRAHEDRAL_CCW', 'CHI_TETRAHEDRAL_CW'],
            ['SP', 'SP2', 'SP3'], ['1', '2']], sparse=False, handle_unknown='ignore')

smile_array = np.concatenate([smile_array[[1, 2, 3, 4, 5, 6]].to_numpy(),
                encoder.fit_transform(smile_array[[0, 7, 8, 9]].astype(str))], axis=1)
return smile_array


def __extract_mol_structure(mol, conf_id, resolution, scale):
    digitizer = {'SINGLE': 1, 'AROMATIC': 2, 'DOUBLE': 3, 'TRIPLE': 4, 'C': 6, 'N': 7, 'O': 8, 'F': 9, 'P': 15, 'S': 16,
                'Cl': 17, 'Br': 35, 'Other': 40}
    pixel_scale = (scale[1] - scale[0]) / resolution
    matrix = np.zeros((resolution, resolution, 2), dtype='b')

    conformer = mol.GetConformer(conf_id)
    for atom in mol.GetAtoms():
        symbol = atom.GetSymbol()
        x = conformer.GetAtomPosition(atom.GetIdx()).x
        y = conformer.GetAtomPosition(atom.GetIdx()).y
        if x < scale[0] or x > scale[1] or y < scale[0] or y > scale[1]:
            return None

        j = int(np.floor((x - scale[0]) / pixel_scale))
```

```python
        i = int(np.floor((scale[1] - y) / pixel_scale))
        if symbol not in digitizer.keys():
            symbol = 'Other'

        matrix[i, j, 0] = digitizer[symbol]

    for bond in mol.GetBonds():
        bond_type = bond.GetBondType()
        x_start = conformer.GetAtomPosition(bond.GetBeginAtomIdx()).x
        y_start = conformer.GetAtomPosition(bond.GetBeginAtomIdx()).y
        j_start = int(np.floor((x_start - scale[0]) / pixel_scale))
        i_start = int(np.floor((scale[1] - y_start) / pixel_scale))
        x_end = conformer.GetAtomPosition(bond.GetEndAtomIdx()).x
        y_end = conformer.GetAtomPosition(bond.GetEndAtomIdx()).y
        j_end = int(np.floor((x_end - scale[0]) / pixel_scale))
        i_end = int(np.floor((scale[1] - y_end) / pixel_scale))
        pixel_coords = __pixelate(i_start, j_start, i_end, j_end)
        for pixel in pixel_coords[1:]:
            matrix[pixel[0], pixel[1], 1] = digitizer[str(bond_type)]

    return matrix

def __pixelate(x0, y0, x1, y1):
    pixel_coords = []
    if abs(y1 - y0) < abs(x1 - x0):
        if x0 > x1:
            x0, x1 = x1, x0
            y0, y1 = y1, y0
        dx = x1 - x0
        dy = y1 - y0
        yi = 1
        if dy < 0:
            yi = -1
            dy = -dy
        d = 2 * dy - dx
        y = y0
        for x in range(x0, x1):
            pixel_coords.append((x, y))
            if d > 0:
                y += yi
                d -= 2 * dx
            d += 2 * dy
    else:
        if y0 > y1:
            x0, x1 = x1, x0
            y0, y1 = y1, y0
        dx = x1 - x0
        dy = y1 - y0
        xi = 1
        if dx < 0:
            xi = -1
            dx = -dx
        d = 2 * dx - dy
        x = x0
        for y in range(y0, y1):
            pixel_coords.append((x, y))
            if d > 0:
                x += xi
```

```
        d -= 2 * dy
    d += 2 * dx


    return pixel_coords
```

## export.py

```python
from joblib import dump
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier

def export_model(score, classifier, x, y, filepath, study):
    try:
        if score > study.best_value:
            __export_models(classifier, x, y, filepath)
    except ValueError:
        __export_models(classifier, x, y, filepath)

def __export_models(classifier, x, y, filepath):
    classifier.fit(x, y)
    if type(classifier) == KerasClassifier:
        classifier.model.save(filepath)
    else:
        dump(classifier, filepath)
```

## pubchem.py

```python
import numpy as np
import requests


def get_assay_results(aid, tids=None):
    assay_results = []
    url = f'https://pubchem.ncbi.nlm.nih.gov/rest/pug/assay/aid/{aid}'
    request = requests.get(f'{url}/sids/json')
    sids = request.json()['InformationList']['Information'][0]['SID']
    limit = 10000
    batches = [sids[i * limit:(i + 1) * limit] for i in range((len(sids) + limit - 1) // limit)]
    for batch in batches:
        request = requests.post(f'{url}/json', data={'sid': ','.join(map(str, batch))})
        data = request.json()['PC_AssaySubmit']['data']
        for compound in data:
            if tids is None:
                props = [list(prop['value'].values())[0] for prop in compound['data']]
            else:
                props = [list(prop['value'].values())[0] for prop in compound['data'] if prop['tid'] in tids]
            assay_results.append([compound['sid']] + props)

    return np.array(assay_results, dtype=object)


def get_smile(sids):
    url = f'https://pubchem.ncbi.nlm.nih.gov/rest/pug/assay/substance/sid/cids/json'
    request = requests.post(url, data={'sid': ','.join(map(str, sids))})
    compounds = request.json()['InformationList']['Information']
    smile_table = np.ndarray((len(compounds), 2), dtype=object)
    for index in range(smile_table.shape[0]):
        smile_table[index, 0] = compounds[index]['CID'][0] if 'CID' in compounds[index] else None
```

```python
cids = smile_table[smile_table[:, 0] != None, 0].astype(int)
url = f'https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/cid/property/CanonicalSMILES/json'
request = requests.post(url, data={'cid': ','.join(map(str, cids))})
smiles = request.json()['PropertyTable']['Properties']
smiles_index = 0
for index in range(smile_table.shape[0]):
    smile = smiles[smiles_index]
    if smile_table[index, 0] == smile['CID']:
        smile_table[index, 1] = smile['CanonicalSMILES']
        smiles_index += 1
    else:
        smile_table[index, 1] = None

return smile_table
```

# Appendix B

# Data Retrieval

```python
import numpy as np
import pandas as pd

from featurize import generate_descriptors, generate_fingerprints, extract_smiles, extract_smile_structures
from pubchem import get_assay_results, get_smile

def create_labeled_dataset():
    assay_data = get_assay_results(aid='1851', tids={60, 68})
    assay_data = pd.DataFrame(assay_data, columns=['sid', 'score', 'curve_class'], dtype='object')

    smiles = get_smile(sids=assay_data.iloc[:, 0].astype(int))
    smiles = pd.DataFrame(smiles, columns=['cid', 'smile'])

    assay_data = pd.concat((smiles.smile, assay_data.iloc[:, [1, 2]]), axis=1).dropna()

    inhibitor = assay_data.loc[(assay_data.score >= 40) & assay_data.curve_class.isin({-1.1, -1.2, -2.1}), ['smile']]
    inhibitor['label'] = 'inhibitor'

    noninhibitor = assay_data.loc[(assay_data.score == 0) & (assay_data.curve_class == 4), ['smile']]
    noninhibitor['label'] = 'noninhibitor'

    return pd.concat((inhibitor, noninhibitor), axis=0).drop_duplicates('smile').reset_index(drop=True)

labeled_data = create_labeled_dataset()
mordred_features = generate_descriptors(labeled_data.smile.to_list())
fingerprints = generate_fingerprints(labeled_data.smile)
labeled_data = pd.concat([labeled_data, mordred_features, fingerprints], axis=1)
labeled_data.to_csv('data/cyp3a4_labeled_data.csv', index=False)

smile_features = extract_smiles(labeled_data.smile, max_length=250)
np.save('data/cyp3a4_smile_features', smile_features)

smile_structure = extract_smile_structures(labeled_data.smile, resolution=100, scale=(-15, 15))
np.save('data/cyp3a4_smile_structure', smile_structure)
```

# Appendix C

## Models

```python
import numpy as np
import pandas as pd
import optuna
import matplotlib.pyplot as plt
from imblearn import over_sampling, under_sampling, pipeline
from sklearn import ensemble, svm, linear_model, neural_network
from sklearn import impute, feature_selection, preprocessing, model_selection
from tensorflow.keras import Sequential, Input
from tensorflow.keras.layers import Conv1D, Conv2D, Dense, Dropout, Flatten, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.wrappers.scikit_learn import KerasClassifier

from export import export_model

df = pd.read_csv('data/cyp3a4_labeled_data.csv', low_memory=False)
features = df.drop(['smile', 'label'], axis=1)
smile_features = np.load('data/cyp3a4_smile_features.npy', allow_pickle=True)
smile_structure = np.load('data/cyp3a4_smile_structure.npy', allow_pickle=True)
labels = df.label.values.reshape(-1)

db_path = 'sqlite:///data/p450_ml.db'
best_scores = {'Logistic Regression': [], 'Random Forest': [], 'Support Vector Machine': [],
               'Neural Network': [], 'SMILE Auto-Extractor': [], 'Structure Auto-Extractor': []}

def create_pipe(trial):
    pipe = []
    pipe.append(impute.SimpleImputer())
    pipe.append(preprocessing.MinMaxScaler())
    pipe.append(feature_selection.VarianceThreshold(trial.suggest_uniform('var_thresh', 0, 0.25)))
    balance = trial.suggest_int('balance', 0, 2)
    if balance == 2:
        pipe.append(over_sampling.SMOTE())
    elif balance == 1:
        pipe.append(under_sampling.RandomUnderSampler())

    return pipe

def log_score(scores, name):
    try:
        if scores.mean() > study.best_value:
            best_scores[name] = scores
    except ValueError:
        best_scores[name] = scores

def objective(trial):
    pipe = create_pipe(trial)
    pipe.append(linear_model.LogisticRegression(C=trial.suggest_loguniform('c', 1e-5, 1e5)))
    classifier = pipeline.make_pipeline(*pipe)
    scores = model_selection.cross_val_score(classifier, features, labels, scoring='accuracy',
cv=model_selection.StratifiedKFold(3, shuffle=True), n_jobs=3)
    log_score(scores, 'Logistic Regression')
    export_model(scores.mean(), classifier, features, labels, 'models/lr-model.joblib', study)
    return scores.mean()
```

87

```python
study = optuna.create_study(study_name='lr', storage=db_path, direction='maximize', load_if_exists=True)
study.optimize(objective, n_trials=50)

fig = optuna.visualization.plot_optimization_history(study)
fig.show()
fig = optuna.visualization.plot_slice(study)
fig.show()

def objective(trial):
    pipe = create_pipe(trial)
    pipe.append(ensemble.RandomForestClassifier(max_features=trial.suggest_loguniform('max_features', 0.01,
1), n_estimators=trial.suggest_int('n_estimators', 1, 1000)))
    classifier = make_pipeline(*pipe)
    scores = model_selection.cross_val_score(classifier, features, labels, scoring='accuracy',
cv=model_selection.StratifiedKFold(3, shuffle=True), n_jobs=3)
    log_score(scores, 'Random Forest')
    export_model(scores.mean(), classifier, features, labels, 'models/rf-model.joblib', study)
    return scores.mean()

study = optuna.create_study(study_name='rf', storage=db_path, direction='maximize', load_if_exists=True)
study.optimize(objective, n_trials=50)
fig = optuna.visualization.plot_optimization_history(study)
fig.show()
fig = optuna.visualization.plot_slice(study)
fig.show()

def objective(trial):
    pipe = create_pipe(trial)
    pipe.append(svm.SVC(C=trial.suggest_loguniform('c', 1e-5, 1e5), gamma=trial.suggest_loguniform('gamma',
1e-5, 1e5), probability=True))
    classifier = make_pipeline(*pipe)
    scores = model_selection.cross_val_score(classifier, features, labels, scoring='accuracy',
cv=model_selection.StratifiedKFold(3, shuffle=True), n_jobs=3)
    log_score(scores, 'Support Vector Machine')
    export_model(scores.mean(), classifier, features, labels, 'models/svm-model.joblib', study)
    return scores.mean()

study = optuna.create_study(study_name='svm', storage=db_path, direction='maximize', load_if_exists=True)
study.optimize(objective, n_trials=15)
fig = optuna.visualization.plot_optimization_history(study)
fig.show()
fig = optuna.visualization.plot_slice(study)
fig.show()

def objective(trial):
    pipe = create_pipe(trial)
    layers = []
    for i in range(trial.suggest_int('layers', 1, 3)):
        n_units = trial.suggest_int(f'units_{i}', 1, 300)
        layers.append(n_units)

    pipe.append(neural_network.MLPClassifier(hidden_layer_sizes=tuple(layers),
                                             alpha=trial.suggest_loguniform('alpha', 1e-10, 1e10)))
    classifier = make_pipeline(*pipe)
    scores = model_selection.cross_val_score(classifier, features, labels, scoring='accuracy',
                                             cv=model_selection.StratifiedKFold(3, shuffle=True), n_jobs=3)
    log_score(scores, 'Neural Network')
```

```python
    export_model(scores.mean(), classifier, features, labels, 'models/nn-model.joblib', study)
    return scores.mean()


study = optuna.create_study(study_name='nn', storage=db_path, direction='maximize', load_if_exists=True)
study.optimize(objective, n_trials=50)
fig = optuna.visualization.plot_optimization_history(study)
fig.show()
fig = optuna.visualization.plot_slice(study)
fig.show()


def build_cnn_model_1d(cnn_layers=(64, 3, 1, 0.4), dense_layers=(32, 0.4), learning_rate=0.001,
                       shape=(250, 28)):
    model = Sequential()
    model.add(Input(shape=shape))

    for layer in cnn_layers:
        model.add(Conv1D(filters=layer[0], kernel_size=layer[1], strides=layer[2], activation='relu'))
        model.add(BatchNormalization(axis=2))
        if layer[3] > 0:
            model.add(Dropout(layer[3]))

    model.add(Flatten())

    for layer in dense_layers:
        model.add(Dense(units=layer[0], activation='relu'))
        model.add(BatchNormalization(axis=1))
        if layer[1] > 0:
            model.add(Dropout(layer[1]))

    model.add(Dense(units=1, activation='sigmoid'))
    model.compile(optimizer=Adam(lr=learning_rate), loss="binary_crossentropy", metrics=["accuracy"])

    return model

def objective(trial):
    cnn_layers = []
    for i in range(trial.suggest_int('cnn_layers', 1, 2)):
        filters = trial.suggest_int(f'filter_{i}', 1, 50)
        kernel = trial.suggest_int(f'kernel_{i}', 1, 5)
        stride = trial.suggest_int(f'stride_{i}', 1, 5)
        dropout = trial.suggest_uniform(f'dropout_cnn_{i}', 0.2, 0.5)
        cnn_layers.append((filters, kernel, stride, dropout))

    dense_layers = []
    for i in range(trial.suggest_int('dense_layers', 0, 1)):
        n_units = trial.suggest_int(f'unit_{i}', 1, 50)
        dropout = trial.suggest_uniform(f'dropout_nn_{i}', 0.2, 0.5)
        dense_layers.append((n_units, dropout))

    classifier = KerasClassifier(build_fn=build_cnn_model_1d, epochs=100, batch_size=32, learning_rate=0.0005,
                                 verbose=0, cnn_layers=tuple(cnn_layers), dense_layers=tuple(dense_layers))

    scores = model_selection.cross_val_score(classifier, smile_features, labels, scoring='accuracy',
                                             cv=model_selection.StratifiedKFold(3, shuffle=True))
    log_score(scores, 'SMILE Auto-Extractor')
    export_model(scores.mean(), classifier, smile_features, labels, 'models/cnn-model.h5', study)
    return scores.mean()
```

```python
study = optuna.create_study(study_name='cnn', storage=db_path, direction='maximize', load_if_exists=True)
study.optimize(objective, n_trials=50)
fig = optuna.visualization.plot_optimization_history(study)
fig.show()
fig = optuna.visualization.plot_slice(study)
fig.show()

def build_cnn_model_2d(cnn_layers=(64, 3, 1, 0.4), dense_layers=(32, 0.4), learning_rate=0.001,
                       shape=(100, 100, 2)):
    model = Sequential()
    model.add(Input(shape=shape))

    for layer in cnn_layers:
        model.add(Conv2D(filters=layer[0], kernel_size=layer[1], strides=layer[2], padding="same",
                         activation='relu'))
        model.add(BatchNormalization(axis=2))
        if layer[3] > 0:
            model.add(Dropout(layer[3]))

    model.add(Flatten())

    for layer in dense_layers:
        model.add(Dense(units=layer[0], activation='relu'))
        model.add(BatchNormalization(axis=1))
        if layer[1] > 0:
            model.add(Dropout(layer[1]))

    model.add(Dense(units=1, activation='sigmoid'))
    model.compile(optimizer=Adam(lr=learning_rate), loss="binary_crossentropy", metrics=["accuracy"])

    return model

def objective(trial):
    cnn_layers = []
    for i in range(trial.suggest_int('cnn_layers', 1, 6)):
        filters = trial.suggest_int(f'filter_{i}', 1, 20)
        kernel = trial.suggest_int(f'kernel_{i}', 1, 25)
        stride = trial.suggest_int(f'stride_{i}', 1, 5)
        dropout = trial.suggest_uniform(f'dropout_cnn_{i}', 0.2, 0.7)
        cnn_layers.append((filters, kernel, stride, dropout))

    dense_layers = []
    for i in range(trial.suggest_int('dense_layers', 0, 1)):
        n_units = trial.suggest_int(f'unit_{i}', 75, 250)
        dropout = trial.suggest_uniform(f'dropout_nn_{i}', 0, 0.5)
        dense_layers.append((n_units, dropout))

    classifier = KerasClassifier(build_fn=build_cnn_model_2d, epochs=75, batch_size=64, learning_rate=0.001,
                                 verbose=0, cnn_layers=tuple(cnn_layers), dense_layers=tuple(dense_layers))

    scores = model_selection.cross_val_score(classifier, smile_structure, labels, scoring='accuracy',
                                             cv=model_selection.StratifiedKFold(3, shuffle=True))
    log_score(scores, 'Structure Auto-Extractor')
    export_model(scores.mean(), classifier, smile_structure, labels, 'models/2d-cnn-model.h5', study)
    return scores.mean()

study = optuna.create_study(study_name='2d-cnn', storage=db_path, direction='maximize', load_if_exists=True)
study.optimize(objective, n_trials=50)
```

```
fig = optuna.visualization.plot_optimization_history(study)
fig.show()
fig = optuna.visualization.plot_slice(study)
fig.show()

scores = list(best_scores.values())[:4]
labels = list(best_scores.keys())[:4]

plt.figure(figsize=(6, 3), dpi=300)
plt.rcParams.update({'font.size': 7})
plt.xticks(rotation=0)
plt.ylim(0.74, 0.86)
plt.ylabel('3-Fold Cross-Validation Accuracy')
plt.boxplot(scores, labels=labels, boxprops={"linewidth": 0.8},
            medianprops={"color": 'black', "linewidth": 0.8},
            whiskerprops={"linewidth": 0.8}, capprops={"linewidth": 0.8})
for i in range(len(scores)):
    y = scores[i]
    x = np.random.normal(i+0.7, 0, size=len(y))
    plt.plot(x, y, 'k.', markersize=2)
plt.savefig('images/cv_accuracies_standard.svg')

scores = list(best_scores.values())
scores = [scores[i] for i in [1, 4, 5]]
labels = list(best_scores.keys())
labels = [labels[i] for i in [1, 4, 5]]
plt.figure(figsize=(6, 3), dpi=300)
plt.rcParams.update({'font.size': 7})
plt.xticks(rotation=0)
plt.ylim(0.74, 0.86)
plt.ylabel('3-Fold Cross-Validation Accuracy')
plt.boxplot(scores, labels=labels, boxprops={"linewidth": 0.8},
            medianprops={"color": 'black', "linewidth": 0.8},
            whiskerprops={"linewidth": 0.8}, capprops={"linewidth": 0.8})
for i in range(len(scores)):
    y = scores[i]
    x = np.random.normal(i+0.7, 0, size=len(y))
    plt.plot(x, y, 'k.', markersize=2)
plt.savefig('images/cv_accuracies_auto_extract.svg')

study = optuna.load_study(study_name='lr', storage=db_path)
study.best_params

study = optuna.load_study(study_name='rf', storage=db_path)
study.best_params

study = optuna.load_study(study_name='svm', storage=db_path)
study.best_params

study = optuna.load_study(study_name='nn', storage=db_path)
study.best_params

study = optuna.load_study(study_name='cnn', storage=db_path)
study.best_params

study = optuna.load_study(study_name='2d-cnn', storage=db_path)
study.best_params
```

# Appendix D

# Exploration

```python
import pandas as pd
import matplotlib.pyplot as plt
from joblib import load
import shap
import os
import warnings
warnings.filterwarnings('ignore')

df = pd.read_csv('data/cyp3a4_labeled_data.csv', low_memory=False)
features = df.drop(['smile', 'label'], axis=1).astype(float)

def adjust_plot(title, colorbar_label):
    plt.title(title, fontsize=11)
    ax = plt.gcf().axes[0]
    for label in (ax.get_xticklabels() + ax.get_yticklabels()):
        label.set_fontsize(8)
    for label in [ax.xaxis.label, ax.yaxis.label]:
        label.set_fontsize(10)
    ax = plt.gcf().axes[1]
    for label in (ax.get_yticklabels()):
        label.set_fontsize(8)
    if colorbar_label:
        ax.yaxis.label.set_fontsize(10)
    else:
        ax.set_ylabel('')
    plt.tight_layout()

def shap_feature_importance(features, model, title, save_path, sample_size=100):
    samples = shap.sample(features, sample_size)
    explainer = shap.KernelExplainer(model.predict_proba, samples)
    shap_values = explainer.shap_values(samples, nsamples=5, l1_reg="aic")
    shap.summary_plot(shap_values[0], samples, feature_names=samples.columns, max_display=5,
                      plot_size=(6, 2), show=False)
    plt.xlabel('SHAP Value')
    adjust_plot(title, False)
    plt.savefig(f"{save_path}_feature_importance.svg", dpi=300)
    plt.show()
    shap.dependence_plot('rank(0)', shap_values[0], samples, interaction_index='rank(1)', show=False)
    adjust_plot(title, True)
    plt.savefig(f"{save_path}_dependence.svg", dpi=300)
    plt.show()

titles = ['Logistic Regression', 'Neural Network', 'Random Forest', 'Support Vector Machine']
model_paths = os.listdir('models')
index = 0
for model_path in model_paths:
    if '.h5' in model_path:
        pass
    else:
        model = load(f'models/{model_path}')
        print(model_path)
        shap_feature_importance(features, model, titles[index], f"images/{model_path.split('.')[0]}")
        index += 1
```

# Appendix E

## Test

```python
import os
import numpy as np
import pandas as pd
import plotly.figure_factory as ff
import plotly.graph_objects as go
import matplotlib.pyplot as plt
import shap
from joblib import load
from tensorflow.keras.models import load_model
from featurize import generate_descriptors, generate_fingerprints, extract_smiles, extract_smile_structures
shap.initjs()
import warnings
warnings.filterwarnings('ignore')

test_set = pd.read_csv('data/fda_test.csv', low_memory=False)
mordred_features = generate_descriptors(test_set.smile.to_list())
fingerprints = generate_fingerprints(test_set.smile)
test_set = pd.concat([test_set, mordred_features, fingerprints], axis=1)
features = test_set.drop(['name', 'smile', 'type'], axis=1).astype(float)
smile_features = extract_smiles(test_set.smile, max_length=250).astype(float)
smile_structure = extract_smile_structures(test_set.smile, resolution=100, scale=(-15, 15)).astype(float)
meta = test_set[['name', 'type']]

def image_plot(shap_values, labels, figsize):
    fig, axes = plt.subplots(nrows=shap_values.shape[0], ncols=1, figsize=figsize)
    for row in range(shap_values.shape[0]):
        abs_vals = np.abs(shap_values.sum(-1)).flatten()
        max_val = np.nanpercentile(abs_vals, 99.9)
        axes[row].set_title(labels[row], fontsize=11)
        sv = shap_values[row].sum(-1)
        im = axes[row].imshow(sv, cmap=shap.plots.colors.red_transparent_blue, vmin=-max_val, vmax=max_val)
        for label in (axes[row].get_xticklabels() + axes[row].get_yticklabels()):
            label.set_fontsize(8)
    fig.subplots_adjust(wspace=0, hspace=0.3)
    cb = fig.colorbar(im, ax=np.ravel(axes).tolist(), label="SHAP value", orientation="horizontal",
                      aspect=figsize[0]/0.2, pad=0.08)
    cb.ax.xaxis.label.set_fontsize(10)
    for label in (cb.ax.get_xticklabels()):
        label.set_fontsize(8)
    cb.outline.set_visible(False)

def plot_radar(values_best, values_worse, categories):
    fig = go.Figure()
    fig.add_trace(go.Scatterpolar(r=values_best, theta=categories, fill='toself', name='Best Prediction'))
    fig.add_trace(go.Scatterpolar(r=values_worse, theta=categories, fill='toself', name='Worse Prediction'))
    fig.update_layout(showlegend=True, autosize=False, width=500, height=500)
    fig.show()
model_paths = os.listdir('models')
results = np.ndarray((meta.shape[0] * len(model_paths), 4), dtype=object)
for model_index, model_path in enumerate(model_paths):
    print(model_path)
    if '.h5' in model_path:
        model = load_model(f'models/{model_path}')
```

```python
    if '2d' in model_path:
        predictions = model.predict_proba(smile_structure)[:, 0]
        sorted_indices = np.argsort(predictions, axis=0)
        index = [sorted_indices[-1], sorted_indices[1]]
        explainer = shap.GradientExplainer(model, smile_structure)
        shap_values = explainer.shap_values(smile_structure)
        plt.figure(figsize=(2.5, 2.5))
        plt.matshow(np.amax(smile_structure[index[0],:,:,:], 2), cmap=plt.cm.gray_r, fignum=1)
        plt.savefig(f"images/example_structure_1.svg", dpi=300)
        plt.figure(figsize=(2.5, 2.5))
        plt.matshow(np.amax(smile_structure[index[1],:,:,:], 2), cmap=plt.cm.gray_r, fignum=2)
        plt.savefig(f"images/example_structure_2.svg", dpi=300)
        image_plot(shap_values[0][index], meta.name[index].values, figsize=(3,8))
        plt.savefig(f"images/2d_shap.svg", dpi=300)
        plt.show()
    else:
        predictions = model.predict_proba(smile_features)[:, 0]
else:
    model = load(f'models/{model_path}')
    predictions = model.predict_proba(features)[:, 0]
    sorted_indices = np.argsort(predictions, axis=0)
    index = [sorted_indices[-1], sorted_indices[1]]
    explainer = shap.KernelExplainer(model.predict_proba, features)
    shap_values = explainer.shap_values(features, nsamples=50)
    best = True
    for i in index:
        print(meta.name[i])
        shap.force_plot(explainer.expected_value[0], shap_values[0][i,:], features.iloc[i,:].values,
                list(features.columns.astype(str)),  matplotlib=True, show=False, figsize=(20, 3.5), text_rotation=10)
        plt.title(meta.name[i], fontsize=12)
        plt.tight_layout()
        plt.savefig(f"images/{model_path.split('.')[0]}_force_{'best' if best else 'worst'}.svg", dpi=300)
        plt.show()
        best = False
    for pred_index, pred in enumerate(predictions):
        index = pred_index + (meta.shape[0] * model_index)
        results[index, :] = [meta.iloc[pred_index, 0], meta.iloc[pred_index, 1], model_path.split('-')[0], pred]
results = pd.DataFrame(results, columns=['name', 'type', 'model', 'inhibitor_conf'])
heatmap = None
for strength in ['strong', 'moderate', 'weak']:
    temp = results[results.type == strength]
    temp = temp.pivot(index='name', columns='model', values='inhibitor_conf').reset_index()
    temp['name'] = temp.name + ' [' + strength + ']'
    heatmap = pd.concat([heatmap, temp])
plt.figure(figsize=(6, 6))
plt.rcParams.update({'font.size': 7})
x = ['CNN Structure', 'CNN SMILE', 'LR', 'NN', 'RF', 'SVM']
y = heatmap['name'].to_list()
z = heatmap.values[:, 1:].astype(float)
im = plt.imshow(z, aspect='auto')
plt.xticks(np.arange(len(x)), x)
plt.yticks(np.arange(len(y)), y)
for i in range(len(y)):
    for j in range(len(x)):
        text = plt.text(j, i, round(z[i, j], 2), ha="center", va="center", color="w")
plt.tight_layout()
plt.savefig(f"images/fda_test.svg", dpi=300)
plt.show()
```