

DRUG DISCOVERY

##### **A PROJECT REPORT**

###### ***Submitted by***

##### **YATHIN. S [REGISTER NO: 211417104309]**

**SATHIYAN. K [REGISTER NO: 211417104249]**

**SANTHOSH. P [REGISTER NO: 211417104241]**

***in partial fulfillment for the award of the degree***

***of***

**BACHELOR OF ENGINEERING**

IN

# **COMPUTER SCIENCE AND ENGINEERING**

**PANIMALAR ENGINEERING COLLEGE, CHENNAI-600123.**

**ANNA UNIVERSITY: CHENNAI 600 025**

##### **APRIL 2021**

**BONAFIDE CERTIFICATE**

Certified that this project report **“DRUG DISCOVERY”** is the bonafide work of “**SANTHOSH. P [211417104241]”** who carried out the project work under my supervision.

**SIGNATURE SIGNATURE**

**Dr.S.MURUGAVALLI,M.E.,Ph.D., Mr.A.KARTHIKEYAN**

**HEAD OF THE DEPARTMENT SUPERVISOR**

**ASSOCIATE PROFESSOR**

DEPARTMENT OF CSE, DEPARTMENT OF CSE,

PANIMALAR ENGINEERING COLLEGE, PANIMALAR ENGINEERING COLLEGE,

NASARATHPETTAI, NASARATHPETTAI,

POONAMALLEE, POONAMALLEE,

CHENNAI-600 123. CHENNAI-600 123

Certified that the above candidate(s) was/ were examined in the Anna University Project Viva-Voce Examination held on...........................

**INTERNAL EXAMINER EXTERNAL EXAMINER**

**ACKNOWLEDGEMENT**

We express our deep gratitude to our respected Secretary and Correspondent **Dr.P.CHINNADURAI, M.A., Ph.D.** for his kind words and enthusiastic motivation, which inspired us a lot in completing this project.

We would like to extend our heartfelt and sincere thanks to our Directors **Tmt.C.VIJAYARAJESWARI**, **Thiru.C.SAKTHIKUMAR,M.E.,Tmt. SARANYASREE SAKTHIKUMAR B.E.,M.B.A.,** for providing us with the necessary facilities for completion of this project.

We also express our gratitude to our Principal **Dr.K.Mani, M.E., Ph.D.** for his timely concern and encouragement provided to us throughout the course.

We thank the HOD of CSE Department, **Dr. S.MURUGAVALLI , M.E.,Ph.D.,** for the support extended throughout the project.

We would like to thank my Project Guide**Mr. A.Karthikeyan** and all the faculty members of the Department of CSE for their advice and suggestions for the successful completion of the project.

**ABSTRACT**

The advancements of information technology and related processing techniques have created a fertile base for progress in many scientific fields and industries. In the fields of drug discovery and development, machine learning techniques have been used for the development of novel drug candidates. The methods for designing drug targets and novel drug discovery now routinely combine machine learning to enhance the efficiency, efficacy, and quality of developed outputs. The generation and incorporation of big data, through technologies such as high-throughput screening and high through-put computational analysis of databases used for both lead and target discovery, has increased the reliability of the machine learning and deep learning incorporated techniques. The use of these virtual screening and encompassing online information has also been highlighted in developing lead synthesis pathways

**TABLE OF CONTENTS**

| **CHAPTER NO.** | **TITLE** | **PAGE NO.** |
| --- | --- | --- |
|  | **ABSTRACT** | iii |
|  | **LIST OF TABLES** | vii |
|  | **LIST OF FIGURES** | viii |
|  | **LIST OF SYMBOLS, ABBREVIATIONS** | ix |
| **1.** | **INTRODUCTION** |  |
|  | 1.1 Overview | 7 |
| **2.** | **LITERATURE SURVEY** | 8 |
| **3.** | **SYSTEM ANALYSIS** |  |
|  | 3.1 Existing System | 11 |
|  | 3.2 Proposed system | 11 |
|  | 3.3 Hardware Requirement | 12 |
|  | 3.4 Software Requirement | 12 |
|  | 3.5 Software Specification |  |
|  | 3.5.1 Machine Learning | 13 |
|  | 3.5.2 Anaconda | 20 |
| **4.** | **SYSTEM ARCHITECTURE** |  |
|  | 4.1. Architecture Overview | 24 |
|  | 4.2 Usecase Diagram | 25 |
|  | 4.3 Activity Diagram | 26 |
| **5.** | **SYSTEM MODULE** |  |
|  | 5.1 Module | 27 |
|  | 5.2 Module Description | 27 |
| **6.** | **SYSTEM I****MPLEMENTATION** |  |
|  | 6.1 Sample Coding | 29 |
| **7.** | **SYSTEM TESTING** |  |
|  | 7.1 Testing Technique | 53 |
|  | 7.1.1 White Box Testing | 53 |
|  | 7.1.2 Black Box Testing | 54 |
|  | 7.2 Software Testing Strategies | 54 |
| **8.** | **CONCLUSION** |  |
|  | 8.1 Conclusion | 55 |
|  | **APPENDICES** |  |
|  | A.1 Sample Screens | 56 |
|  | **REFERENCES** | 59 |

**CHAPTER 1**

**INTRODUCTION**

A great variety of experimental data, at a chemical, transcriptomic, or genomic level is available to readily use for drug development. Summarizing the huge amount of biological data at hand into meaningful models, to grasp the full mechanism of diseases, seems harder and harder. However, systems biology and machine learning approaches are continuously enhanced in order to accelerate the path to efficient drug development. We will focus on three significant related and intermingled questions, that can be subject to automation: drug discovery, drug testing, and drug repurposing. Firstly, this review briefly dwells on the current context in drug development. Later, we will review generic machine learning algorithms, and more specifically, we will focus on sequential learning algorithms and recommender systems. These algorithms have also proven themselves useful in other research fields, and are active biomedical fields of research.

**CHAPTER 2**

**Literature survey**

# **[1] Applications of machine learning in drug discovery and development**

Drug discovery and development pipelines are long, complex and depend on numerous factors. Machine learning (ML) approaches provide a set of tools that can improve discovery and decision making for well-specified questions with abundant, high-quality data. Opportunities to apply ML occur in all stages of drug discovery. Examples include target validation, identification of prognostic biomarkers and analysis of digital pathology data in clinical trials. Applications have ranged in context and methodology, with some approaches yielding accurate predictions and insights. The challenges of applying ML lie primarily with the lack of interpretability and repeatability of ML-generated results, which may limit their application. In all areas, systematic and comprehensive high-dimensional data still need to be generated. With ongoing efforts to tackle these issues, as well as increasing awareness of the factors needed to validate ML approaches, the application of ML can promote data-driven decision making and has the potential to speed up the process and reduce failure rates in drug discovery and development

**[2] Survey of Machine Learning Techniques in Drug Discovery**

 Drug discovery, which is the process of discovering new candidate medications, is very important for pharmaceutical industries. At its current stage, discovering new drugs is still a very expensive and time-consuming process, requiring Phases I, II and III for clinical trials. Recently, machine learning techniques in Artificial Intelligence (AI), especially the deep learning techniques which allow a computational model to generate multiple layers, have been widely applied and achieved state-of-the-art performance in different fields, such as speech recognition, image classification, bioinformatics, etc. One very important application of these AI techniques is in the field of drug discovery.

# [3]**From machine learning to deep learning: progress in machine intelligence for rational drug discovery**

Machine intelligence, which is normally presented as artificial intelligence, refers to the intelligence exhibited by computers. In the history of rational drug discovery, various machine intelligence approaches have been applied to guide traditional experiments, which are expensive and timeconsuming. Over the past several decades, machine-learning tools, such as quantitative structure– activity relationship (QSAR) modeling, were developed that can identify potential biological active molecules from millions of candidate compounds quickly and cheaply. However, when drug discovery moved into the era of ‘big’ data, machine learning approaches evolved into deep learning approaches, which are a more powerful and efficient way to deal with the massive amounts of data generated from modern drug discovery approaches. Here, we summarize the history of machine learning and provide insight into recently developed deep learning approaches and their applications in rational drug discovery. We suggest that this evolution of machine intelligence now provides a guide for early-stage drug design and discovery in the current big data era.

# **[4] Exploiting machine learning for end-to-end drug discovery and development**

# A variety of machine learning methods such as naive Bayesian, support vector machines and more recently deep neural networks are demonstrating their utility for drug discovery and development. These leverage the generally bigger datasets created from high-throughput screening data and allow prediction of bioactivities for targets and molecular properties with increased levels of accuracy. We have only just begun to exploit the potential of these techniques but they may already be fundamentally changing the research process for identifying new molecules and/or repurposing old drugs. The integrated application of such machine learning models for end-to-end (E2E) application is broadly relevant and has considerable implications for developing future therapies and their targeting.

# **[5] Machine-learning approaches in drug discovery: methods and applications**

During the past decade, virtual screening (VS) has evolved from traditional similarity searching, which utilizes single reference compounds, into an advanced application domain for data mining and machine-learning approaches, which require large and representative training-set compounds to learn robust decision rules. The explosive growth in the amount of public domain-available chemical and biological data has generated huge effort to design, analyze, and apply novel learning methodologies. Here, I focus on machine-learning techniques within the context of ligand-based VS (LBVS). In addition, I analyze several relevant VS studies from recent publications, providing a detailed view of the current state-of-the-art in this field and highlighting not only the problematic issues, but also the successes and opportunities for further advances.

**CHAPTER 3**

**SYSTEM ANALYSIS**

**3.1 EXISTING SYSTEM**

The advancements of information technology and related processing techniques have created a fertile base for progress in many scientific fields and industries. In the fields of drug discovery and development, machine learning techniques have been used for the development of novel drug candidates. The methods for designing drug targets and novel drug discovery now routinely combine machine learning and deep learning algorithms to enhance the efficiency, efficacy, and quality of developed outputs. The generation and incorporation of big data, through technologies such as high-throughput screening and high through-put computational analysis of databases used for both lead and target discovery, has increased the reliability of the machine learning and deep learning incorporated techniques. The use of these virtual screening and encompassing online information has also been highlighted in developing lead synthesis pathways. In this review, machine learning and deep learning algorithms utilized in drug discovery and associated techniques will be discussed. The applications that produce promising results and methods will be reviewed.

**3.2 PROPOSED SYSTEM**

Drug discovery and development pipelines are long, complex and depend on numerous factors. Machine learning (ML) approaches provide a set of tools that can improve discovery and decision making for well-specified questions with abundant, high-quality data. Opportunities to apply ML occur in all stages of drug discovery.

Examples include target validation, identification of prognostic biomarkers and analysis of digital pathology data in clinical trials. Applications have ranged in context and methodology, with some approaches yielding accurate predictions and insights.

The challenges of applying ML lie primarily with the lack of interpretability and repeatability of ML-generated results, which may limit their application.

In all areas, systematic and comprehensive high-dimensional data still need to be generated.

With ongoing efforts to tackle these issues, as well as increasing awareness of the factors needed to validate ML approaches, the application of ML can promote data-driven decision making and has the potential to speed up the process and reduce failure rates in drug discovery and development.

**3.3 HARDWARE REQUIREMENTS**

Processor : Intel Pentium Dual Core 2.00GHz

Hard disk : 500 GB

RAM : 8 GB (minimum)

**3.5 SOFTWARE REQUIREMENTS**

* Python 3.6.4 Version

**3.6 SOFTWARE SPECIFICATION**

**3.6.1 Machine learning**

**Machine learning** (**ML**) is the study of computer [algorithms](https://en.wikipedia.org/wiki/Algorithm) that improve automatically through experience and by the use of data. It is seen as a part of [artificial intelligence](https://en.wikipedia.org/wiki/Artificial_intelligence). Machine learning algorithms build a model based on sample data, known as "[training data](https://en.wikipedia.org/wiki/Training_data)", in order to make predictions or decisions without being explicitly programmed to do so. Machine learning algorithms are used in a wide variety of applications, such as [email filtering](https://en.wikipedia.org/wiki/Email_filtering) and [computer vision](https://en.wikipedia.org/wiki/Computer_vision), where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks.

The types of machine learning algorithms are mainly divided into four categories:

* **Supervised learning,**
* **Un-supervised learning,**
* **Semi-supervised learning,**
* **Reinforcement learning**.
* **Supervised learning**

Supervised learning algorithms build a mathematical model of a set of data that contains both the inputs and the desired outputs. The data is known as [training data](https://en.wikipedia.org/wiki/Training_data), and consists of a set of training examples. Each training example has one or more inputs and the desired output, also known as a supervisory signal. In the mathematical model, each training example is represented by an [array](https://en.wikipedia.org/wiki/Array_data_structure) or vector, sometimes called a feature vector, and the training data is represented by a [matrix](https://en.wikipedia.org/wiki/Matrix_(mathematics)). Through [iterative optimization](https://en.wikipedia.org/wiki/Mathematical_optimization#Computational_optimization_techniques) of an [objective function](https://en.wikipedia.org/wiki/Loss_function), supervised learning algorithms learn a function that can be used to predict the output associated with new inputs. An optimal function will allow the algorithm to correctly determine the output for inputs that were not a part of the training data. An algorithm that improves the accuracy of its outputs or predictions over time is said to have learned to perform that task.

**CLASSIFICATION**

As the name suggests, Classification is the task of “classifying things” into sub- categories. But, by a machine. If that doesn’t sound like much, imagine your computer being able to differentiate between you and a stranger. Between a potato and a tomato. Between an A grade and a F. In Machine Learning and Statistics, Classification is the problem of identifying to which of a set of categories (sub populations), a new observation belongs to, on the basis of a training set of data containing observations and whose categories membership is known.

**TYPES OF CLASSIFICATION**

Classification is of two types:

* Binary Classification
* Multiclass Classification

**Binary Classification**

When we have to categorize given data into 2 distinct classes. Example – On the basis of given health conditions of a person, we have to determine whether the person has a certain disease or not.

**Multiclass Classification**

The number of classes is more than 2. For Example

– On the basis of data about different species of flowers, we have to determine which specie does our observation belong to

Fig 2 : Binary and Multiclass Classification. Here x1 and x2 are our variables upon which the class is predicted.Suppose we have to predict whether a given patient has a certain disease or not, on the basis of 3 variables, called features. Which means there are two possible outcomes:

1. The patient has the said disease. Basically a result labelled “Yes” or “True”.

2. The patient is disease free. A result labelled “No” or “False”.

This is a binary classification problem.We have a set of observations called training data set, which comprises of sample data with actual classification results. We train a model, called Classifier on this data set, and use that model to predict whether a certain patient will have the

1. X : pre-classified data, in the form of a N\*M matrix. N is the no. of observations and M is the number of features

2. y : An N-d vector corresponding to predicted classes for each of the N observations.

3. Feature Extraction : Extracting valuable information from input X using a series of transforms.

4. ML Model : The “Classifier” we’ll train.

5. y’ : Labels predicted by the Classifier.

6. Quality Metric : Metric used for measuring the performance of the model.

7. ML Algorithm : The algorithm that is used to update weights w’, which update the model and “learns” iteratively.

Types of Classifiers (Algorithms)

There are various types of classifiers. Some of them are :

• Linear Classifiers : Logistic Regression

• Tree Based Classifiers : Decision Tree Classifier

• Support Vector Machines

• Artificial Neural Networks

• Bayesian Regression

• Gaussian Naive Bayes Classifiers

• Stochastic Gradient Descent (SGD) Classifier

• Ensemble Methods : Random Forests, AdaBoost, Bagging Classifier, Voting Classifier, ExtraTrees Classifier

Practical Applications of Classification

• Google’s self driving car uses deep learning enabled classification techniques which enables it to detect and classify obstacles.

• Spam E-mail filtering is one of the most widespread and well recognized uses of Classification techniques.

• Detecting Health Problems, Facial Recognition, Speech Recognition, Object Detection, Sentiment Analysis all use Classification at their core.

**REGRESSION**

A regression problem is when the output variable is a real or continuous value, such as “salary” or “weight”. Many different models can be used, the simplest is the linear regression. It tries to fit data with the best hyper-plane which goes through the points.

* **Un-supervised learning**

Un-supervised learning algorithms take a set of data that contains only inputs, and find structure in the data, like grouping or clustering of data points. The algorithms, therefore, learn from test data that has not been labeled, classified or categorized. Instead of responding to feedback, unsupervised learning algorithms identify commonalities in the data and react based on the presence or absence of such commonalities in each new piece of data. A central application of unsupervised learning is in the field of [density estimation](https://en.wikipedia.org/wiki/Density_estimation) in [statistics](https://en.wikipedia.org/wiki/Statistics), such as finding the [probability density function](https://en.wikipedia.org/wiki/Probability_density_function). Though unsupervised learning encompasses other domains involving summarizing and explaining data features.

**CLUSTERING**

It is basically a type of unsupervised learning method. An unsupervised learning method is a method in which we draw references from datasets consisting of input data without labelled responses. Generally, it is used as a process to find meaningful structure, explanatory underlying processes, generative features, and groupings inherent in a set of examples. Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups. It is basically a collection of objects on the basis of similarity and dissimilarity between them.For example, the data points in the graph below clustered together can be classified into one single group. We can distinguish the clusters,

These data points are clustered by using the basic concept that the data point lies within the given constraint from the cluster center. Various distance methods and techniques are used for calculation of the outliers.

Clustering is very much important as it determines the intrinsic grouping among the unlabeled data present. There are no criteria for a good clustering. It depends on the user, what is the criteria they may use which satisfy their need. For instance, we could be interested in finding representatives for homogeneous groups (data reduction), in finding “natural clusters” and describe their unknown properties (“natural” data types), in finding useful and suitable groupings (“useful” data classes) or in finding unusual data objects (outlier detection). This algorithm must make some assumptions which constitute the similarity of points and each assumption make different and equally valid clusters.

**3.6.5.1.1 Clustering Methods :**

1. **Density-Based Methods :** These methods consider the clusters as the dense region having some similarity and different from the lower dense region of the space. These methods have good accuracy and ability to merge two clusters.Example DBSCAN

(Density-Based Spatial Clustering of Applications with Noise) , OPTICS (Ordering Points to Identify Clustering Structure) etc.

2. **Hierarchical Based Methods :** The clusters formed in this method forms a tree type structure based on the hierarchy. New clusters are formed using the previously formed one. It is divided into two category

• Agglomerative (bottom up approach)

• Divisive (top down approach) .

3. **Partitioning Methods :** These methods partition the objects into k clusters and each partition forms one cluster. This method is used to optimize an objective criterion similarity function such as when the distance is a major parameter example K-means, CLARANS (Clustering Large Applications based upon randomized Search) etc.

4. **Grid-based Methods :** In this method the data space are formulated into a finite number of cells that form a grid-like structure. All the clustering operation done on these grids are fast and independent of the number of data objects example STING (Statistical Information Grid), wave cluster, CLIQUE (CLusteringIn Quest) etc.

Clustering Algorithms:

• K-Means Clustering.

• Mean-Shift Clustering for a single sliding window.

• The entire process of Mean-Shift Clustering.

• DBSCAN Smiley Face Clustering.

• EM Clustering using GMMs.

• Agglomerative Hierarchical Clustering.

* **Semi-supervised learning**

Semi-supervised learning falls between [unsupervised learning](https://en.wikipedia.org/wiki/Unsupervised_learning) (without any labeled training data) and [supervised learning](https://en.wikipedia.org/wiki/Supervised_learning) (with completely labeled training data). Some of the training examples are missing training labels, yet many machine-learning researchers have found that unlabeled data, when used in conjunction with a small amount of labeled data, can produce a considerable improvement in learning accuracy.

* **Reinforcement learning**

Reinforcement learning is an area of machine learning concerned with how [software agents](https://en.wikipedia.org/wiki/Software_agent) ought to take [actions](https://en.wikipedia.org/wiki/Action_selection) in an environment so as to maximize some notion of cumulative reward. Due to its generality, the field is studied in many other disciplines, such as [game theory](https://en.wikipedia.org/wiki/Game_theory), [control theory](https://en.wikipedia.org/wiki/Control_theory), [operations research](https://en.wikipedia.org/wiki/Operations_research), [information theory](https://en.wikipedia.org/wiki/Information_theory), [simulation-based optimization](https://en.wikipedia.org/wiki/Simulation-based_optimization), multi-agent systems, [swarm intelligence](https://en.wikipedia.org/wiki/Swarm_intelligence), [statistics](https://en.wikipedia.org/wiki/Statistics) and [genetic algorithms](https://en.wikipedia.org/wiki/Genetic_algorithm). In machine learning, the environment is typically represented as a [Markov decision process](https://en.wikipedia.org/wiki/Markov_decision_process) (MDP). Many reinforcement learning algorithms use [dynamic programming](https://en.wikipedia.org/wiki/Dynamic_programming) techniques. Reinforcement learning algorithms do not assume knowledge of an exact mathematical model of the MDP, and are used when exact models are infeasible. Reinforcement learning algorithms are used in autonomous vehicles or in learning to play a game against a human opponent.

**3.6.2 ANACONDA**

Anaconda is a free and open source distribution of the Python and R programming languages for data science and machine learning related applications (large-scale data processing, predictive analytics, scientific computing), that aims to simplify package management and deployment. Package versions are managed by the package management system conda. Anaconda Distribution is used by over 6 million users, and it includes more than 250 popular data science packages suitable for Windows, Linux, and MacOS.

Python is a high-level programming language devised by Guido van Rossum & first released in 1991. It’s the most popular coding language used by software developers to build, control, manage and for testing. It is also an interpreter which executes Python programs. The python interpreter is called python.exe on Windows.

**Python Packages**

Packages or additional libraries help in scientific computing and computational modelling. In Python, the packages are not the part of the Python standard library. Few major packages are –

numpy (NUMeric Python): matrices and linear algebra

scipy (SCIentific Python): many numerical routines

matplotlib: (PLOTting LIBrary) creating plots of data

sympy (SYMbolic Python): symbolic computation

pytest (Python TESTing): a code testing framework

Together with a list of Python packages, tools like editors, Python distributions include the Python interpreter. Anaconda is one of several Python distributions. Anaconda is a new distribution of the Python and R data science package. It was formerly known as Continuum Analytics. Anaconda has more than 100 new packages.

This work environment, Anaconda is used for scientific computing, data science, statistical analysis, and machine learning. The latest version of Anaconda 5.0.1 is released in October 2017.The released version 5.0.1 addresses some minor bugs and adds useful features, such as updated R language support. All of these features weren’t available in the original 5.0.0 release.

This package manager is also an environment manager, a Python distribution, and a collection of open source packages and contains more than 1000 R and Python Data Science Packages.

**IPYTHON NOTEBOOKS**

IPython is a command shell for interactive computing in multiple programming languages, originally developed for the Python programming language, that offers introspection, rich media, shell syntax, tab completion, and history. IPython provides the following features:

* Interactive shells (terminal and Qt-based).

• A browser-based notebook interface with support for code, text, mathematical expressions, inline plots and other media.

• Support for interactive data visualization and use of GUI toolkits.

• Flexible, embeddable interpreters to load into one's own projects.

• Tools for parallel computing.

IPython is based on an architecture that provides parallel and distributed computing. IPython enables parallel applications to be developed, executed, debugged and monitored interactively. Hence, the I (Interactive) in IPython.[3]This architecture abstracts out parallelism, which enables IPython to support many different styles of parallelism[4]including:

With the release of IPython 4.0, the parallel computing capabilities have been made optional and released under the ipyparallel python package.IPython frequently draw from SciPy stack[5] libraries like NumPy and SciPy, often installed alongside from one of many Scientific Python distributions. IPython provide integration some library of the SciPy stack like matplotlib, like inline graph when in used with the Jupyter notebook. Python libraries can implement IPython specific hooks to customize object Rich object display. SymPy for example implement rendering of Mathematical Expression as rendered LaTeX when used within IPython context.

Other features:

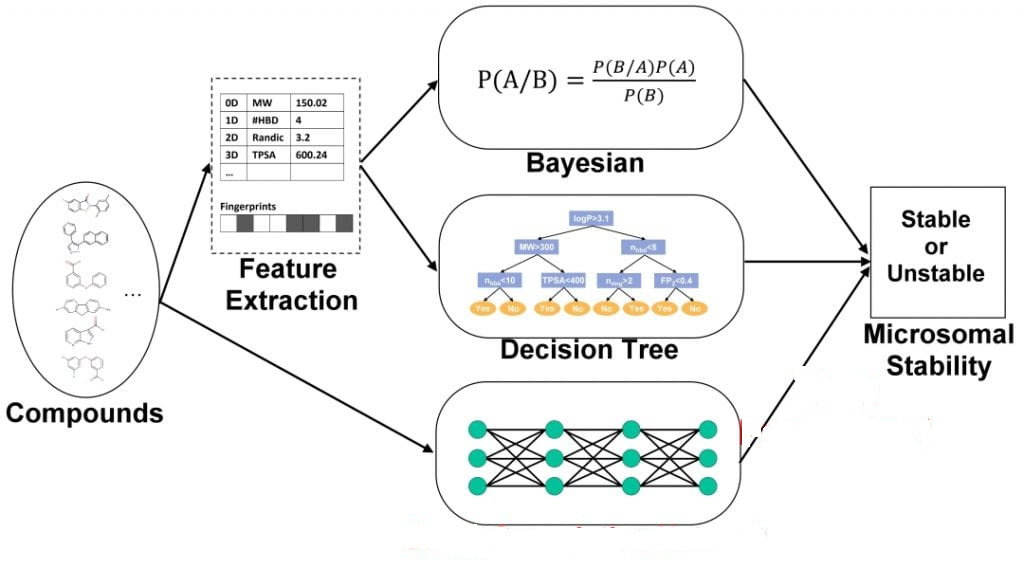
IPython also allows non-blocking interaction with Tkinter, PyGTK, PyQt/PySide and wxPython (the standard Python shell only allows interaction with Tkinter). IPython can interactively manage parallel computing clusters using asynchronous status call-backs and/or MPI. IPython can also be used as a system shell replacement. Its default behaviour is largely similar to Unix shells, but it allows customization and the flexibility of executing code in a live Python environment. Using IPython as a shell replacement is less common and it is now recommended to use Xonsh which provide most of the IPython feature with better shell integrations

**CHAPTER 4**

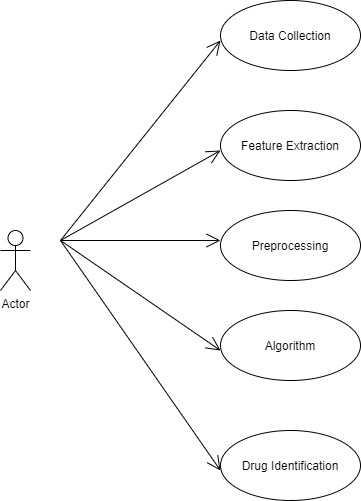
**SYSTEM ARCHITECTURE**

**4.1 ARCHITECTURE OVERVIEW**

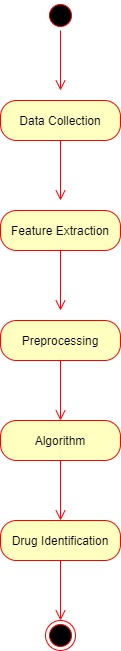
System architecture is the conceptual model that defines the structure, behavior, and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structures and behaviors of the system.

****

**4.2 Usecase Diagram**

****

**4.3 Activity Diagram**

****

**CHAPTER 5**

**SYSTEM MODULE**

**5.1 MODULE**

* Dataset
* SMILES Representation of Molecules
* Model

**5.2 MODULE DESCRIPTION**

**Dataset**

* A dataset containing drug molecules (encoded as SMILES) and their binding affinities. The task is to use this dataset to make a regression model for binding affinity prediction.
* SARS-CoV-2 virus contains proteins responsible for action and replication of the virus.
* The protein functions can be stopped by introducing drug molecules that are capable of blocking the protein. In other words, preparation of a drug involves finding molecules that can effectively bind to the protein i.e have a high binding affinity.
* In this task, you are provided with a dataset of drug molecules and their binding affinity towards SARS-CoronaVirus Main Proteaese(Mpro), one of the proteins in the target virus.
* The data has been generated using Protein-Ligand docking

**SMILES Representation of Molecules**

* SMILES are character strings to represent drug molecules. For example, a carbon atom can be represented as “C”, an oxygen atom can be represented as “O”, double bond by “=”. The molecule Carbon dioxide is represented as “C(=O)=O”.

**Model**

We load trained mol2vec which is trained on Morgan fingerprints with radius = 1 to yield 300 dimensional embeddings. Link to load trained mol2vec

**CHAPTER 6**

**SYSTEM IMPLEMENTATION**

**6.1 SAMPLE CODE**

**from** google.colab **import** drive

drive**.**mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force\_remount=True).

**import** sys

**import** os

**import** requests

**import** subprocess

**import** shutil

**from** logging **import** getLogger, StreamHandler, INFO

logger **=** getLogger(\_\_name\_\_)

logger**.**addHandler(StreamHandler())

logger**.**setLevel(INFO)

**def** install(

chunk\_size**=**4096,

file\_name**=**"Miniconda3-latest-Linux-x86\_64.sh",

url\_base**=**"https://repo.continuum.io/miniconda/",

conda\_path**=**os**.**path**.**expanduser(os**.**path**.**join("~", "miniconda")),

rdkit\_version**=None**,

add\_python\_path**=True**,

force**=False**):

"""install rdkit from miniconda

```

import rdkit\_installer

rdkit\_installer.install()

```

"""

python\_path **=** os**.**path**.**join(

conda\_path,

"lib",

"python{0}.{1}"**.**format(**\***sys**.**version\_info),

"site-packages",

)

**if** add\_python\_path **and** python\_path **notin** sys**.**path:

logger**.**info("add {} to PYTHONPATH"**.**format(python\_path))

sys**.**path**.**append(python\_path)

**if** os**.**path**.**isdir(os**.**path**.**join(python\_path, "rdkit")):

logger**.**info("rdkit is already installed")

**ifnot** force:

**return**

logger**.**info("force re-install")

url **=** url\_base **+** file\_name

python\_version **=** "{0}.{1}.{2}"**.**format(**\***sys**.**version\_info)

logger**.**info("python version: {}"**.**format(python\_version))

**if** os**.**path**.**isdir(conda\_path):

logger**.**warning("remove current miniconda")

shutil**.**rmtree(conda\_path)

**elif** os**.**path**.**isfile(conda\_path):

logger**.**warning("remove {}"**.**format(conda\_path))

os**.**remove(conda\_path)

logger**.**info('fetching installer from {}'**.**format(url))

res **=** requests**.**get(url, stream**=True**)

res**.**raise\_for\_status()

**with** open(file\_name, 'wb') **as** f:

**for** chunk **in** res**.**iter\_content(chunk\_size):

f**.**write(chunk)

logger**.**info('done')

logger**.**info('installing miniconda to {}'**.**format(conda\_path))

subprocess**.**check\_call(["bash", file\_name, "-b", "-p", conda\_path])

logger**.**info('done')

logger**.**info("installing rdkit")

subprocess**.**check\_call([

os**.**path**.**join(conda\_path, "bin", "conda"),

"install",

"--yes",

"-c", "rdkit",

"python=={}"**.**format(python\_version),

"rdkit" **if** rdkit\_version **isNoneelse** "rdkit=={}"**.**format(rdkit\_version)])

logger**.**info("done")

**import** rdkit

logger**.**info("rdkit-{} installation finished!"**.**format(rdkit**.**\_\_version\_\_))

**if** \_\_name\_\_ **==** "\_\_main\_\_":

install()

rdkit is already installed

rdkit is already installed

**!**pip install git+https://github.com/samoturk/mol2vec;

Collecting git+https://github.com/samoturk/mol2vec

Cloning https://github.com/samoturk/mol2vec to /tmp/pip-req-build-8uxng7x9

Running command git clone -q https://github.com/samoturk/mol2vec /tmp/pip-req-build-8uxng7x9

Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from mol2vec==0.1) (1.18.3)

Requirement already satisfied: gensim in /usr/local/lib/python3.6/dist-packages (from mol2vec==0.1) (3.6.0)

Requirement already satisfied: tqdm in /usr/local/lib/python3.6/dist-packages (from mol2vec==0.1) (4.38.0)

Requirement already satisfied: joblib in /usr/local/lib/python3.6/dist-packages (from mol2vec==0.1) (0.14.1)

Requirement already satisfied: pandas in /usr/local/lib/python3.6/dist-packages (from mol2vec==0.1) (1.0.3)

Requirement already satisfied: matplotlib in /usr/local/lib/python3.6/dist-packages (from mol2vec==0.1) (3.2.1)

Requirement already satisfied: IPython in /usr/local/lib/python3.6/dist-packages (from mol2vec==0.1) (5.5.0)

Requirement already satisfied: seaborn in /usr/local/lib/python3.6/dist-packages (from mol2vec==0.1) (0.10.0)

Requirement already satisfied: scipy>=0.18.1 in /usr/local/lib/python3.6/dist-packages (from gensim->mol2vec==0.1) (1.4.1)

Requirement already satisfied: six>=1.5.0 in /usr/local/lib/python3.6/dist-packages (from gensim->mol2vec==0.1) (1.12.0)

Requirement already satisfied: smart-open>=1.2.1 in /usr/local/lib/python3.6/dist-packages (from gensim->mol2vec==0.1) (1.11.1)

Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist-packages (from pandas->mol2vec==0.1) (2018.9)

Requirement already satisfied: python-dateutil>=2.6.1 in /usr/local/lib/python3.6/dist-packages (from pandas->mol2vec==0.1) (2.8.1)

Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib->mol2vec==0.1) (2.4.7)

Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib->mol2vec==0.1) (1.2.0)

Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-packages (from matplotlib->mol2vec==0.1) (0.10.0)

Requirement already satisfied: pygments in /usr/local/lib/python3.6/dist-packages (from IPython->mol2vec==0.1) (2.1.3)

Requirement already satisfied: simplegeneric>0.8 in /usr/local/lib/python3.6/dist-packages (from IPython->mol2vec==0.1) (0.8.1)

Requirement already satisfied: pickleshare in /usr/local/lib/python3.6/dist-packages (from IPython->mol2vec==0.1) (0.7.5)

Requirement already satisfied: pexpect; sys\_platform != "win32" in /usr/local/lib/python3.6/dist-packages (from IPython->mol2vec==0.1) (4.8.0)

Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.6/dist-packages (from IPython->mol2vec==0.1) (4.3.3)

Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.6/dist-packages (from IPython->mol2vec==0.1) (46.1.3)

Requirement already satisfied: decorator in /usr/local/lib/python3.6/dist-packages (from IPython->mol2vec==0.1) (4.4.2)

Requirement already satisfied: prompt-toolkit<2.0.0,>=1.0.4 in /usr/local/lib/python3.6/dist-packages (from IPython->mol2vec==0.1) (1.0.18)

Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages (from smart-open>=1.2.1->gensim->mol2vec==0.1) (2.21.0)

Requirement already satisfied: boto in /usr/local/lib/python3.6/dist-packages (from smart-open>=1.2.1->gensim->mol2vec==0.1) (2.49.0)

Requirement already satisfied: boto3 in /usr/local/lib/python3.6/dist-packages (from smart-open>=1.2.1->gensim->mol2vec==0.1) (1.12.46)

Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.6/dist-packages (from pexpect; sys\_platform != "win32"->IPython->mol2vec==0.1) (0.6.0)

Requirement already satisfied: ipython-genutils in /usr/local/lib/python3.6/dist-packages (from traitlets>=4.2->IPython->mol2vec==0.1) (0.2.0)

Requirement already satisfied: wcwidth in /usr/local/lib/python3.6/dist-packages (from prompt-toolkit<2.0.0,>=1.0.4->IPython->mol2vec==0.1) (0.1.9)

Requirement already satisfied: urllib3<1.25,>=1.21.1 in /usr/local/lib/python3.6/dist-packages (from requests->smart-open>=1.2.1->gensim->mol2vec==0.1) (1.24.3)

Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (from requests->smart-open>=1.2.1->gensim->mol2vec==0.1) (3.0.4)

Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from requests->smart-open>=1.2.1->gensim->mol2vec==0.1) (2020.4.5.1)

Requirement already satisfied: idna<2.9,>=2.5 in /usr/local/lib/python3.6/dist-packages (from requests->smart-open>=1.2.1->gensim->mol2vec==0.1) (2.8)

Requirement already satisfied: botocore<1.16.0,>=1.15.46 in /usr/local/lib/python3.6/dist-packages (from boto3->smart-open>=1.2.1->gensim->mol2vec==0.1) (1.15.46)

Requirement already satisfied: s3transfer<0.4.0,>=0.3.0 in /usr/local/lib/python3.6/dist-packages (from boto3->smart-open>=1.2.1->gensim->mol2vec==0.1) (0.3.3)

Requirement already satisfied: jmespath<1.0.0,>=0.7.1 in /usr/local/lib/python3.6/dist-packages (from boto3->smart-open>=1.2.1->gensim->mol2vec==0.1) (0.9.5)

Requirement already satisfied: docutils<0.16,>=0.10 in /usr/local/lib/python3.6/dist-packages (from botocore<1.16.0,>=1.15.46->boto3->smart-open>=1.2.1->gensim->mol2vec==0.1) (0.15.2)

Building wheels for collected packages: mol2vec

Building wheel for mol2vec (setup.py) ... done

Created wheel for mol2vec: filename=mol2vec-0.1-cp36-none-any.whl size=14026 sha256=5f0c63de6bb59edcde3ae8e0dcae4544d177eef03b9ea0796e0b637ef15de9c5

Stored in directory: /tmp/pip-ephem-wheel-cache-l\_ijpz99/wheels/96/0f/2d/a1092b9677c96453dc244b209544cac61bc8b974cbffb50063

Successfully built mol2vec

Installing collected packages: mol2vec

Successfully installed mol2vec-0.1

*#import the essential libraries*

**%matplotlib** inline

**import** numpy **as** np

**import** pandas **as** pd

**import** matplotlib.pyplot **as** plt

train **=** pd**.**read\_csv('/content/drive/My Drive/covid/train.csv')

train**.**head()

|  | **SMILES sequence** | **Binding Affinity** |
| --- | --- | --- |
| **0** | CCNC(C)C(NC)c1ccccc1 | -18.0861 |
| **1** | CONC(=O)c1cncnc1 | -17.5783 |
| **2** | CCNC1CCCN(Cc2ccsc2)C1 | -20.3645 |
| **3** | CC(NC(=O)CSCCN)c1ccccc1 | -19.3144 |
| **4** | CCC(CS)CN(C)c1ccccc1 | -15.8451 |

train\_aff **=** train['Binding Affinity']

train**.**drop(columns**=**'Binding Affinity',inplace**=True**)

train**.**head()

|  |  |
| --- | --- |
| **0** | CCNC(C)C(NC)c1ccccc1 |
| **1** | CONC(=O)c1cncnc1 |
| **2** | CCNC1CCCN(Cc2ccsc2)C1 |
| **3** | CC(NC(=O)CSCCN)c1ccccc1 |
| **4** | CCC(CS)CN(C)c1ccccc1 |

In [15]:

*#It stores the binding affinity*

train\_aff**.**head()

Out[15]:

0 -18.0861

1 -17.5783

2 -20.3645

3 -19.3144

4 -15.8451

Name: Binding Affinity, dtype: float64

In [0]:

test **=** pd**.**read\_csv('/content/drive/My Drive/covid/test.csv')

test\_name **=** pd**.**read\_csv('/content/drive/My Drive/covid/test.csv')*## Creating this just as a copy of test data to create the csv using this later*

In [17]:

test**.**head()

Out[17]:

|  | **SMILES sequence** | **Binding Affinity** |
| --- | --- | --- |
| **0** | Cc1ccc(C2CNCCN2C)cc1 | NaN |
| **1** | CCOC(CO)c1ccccc1 | NaN |
| **2** | CC(=O)Nc1cnn(C)n1 | NaN |
| **3** | CCC(C)NCc1ncccn1 | NaN |
| **4** | CC(C)=C1CC(N)C1 | NaN |

test\_name**.**head()

|  | **SMILES sequence** | **Binding Affinity** |
| --- | --- | --- |
| **0** | Cc1ccc(C2CNCCN2C)cc1 | NaN |
| **1** | CCOC(CO)c1ccccc1 | NaN |
| **2** | CC(=O)Nc1cnn(C)n1 | NaN |
| **3** | CCC(C)NCc1ncccn1 | NaN |
| **4** | CC(C)=C1CC(N)C1 | NaN |

In [0]:

test**.**drop(columns**=**'Binding Affinity',inplace**=True**)

test\_name**.**drop(columns**=**'Binding Affinity',inplace**=True**)

In [20]:

test**.**head()

Out[20]:

|  | **SMILES sequence** |
| --- | --- |
| **0** | Cc1ccc(C2CNCCN2C)cc1 |
| **1** | CCOC(CO)c1ccccc1 |
| **2** | CC(=O)Nc1cnn(C)n1 |
| **3** | CCC(C)NCc1ncccn1 |
| **4** | CC(C)=C1CC(N)C1 |

In [21]:

test\_name**.**head()

Out[21]:

|  | **SMILES sequence** |
| --- | --- |
| **0** | Cc1ccc(C2CNCCN2C)cc1 |
| **1** | CCOC(CO)c1ccccc1 |
| **2** | CC(=O)Nc1cnn(C)n1 |
| **3** | CCC(C)NCc1ncccn1 |
| **4** | CC(C)=C1CC(N)C1 |

In [0]:

*##This is the sample of submission (Submission must be in this format)*

sample **=** pd**.**read\_csv('/content/drive/My Drive/covid/sample\_submission.csv')

In [24]:

sample**.**head()

Out[24]:

|  | **SMILES sequence** | **Binding Affinity** |
| --- | --- | --- |
| **0** | Cc1ccc(C2CNCCN2C)cc1 | -10.0 |
| **1** | CCOC(CO)c1ccccc1 | -10.0 |
| **2** | CC(=O)Nc1cnn(C)n1 | -10.0 |
| **3** | CCC(C)NCc1ncccn1 | -10.0 |
| **4** | CC(C)=C1CC(N)C1 | -10.0 |

In [0]:

**from** rdkit **import** Chem

In [0]:

train['mol'] **=** train['SMILES sequence']**.**apply(**lambda** x: Chem**.**MolFromSmiles(x))

test['mol'] **=** test['SMILES sequence']**.**apply(**lambda** x: Chem**.**MolFromSmiles(x))

In [27]:

*#Now let's see what we've got*

print(type(train['mol'][0]))

<class 'rdkit.Chem.rdchem.Mol'>

In [28]:

print(type(test['mol'][0]))

<class 'rdkit.Chem.rdchem.Mol'>

In [32]:

**from** rdkit.Chem **import** Draw

mols **=** train['mol'][:2]

*#MolsToGridImage allows to paint a number of molecules at a time*

Draw**.**MolsToGridImage(mols, molsPerRow**=**5, useSVG**=True**, legends**=**list(train['SMILES sequence'][:20]**.**values))

Out[32]:

'<?xml version=\'1.0\' encoding=\'iso-8859-1\'?>\n<svg version=\'1.1\' baseProfile=\'full\'\n xmlns=\'http://www.w3.org/2000/svg\'\n xmlns:rdkit=\'http://www.rdkit.org/xml\'\n xmlns:xlink=\'http://www.w3.org/1999/xlink\'\n xml:space=\'preserve\'\nwidth=\'1000px\' height=\'200px\' viewBox=\'0 0 1000 200\'>\n<!-- END OF HEADER -->\n<rect style=\'opacity:1.0;fill:#FFFFFF;stroke:none\' width=\'1000\' height=\'200\' x=\'0\' y=\'0\'></rect>\n<rect style=\'opacity:1.0;fill:#FFFFFF;stroke:none\' width=\'1000\' height=\'200\' x=\'0\' y=\'0\'></rect>\n<rect style=\'opacity:1.0;fill:#FFFFFF;stroke:none\' width=\'1000\' height=\'200\' x=\'0\' y=\'0\'></rect>\n<path class=\'bond-0\' d=\'M 41.2474,24.6306 L 43.0908,58.4829\' style=\'fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-1\' d=\'M 43.0908,58.4829 L 55.7613,64.9063\' style=\'fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-1\' d=\'M 55.7613,64.9063 L 68.4319,71.3298\' style=\'fill:none;fill-rule:evenodd;stroke:#0000FF;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-2\' d=\'M 73.637,79.463 L 74.4049,93.5639\' style=\'fill:none;fill-rule:evenodd;stroke:#0000FF;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-2\' d=\'M 74.4049,93.5639 L 75.1727,107.665\' style=\'fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-3\' d=\'M 75.1727,107.665 L 46.7774,126.187\' style=\'fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-4\' d=\'M 75.1727,107.665 L 105.411,122.995\' style=\'fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-5\' d=\'M 105.411,122.995 L 106.179,137.096\' style=\'fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-5\' d=\'M 106.179,137.096 L 106.947,151.196\' style=\'fill:none;fill-rule:evenodd;stroke:#0000FF;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-7\' d=\'M 105.411,122.995 L 133.807,104.472\' style=\'fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-6\' d=\'M 102.357,160.042 L 90.6083,167.705\' style=\'fill:none;fill-rule:evenodd;stroke:#0000FF;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-6\' d=\'M 90.6083,167.705 L 78.8594,175.369\' style=\'fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-8\' d=\'M 133.807,104.472 L 131.963,70.6199\' style=\'fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-8\' d=\'M 140.3,99.0256 L 139.01,75.329\' style=\'fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-13\' d=\'M 133.807,104.472 L 164.045,119.802\' style=\'fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-9\' d=\'M 131.963,70.6199 L 160.358,52.0974\' style=\'fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-10\' d=\'M 160.358,52.0974 L 190.597,67.4271\' style=\'fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-10\' d=\'M 161.828,60.4446 L 182.995,71.1754\' style=\'fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-11\' d=\'M 190.597,67.4271 L 192.44,101.279\' style=\'fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-12\' d=\'M 192.44,101.279 L 164.045,119.802\' style=\'fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-12\' d=\'M 184.477,98.3787 L 164.6,111.344\' style=\'fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<text dominant-baseline="central" text-anchor="start" x=\'69.562\' y=\'75.5077\' style=\'font-size:11px;font-style:normal;font-weight:normal;fill-opacity:1;stroke:none;font-family:sans-serif;fill:#0000FF\' ><tspan>NH</tspan></text>\n<text dominant-baseline="central" text-anchor="start" x=\'103.487\' y=\'158.542\' style=\'font-size:11px;font-style:normal;font-weight:normal;fill-opacity:1;stroke:none;font-family:sans-serif;fill:#0000FF\' ><tspan>NH</tspan></text>\n<text dominant-baseline="central" text-anchor="middle" x=\'100\' y=\'189.8\' style=\'font-size:12px;font-style:normal;font-weight:normal;fill-opacity:1;stroke:none;font-family:sans-serif;fill:#000000\' ><tspan>CCNC(C)C(NC)c1ccccc1</tspan></text>\n<path class=\'bond-0\' d=\'M 208.956,73.4832 L 218.5,82.7168\' style=\'fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-0\' d=\'M 218.5,82.7168 L 228.045,91.9505\' style=\'fill:none;fill-rule:evenodd;stroke:#FF0000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-1\' d=\'M 238.599,95.548 L 249.811,92.3442\' style=\'fill:none;fill-rule:evenodd;stroke:#FF0000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-1\' d=\'M 249.811,92.3442 L 261.022,89.1404\' style=\'fill:none;fill-rule:evenodd;stroke:#0000FF;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-2\' d=\'M 270.817,92.479 L 280.551,101.896\' style=\'fill:none;fill-rule:evenodd;stroke:#0000FF;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-2\' d=\'M 280.551,101.896 L 290.285,111.314\' style=\'fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-3\' d=\'M 286.997,110.491 L 283.588,124.109\' style=\'fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-3\' d=\'M 283.588,124.109 L 280.179,137.728\' style=\'fill:none;fill-rule:evenodd;stroke:#FF0000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-3\' d=\'M 293.574,112.137 L 290.166,125.756\' style=\'fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-3\' d=\'M 290.166,125.756 L 286.757,139.374\' style=\'fill:none;fill-rule:evenodd;stroke:#FF0000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-4\' d=\'M 290.285,111.314 L 322.883,101.999\' style=\'fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-5\' d=\'M 322.883,101.999 L 331.115,69.1108\' style=\'fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-5\' d=\'M 330.695,98.7118 L 336.458,75.6903\' style=\'fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-10\' d=\'M 322.883,101.999 L 347.249,125.571\' style=\'fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-6\' d=\'M 331.115,69.1108 L 344.965,65.153\' style=\'fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-6\' d=\'M 344.965,65.153 L 358.815,61.1952\' style=\'fill:none;fill-rule:evenodd;stroke:#0000FF;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-7\' d=\'M 368.61,64.5338 L 378.344,73.9511\' style=\'fill:none;fill-rule:evenodd;stroke:#0000FF;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-7\' d=\'M 378.344,73.9511 L 388.078,83.3685\' style=\'fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-7\' d=\'M 366.816,72.2322 L 373.629,78.8243\' style=\'fill:none;fill-rule:evenodd;stroke:#0000FF;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-7\' d=\'M 373.629,78.8243 L 380.443,85.4165\' style=\'fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-8\' d=\'M 388.078,83.3685 L 384.67,96.9872\' style=\'fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-8\' d=\'M 384.67,96.9872 L 381.261,110.606\' style=\'fill:none;fill-rule:evenodd;stroke:#0000FF;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-9\' d=\'M 374.949,117.656 L 361.099,121.614\' style=\'fill:none;fill-rule:evenodd;stroke:#0000FF;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-9\' d=\'M 361.099,121.614 L 347.249,125.571\' style=\'fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-9\' d=\'M 368.931,112.324 L 359.236,115.094\' style=\'fill:none;fill-rule:evenodd;stroke:#0000FF;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<path class=\'bond-9\' d=\'M 359.236,115.094 L 349.541,117.865\' style=\'fill:none;fill-rule:evenodd;stroke:#000000;stroke-width:2px;stroke-linecap:butt;stroke-linejoin:miter;stroke-opacity:1\' />\n<text dominant-baseline="central" text-anchor="middle" x=\'233.322\' y=\'98.7511\' style=\'font-size:11px;font-style:normal;font-weight:normal;fill-opacity:1;stroke:none;font-family:sans-serif;fill:#FF0000\' ><tspan>O</tspan></text>\n<text dominant-baseline="central" text-anchor="middle" x=\'265.92\' y=\'89.4361\' style=\'font-size:11px;font-style:normal;font-weight:normal;fill-opacity:1;stroke:none;font-family:sans-serif;fill:#0000FF\' ><tspan>N</tspan></text>\n<text dominant-baseline="central" text-anchor="start" x=\'262.152\' y=\'78.1352\' style=\'font-size:11px;font-style:normal;font-weight:normal;fill-opacity:1;stroke:none;font-family:sans-serif;fill:#0000FF\' ><tspan>H</tspan></text>\n<text dominant-baseline="central" text-anchor="start" x=\'277.907\' y=\'145.897\' style=\'font-size:11px;font-style:normal;font-weight:normal;fill-opacity:1;stroke:none;font-family:sans-serif;fill:#FF0000\' ><tspan>O</tspan></text>\n<text dominant-baseline="central" text-anchor="middle" x=\'363.712\' y=\'61.4908\' style=\'font-size:11px;font-style:normal;font-weight:normal;fill-opacity:1;stroke:none;font-family:sans-serif;fill:#0000FF\' ><tspan>N</tspan></text>\n<text dominant-baseline="central" text-anchor="start" x=\'376.079\' y=\'117.952\' style=\'font-size:11px;font-style:normal;font-weight:normal;fill-opacity:1;stroke:none;font-family:sans-serif;fill:#0000FF\' ><tspan>N</tspan></text>\n<text dominant-baseline="central" text-anchor="middle" x=\'300\' y=\'189.8\' style=\'font-size:12px;font-style:normal;font-weight:normal;fill-opacity:1;stroke:none;font-family:sans-serif;fill:#000000\' ><tspan>CONC(=O)c1cncnc1</tspan></text>\n</svg>\n'

In [33]:

train**.**head()

Out[33]:

|  | **SMILES sequence** | **mol** |
| --- | --- | --- |
| **0** | CCNC(C)C(NC)c1ccccc1 | <rdkit.Chem.rdchem.Mol object at 0x7f04d05cfcb0> |
| **1** | CONC(=O)c1cncnc1 | <rdkit.Chem.rdchem.Mol object at 0x7f04d05cfd00> |
| **2** | CCNC1CCCN(Cc2ccsc2)C1 | <rdkit.Chem.rdchem.Mol object at 0x7f04d05cfd50> |
| **3** | CC(NC(=O)CSCCN)c1ccccc1 | <rdkit.Chem.rdchem.Mol object at 0x7f04d05cfda0> |
| **4** | CCC(CS)CN(C)c1ccccc1 | <rdkit.Chem.rdchem.Mol object at 0x7f04d05cfdf0> |

In [34]:

test**.**head()

Out[34]:

|  | **SMILES sequence** | **mol** |
| --- | --- | --- |
| **0** | Cc1ccc(C2CNCCN2C)cc1 | <rdkit.Chem.rdchem.Mol object at 0x7f04cc32cc10> |
| **1** | CCOC(CO)c1ccccc1 | <rdkit.Chem.rdchem.Mol object at 0x7f04cc32cee0> |
| **2** | CC(=O)Nc1cnn(C)n1 | <rdkit.Chem.rdchem.Mol object at 0x7f04d0fd7800> |
| **3** | CCC(C)NCc1ncccn1 | <rdkit.Chem.rdchem.Mol object at 0x7f04d0fd7710> |
| **4** | CC(C)=C1CC(N)C1 | <rdkit.Chem.rdchem.Mol object at 0x7f04cc330030> |

In [35]:

*#Now we'll load a pre-trained mol2vec model. It's trained with radius=1 for Morgan fingerprints to yield 300 dimensional embeddings.*

*#Loading pre-trained model via word2vec*

**from** gensim.models **import** word2vec

model **=** word2vec**.**Word2Vec**.**load('/content/drive/My Drive/model\_300dim.pkl')

/usr/local/lib/python3.6/dist-packages/smart\_open/smart\_open\_lib.py:253: UserWarning: This function is deprecated, use smart\_open.open instead. See the migration notes for details: https://github.com/RaRe-Technologies/smart\_open/blob/master/README.rst#migrating-to-the-new-open-function

'See the migration notes for details: %s' % \_MIGRATION\_NOTES\_URL

In [0]:

**from** mol2vec.features **import** mol2alt\_sentence, mol2sentence, MolSentence, DfVec, sentences2vec

**from** gensim.models **import** word2vec

In [37]:

print('Molecular sentence:', mol2alt\_sentence(train['mol'][1], radius**=**1))

print('\nMolSentence object:', MolSentence(mol2alt\_sentence(train['mol'][1], radius**=**1)))

print('\nDfVec object:',DfVec(sentences2vec(MolSentence(mol2alt\_sentence(train['mol'][1], radius**=**1)), model, unseen**=**'UNK')))

Molecular sentence: ['2246728737', '3975275337', '864674487', '903112553', '847961216', '2204949651', '2246699815', '1054767590', '864942730', '1510328189', '3217380708', '2994748777', '3218693969', '3777168895', '2041434490', '3118255683', '3218693969', '725322217', '2041434490', '3118255683', '3218693969', '3777168895']

MolSentence object: MolSentence with 22 words

DfVec object: (22, 300) dimensional vector

In [38]:

print('Molecular sentence:', mol2alt\_sentence(test['mol'][1], radius**=**1))

print('\nMolSentence object:', MolSentence(mol2alt\_sentence(test['mol'][1], radius**=**1)))

print('\nDfVec object:',DfVec(sentences2vec(MolSentence(mol2alt\_sentence(test['mol'][1], radius**=**1)), model, unseen**=**'UNK')))

Molecular sentence: ['2246728737', '3542456614', '2245384272', '3994088662', '864674487', '2814583100', '2245273601', '602889953', '2245384272', '4022716898', '864662311', '1535166686', '3217380708', '3579962709', '3218693969', '951226070', '3218693969', '98513984', '3218693969', '98513984', '3218693969', '98513984', '3218693969', '951226070']

MolSentence object: MolSentence with 24 words

DfVec object: (24, 300) dimensional vector

In [0]:

*#Constructing sentences*

train['sentence'] **=** train**.**apply(**lambda** x: MolSentence(mol2alt\_sentence(x['mol'], 1)), axis**=**1)

In [0]:

test['sentence'] **=** test**.**apply(**lambda** x: MolSentence(mol2alt\_sentence(x['mol'], 1)), axis**=**1)

In [41]:

*#Extracting embeddings to a numpy.array*

*#Note that we always should mark unseen='UNK' in sentence2vec() so that model is taught how to handle unknown substructures*

train['mol2vec'] **=** [DfVec(x) **for** x **in** sentences2vec(train['sentence'], model, unseen**=**'UNK')]

X **=** np**.**array([x**.**vec **for** x **in** train['mol2vec']])

y **=** train\_aff**.**values

X**.**shape

Out[41]:

(9000, 300)

In [42]:

test['mol2vec'] **=** [DfVec(x) **for** x **in** sentences2vec(test['sentence'], model, unseen**=**'UNK')]

test\_value**=** np**.**array([x**.**vec **for** x **in** test['mol2vec']])

test\_value**.**shape

Out[42]:

(2500, 300)

In [0]:

**from** sklearn.model\_selection **import** train\_test\_split

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, y, test\_size**=.**1, random\_state**=**1)

In [0]:

**from** sklearn **import** svm

Testing with SVM

In [0]:

clf **=** svm**.**SVR(C**=**100)

In [46]:

clf**.**fit(X\_train, y\_train)

Out[46]:

SVR(C=110, cache\_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',

kernel='rbf', max\_iter=-1, shrinking=True, tol=0.001, verbose=False)

In [0]:

y\_pred**=**clf**.**predict(X\_test)

In [50]:

**from** sklearn.metrics **import** mean\_squared\_error

**from** math **import** sqrt

rmse **=** sqrt(mean\_squared\_error(y\_test, y\_pred))

print(rmse)

2.328732546851693

In [0]:

**from** sklearn.metrics **import** mean\_absolute\_error

In [52]:

mean\_absolute\_error(y\_test, y\_pred)

Out[52]:

1.6724466865753191

Testing with Ridge

In [0]:

**from** sklearn.linear\_model **import** RidgeCV

ridge **=** RidgeCV()

In [57]:

ridge**.**fit(X\_train, y\_train)

Out[57]:

RidgeCV(alphas=array([ 0.1, 1. , 10. ]), cv=None, fit\_intercept=True,

gcv\_mode=None, normalize=False, scoring=None, store\_cv\_values=False)

In [0]:

y\_pred\_r **=** ridge**.**predict(X\_test)

In [59]:

rmse **=** sqrt(mean\_squared\_error(y\_test, y\_pred\_r))

rmse

Out[59]:

2.37664859146776

In [60]:

print(mean\_absolute\_error(y\_test, y\_pred\_r))

1.7649498076453982

We have found out that of all Svm.SVR works best of sll the algorithm with 'C' value of 100

In [0]:

clf1 **=** svm**.**SVR(C**=**100)

In [63]:

clf1**.**fit(X, y)

Out[63]:

SVR(C=100, cache\_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',

kernel='rbf', max\_iter=-1, shrinking=True, tol=0.001, verbose=False)

In [0]:

y\_pred\_svm**=**clf**.**predict(test\_value)

In [65]:

final\_aff**=**y\_pred\_svm**.**tolist()

type(final\_aff)

Out[65]:

list

In [0]:

final\_test**=** test\_name**.**values**.**tolist()

In [67]:

type(final\_test)

Out[67]:

list

In [71]:

print(final\_test)

'CCNc1ccc(CN(C)OCC)cn1', 'CCC(=Cc1cnccc1C)CN', 'O=C(CSC1CCCCC1)Nc1ccncc1', 'CCCOC(=O)c1cncnc1', 'Cc1ccsc1C1CC(O)CN1', 'CCNC(=NN)c1cnccc1C', 'NCC(Cc1ccccn1)C1CCCCC1', 'Cc1ccncc1C(N)C(C)C(C)C', 'Cc1ccncc1CNC(=O)OC(C)C', 'CC(SCCOCc1ccccc1)C(=O)O', 'CC(N)CC(=O)NCc1ccccc1', 'CC(C)NCC(N)CCc1ccccc1', 'CC(N)(CO)c1ncccn1', 'O=C(NNc1ccccn1)NC1CCCCC1', 'Cc1ccncc1C(N)CSCC(C)C', 'CCC(CC)c1ncc(N)cn1', 'C=CC(Oc1cnccc1C)C(=O)OC', 'Cc1ccnc(CON)c1', 'CCSCC(C)NCc1cc(C)ccn1', 'CCCN(CC)c1cccc(NCC)n1', 'CCC(CC)CC(=O)c1cnccc1C', 'CN1CCCc2cc(OCCC#N)ccc21', 'C(=Cc1ccccc1)CNc1cccnc1', 'N#Cc1cccc(C2CCOCCN2)n1', 'CC=CC(=O)C(N)Cc1ccccc1', 'NC12CCCC1CCC2', 'Cc1ccnc(CNC(C)(C)C)c1', 'O=C1CCCCC1(O)Cc1ccsc1', 'CC(=O)NCC1CCCCN1Cc1ccsc1', 'C=COc1cnccc1C', 'CCC(CO)N(C)C(=O)c1ccccc1', 'C=CC(C)(C)CNc1ncccc1C#N', 'CCC(CC)Nc1ccccc1C(=O)OC', 'CN1CCC(C=O)c2ccccc21', 'O=C(O)CNc1ccncn1', 'O=C(O)CCCCC=Cc1ccccc1', 'CN(N)C(=O)c1ccc(C#N)cn1', 'CNc1cncc(N2CCCOCC2)c1', 'CCC(O)C(C)Nc1cc(C)ccn1', 'N#CCNC(=O)CCCc1ccccc1', 'CC#CCC(CSc1ccccc1)NN', 'CC(C)CC(C)Nc1ccc(C#N)cn1', 'CC(O)CC(C)c1cncnc1', 'CCCN(C)Cc1cncnc1', 'Cc1ccncc1C(CC(C)C)NN', 'CN(C)Cc1ncc(N)c(O)n1', 'NC1CC2CC(=O)CC2C1', 'C=CC(CO)NC(C)c1cnccc1C', 'Nc1cnncn1', 'CC(C)CCN(N)c1ccccc1', 'COCC(C)CC(N)c1cnccc1C', 'C(=Cc1ccccn1)c1ccccc1', 'CN(CC(=O)O)c1ccc(C#N)cn1', 'C=CCCc1cc(C)ccn1', 'CONCCNc1ncccc1C#N', 'CC1NC(=O)N(CC(N)=O)C1=O', 'C#CC(=O)c1ccncn1', 'COCCOCCOc1cnccc1C', 'CNC(C)(C)CNCc1cnccc1C', 'CC(C)N(C)c1ccc(C#N)cn1', 'CC(O)C=CC=Cc1ccccc1', 'CCC(N)CNc1ccncn1', 'N#CCC(=O)Nc1ccc(C#N)cn1', 'CCNC(=N)c1cc(C)ccn1', 'CN(C)CCNc1ccncc1C#N', 'CC(CCO)SCC(O)COCc1ccccc1', 'CCNc1ncccc1CSC(C)C', 'C=C(C)CN(C)c1cccc(NCC)n1', 'CC(C)(O)CNc1ccncc1C#N', 'C=C1CON(c2ccnc(C#N)c2)C1', 'O=C(Cc1ccsc1)Nc1cccnc1', 'C=C1CC(N)C12CCC2', 'COC(C)(C)CNc1cc(C)ccn1', 'CCNC(C)(C)C(=O)c1cnccc1C', 'Cc1ccncc1C#CC(C)N', 'CCNc1sc2c(c1C#N)CCC2', 'NC1C(=O)SC12CCC2', 'CCC=CCNc1ncccn1', 'CC(C)=CCCNc1ccc(C#N)cn1', 'CCC(CC)NCc1ccc(C#N)cn1', 'CC=CC(=O)c1cnccc1C', 'CC(CC1CCCCC1)NCc1ccncc1', 'O=C(CCc1ccccc1)c1cccnc1', 'CCNc1ccc(CN(C)OC)cn1', 'CC(C)CCCNc1ccc(C#N)cn1', 'CC(C)(N)COCc1cncc(C#N)c1', 'Cc1ccncc1C(O)C(C)CCN', 'CC(C)(C)C(O)c1ncccn1', 'C=CCCC(N)c1cc(C)ccn1', 'Cc1ccncc1C(C)(N)CCN(C)C', 'CCC(CC(=O)OC)Nc1ccccc1', 'CC(C)CCSc1ccncc1C#N', 'C=CC=CCCC(O)c1cc(C)ccn1', 'O=C(CC1CCCCC1)NC1CCCOC1', 'CCNC(C#N)c1cnccc1C', 'CCNc1ccc(C(=O)OC)cn1', 'CCNS(=O)(=O)CC1CCCC1', 'N#CC1CCC(CCN)C1', 'N#Cc1cc(N2CCNNCC2)ccn1', 'COC(=O)c1cccc(C(N)N=O)c1', 'NC(Cc1ccncn1)C(=O)O', 'Cc1ccsc1C1(O)CCC2CCCC21', 'CC(C)SCC(O)(CN)c1ccccc1', 'CN1CCC(OC2CCC(=O)CC2)CC1', 'Cc1cnn(CC(N)=O)c(=O)c1', 'CCSc1ncc(N)cn1', 'Cc1cscc1C1NCCCCC1O', 'O=C(CSc1ccccn1)NC1CCCCC1', 'COCCNc1ccncc1C#N', 'CC(CC(=O)O)c1ncccn1', 'CCNC1CCCOC1C#N', 'CCC(N)C(CC)(CCO)c1ccccc1', 'CC12CC(CO1)N(CC(N)=O)C2', 'CC(C)NC(=O)c1ccncn1', 'CC(CN)C(=O)NCCOCc1ccccc1', 'O=C(O)C=Cc1ncccn1', 'CC(C#N)Cc1cccc2c1CCCN2C', 'Cc1ccnc(CCON)c1', 'CN(CC1COCCN1)c1ccccc1C#N', 'NN1CC(=O)NC1=O', 'CCNC(CCSC)c1cnccc1C', 'CCC(CN)C(=O)Nc1cnccc1C', 'CNCC(O)c1cncnc1', 'CC(=Cc1ccccc1)CCS', 'CC(C)C(C)OCc1cncc(C#N)c1', 'Cc1ccncc1CC(N)=O', 'NC1CCC2CC1CN2', 'N#CN(Cc1ccccc1)c1ccccn1', 'CCCC(=O)c1ccncn1', 'C=CCN(CCO)c1ccncc1C#N', 'COC(=O)c1ccccc1CCC(=O)O', 'O=C(O)CCSc1ccncn1', 'Cc1ccccc1OCCN1CCN(C)CC1', 'CCN(CC(=O)NC)c1ncccc1C#N', 'CCCCC(CC)COc1ccccc1', 'CCNc1cccc(NC(C)(CC)CO)n1', 'Cc1ccncc1C(N)CCN(C)C', 'CCC(CN)Nc1ccncn1', 'CCC(C)(C)C(NC)c1cnccc1C', 'COCCCN(C)c1ccc(C#N)cn1', 'CCNS(=O)(=O)Cc1cccnc1', 'COC(=O)c1ccccc1CCN=C=O', 'CCCSCc1cncc(C#N)c1', 'C=CCC(=O)Cc1ccccc1', 'C=CCN(CC)c1cc(C)ccn1', 'CCNc1ncccc1CN(C)CCO', 'c1cncc(CCNC2CCCCC2)c1', 'COC(C)(C)CC=C1CC(N)C1', 'CN1CCCc2ccccc21', 'CC(Cc1cccnc1)NC1CCCCC1', 'CCC(C)(C)c1ncc(N)cn1', 'C#CCC(CC)NCc1cnccc1C', 'Nc1cncnc1CO', 'Cc1ccncc1C(C)C(C)C(=O)O', 'COC(=O)C(c1cnccc1C)C(C)C', 'CNCCOc1ncccn1', 'Cc1ccnc(CNCCOC(C)C)c1', 'CC(C)(C#N)c1ncccn1', 'OCCC(c1ccccc1)c1ccncc1', 'NS(=O)(=O)c1ccccc1C=CC=O', 'Nc1ccncc1', 'NC1=CCNC(=O)N1', 'CCNc1cc(N(C)CC(N)=O)ccn1', 'CCN(CC)CCN(CC)Cc1ccccc1', 'CCCOCC(=O)c1cnccc1C', 'S=C(NCc1cccnc1)NC1CCCCC1', 'C#CC(C)Oc1ccnc(NCC)c1', 'COCSc1ccc(C#N)cn1', 'Cc1ccncc1CCC(=O)O', 'COC(=O)COC(N)COCc1ccccc1', 'C=CCCC=C1CCCC1N', 'CNCCCOc1cncnc1', 'NC(=O)c1ncc(N)cn1', 'CCNc1cccc(NCC(C)(O)CC)n1', 'Cc1ccnc(C(C)NCCC#N)c1', 'OC1(Cc2ccsc2)CCNC1', 'CCOCC(C)NC(=O)OCc1ccccc1', 'CCNc1cc(N(CC)C(C)C)ccn1', 'Cc1ccnc(NCCC(=O)N(C)C)c1', 'CC1CCN(Cc2ccsc2)CC1', 'O=C(COc1cccnc1)NC1CCCCC1', 'CCCCCCCOc1ccc(C#N)cn1', 'CCC(CC)=C1CCCCC1N', 'CCS(=O)(=O)CCNCCc1ccccc1', 'C#CCOCSc1ncccn1', 'CCCC(O)CNc1ncccc1C#N', 'C=CCNCC(=O)Nc1cnccc1C', 'CCCC(C)OCc1cncc(C#N)c1', 'Cc1ccnc(CNC(CO)C(C)C)c1', 'CCN(C)c1ccncc1C#N', 'COc1nc(SC)ncc1N', 'NC1CCCCS1(=O)=O', 'CN(CCO)C(C(N)=O)c1ccccc1', 'CCOc1ncnc(SC)c1N', 'CCOCC(N)c1cc(C)ccn1', 'CC(O)CN(C)c1ccncn1', 'CCNC1(C#N)CC2C=CC1C2', 'N#CCC(c1ccccc1)c1ccncc1', 'O=C(NCCc1ccncc1)NC1CCNC1', 'NS(=O)(=O)c1ccccc1NCCO', 'CCCNC(C)CCc1cnccc1C', 'CCCNC1CCCN(Cc2ccsc2)C1', 'NC1COCCOC1', 'CCCCC(O)c1cnccc1C', 'CN1CCCc2cc(C(=O)NN)ccc21', 'CNC(=O)CNCc1cncc(C#N)c1', 'C=C(C)CNCCCN(C)Cc1ccccc1', 'N#CC=Cc1ccncn1', 'CCN(CCN)c1ccncn1', 'CCNC(C)CNc1ncccc1C#N', 'NN1CCC2(CC1)CC2', 'Cc1ccnc(OCCCC(C)CN)c1', 'CON(c1ccncc1C#N)C(C)C', 'CNCCCCc1ccc2c(c1)CCCN2C', 'CCNC1(C#N)CC2CCC(C1)S2', 'COC(=O)c1ccc(C(C)SC)cc1', 'N#Cc1ncccc1N1CCC(=O)C1', 'CNCCSCc1cncc(C#N)c1', 'NC1=NC(=O)C(=CC(=O)O)S1', 'C=C(CC)CC(C)C(O)c1ccccc1', 'N#CCC(CN)c1cncnc1']

In [0]:

*# We join the sequence name and its predicted affinity and convert it into a dataframe*

df **=** pd**.**DataFrame(list(zip(fi, final\_aff )), columns **=**['SMILES sequence', 'Binding Affinity'])

In [74]:

df**.**head()

Out[74]:

|  | **SMILES sequence** | **Binding Affinity** |
| --- | --- | --- |
| **0** | Cc1ccc(C2CNCCN2C)cc1 | -21.682052 |
| **1** | CCOC(CO)c1ccccc1 | -14.222270 |
| **2** | CC(=O)Nc1cnn(C)n1 | -23.436068 |
| **3** | CCC(C)NCc1ncccn1 | -19.540014 |
| **4** | CC(C)=C1CC(N)C1 | -19.953755 |

In [0]:

*#Convert the dataframe into a csv named 'sub1.csv'*

file\_name\_csv**=** 'sub1.csv'

df**.**to\_csv(file\_name\_csv,index**=False**)

**CHAPTER 7**

**TESTING**

**7.1 TESTING TECHNIQUES**

Testing is a process of executing a program with the intent of finding an error. A good test case is one that has a high probability of finding an as-yet –undiscovered error. A successful test is one that uncovers an as-yet- undiscovered error. System testing is the stage of implementation, which is aimed at ensuring that the system works accurately and efficiently as expected before live operation commences. It verifies that the whole set of programs hang together. System testing requires a test consists of several key activities and steps for run program, string, system and is important in adopting a successful new system. This is the last chance to detect and correct errors before the system is installed for user acceptance testing.

The software testing process commences once the program is created and the documentation and related data structures are designed. Software testing is essential for correcting errors. Otherwise the program or the project is not said to be complete. Software testing is the critical element of software quality assurance and represents the ultimate the review of specification design and coding. Testing is the process of executing the program with the intent of finding the error. A good test case design is one that as a probability of finding an yet undiscovered error. A successful test is one that uncovers an yet undiscovered error. Any engineering product can be tested in one of the two ways:

**WHITE BOX TESTING**

This testing is also called as Glass box testing. In this testing, by knowing the specific functions that a product has been design to perform test can be conducted that demonstrate each function is fully operational at the same time searching for errors in each function. It is a test case design method that uses the control structure of the procedural design to derive test cases. Basis path testing is a white box testing.

Basis path testing:

* Flow graph notation
* Kilometric complexity
* Deriving test cases
* Graph matrices Control

**BLACK BOX TESTING**

In this testing by knowing the internal operation of a product, test can be conducted to ensure that “all gears mesh”, that is the internal operation performs according to specification and all internal components have been adequately exercised. It fundamentally focuses on the functional requirements of the software.

The steps involved in black box test case design are:

**SOFTWARE TESTING STRATEGIES:**

A software testing strategy provides a road map for the software developer. Testing is a set activity that can be planned in advance and conducted systematically. For this reason a template for software testing a set of steps into which we can place specific test case design methods should be strategy should have the following characteristics:

Testing begins at the module level and works “outward” toward the integration of the entire computer based System.

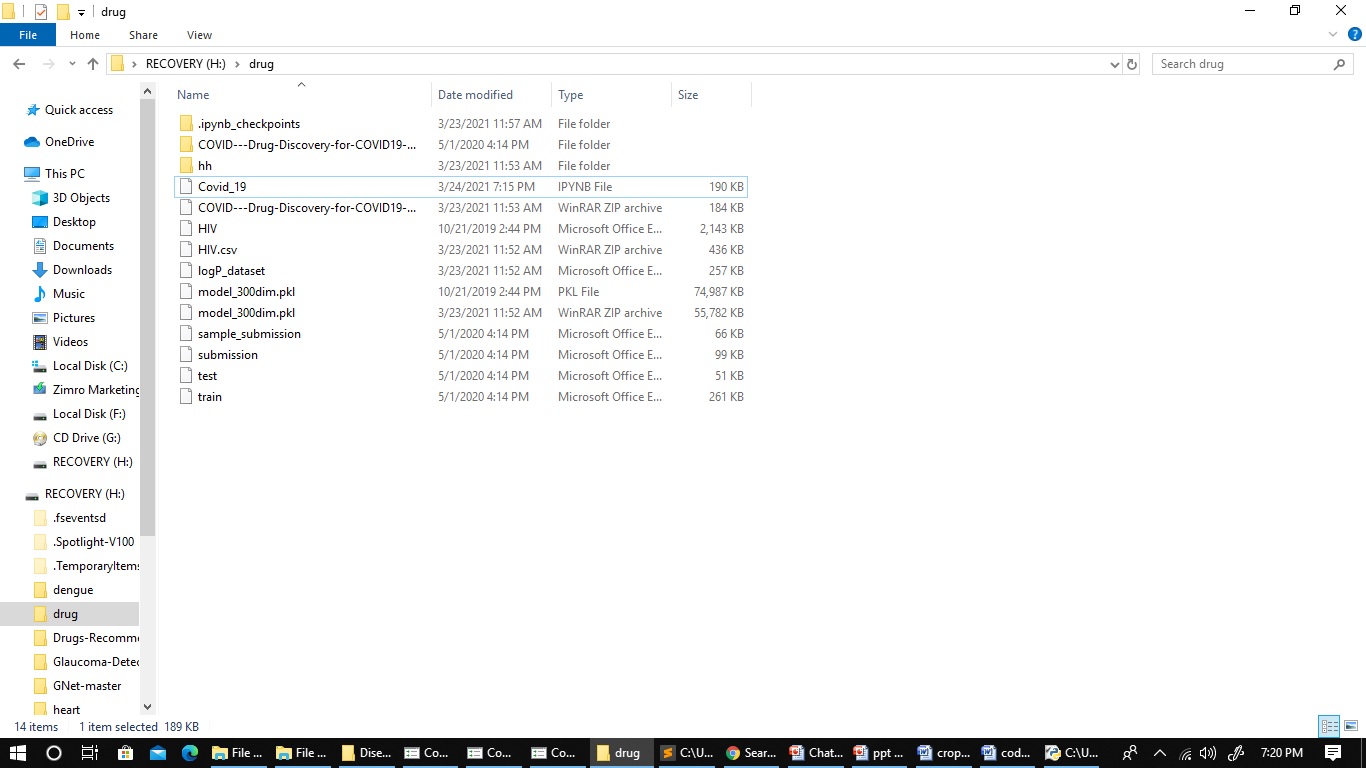
**CHAPTER 8**

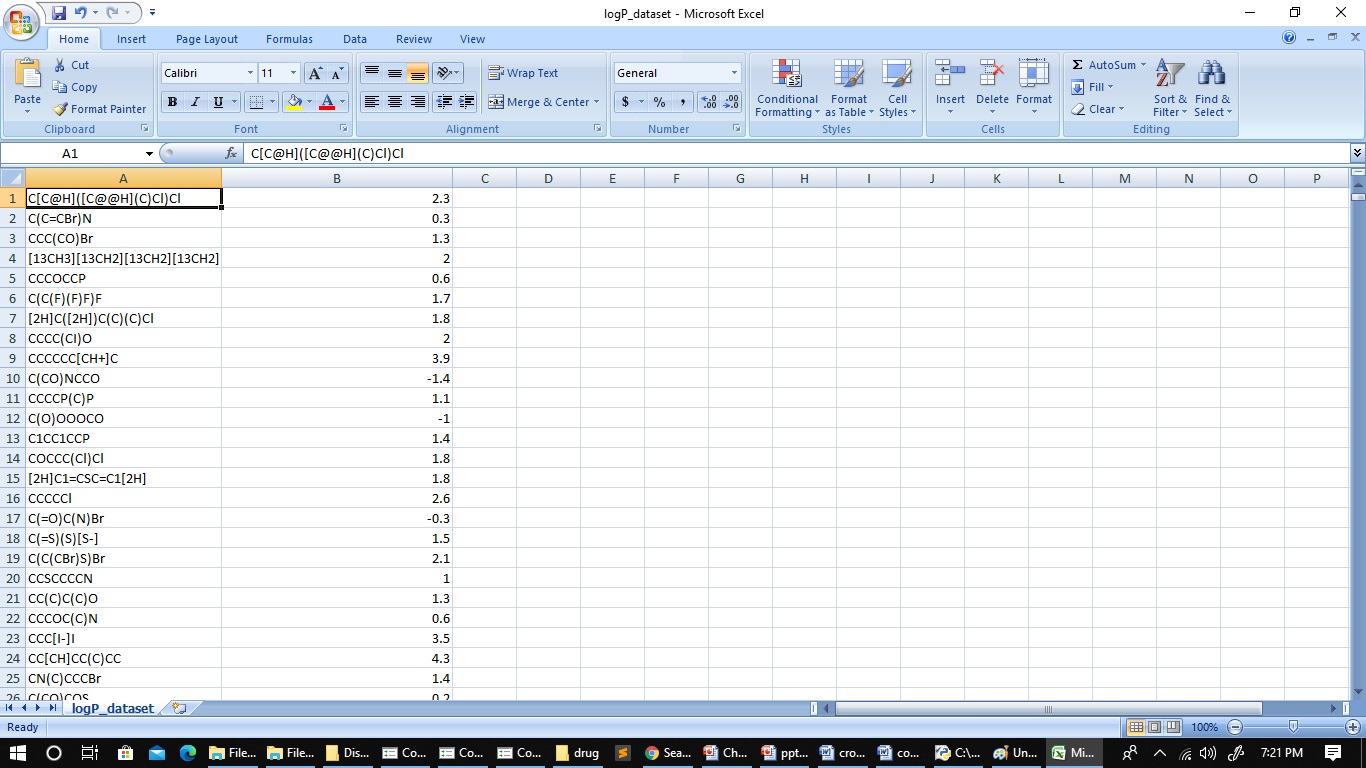
**CONCLUSION**

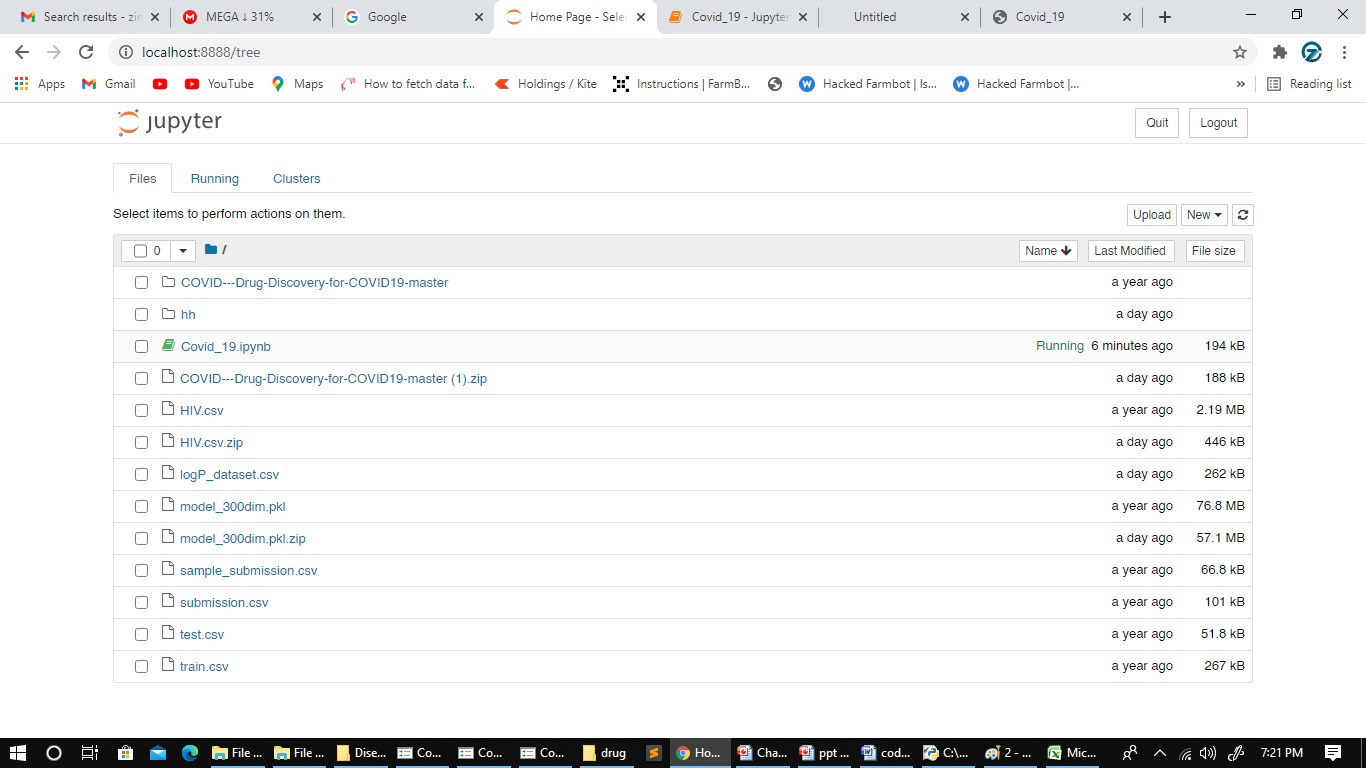
ML-based techniques seek to revitalize the development of drugs. These methods are based on separate applications in target discovery, lead compound discovery, synthesis, protein-ligand interactions, etc. ML applications are paving the way for algorithm-enhanced data query, analysis, and generation. One such example is ML incorporated into target discovery, based heavily on the refinement and search of existing omics and medical data. Through AI integration using ML techniques, viable targets can be found using data clustering, regression, and classification from vast omics databases and sources. Lead compound discovery, e.g., using QSAR, is currently frequently used to develop sensible molecular candidates based on training inputs. Lead compound synthesis has also been expedited with NN-based retrosynthesis algorithms alongside best-chance trees with the input of vast amounts of accumulated data and rules to develop algorithms that can generate synthesis pathways with greater than 90% accuracy in 60 s. Applications of ML in the processes of drug development have been used for some time now. These applications have proven to be steps above previous methods; the development of ML and DL techniques are not all brand new. They have been carefully crafted and developed through decades of research. This curation of function and utility to ML algorithms and techniques has allowed for the continued success and development in drug discovery. Owing to more precise algorithms, more powerful supercomputers, and substantial private and public investment into the field, these applications are becoming more intelligent, cost-effective, and time-efficient while boosting efficacy

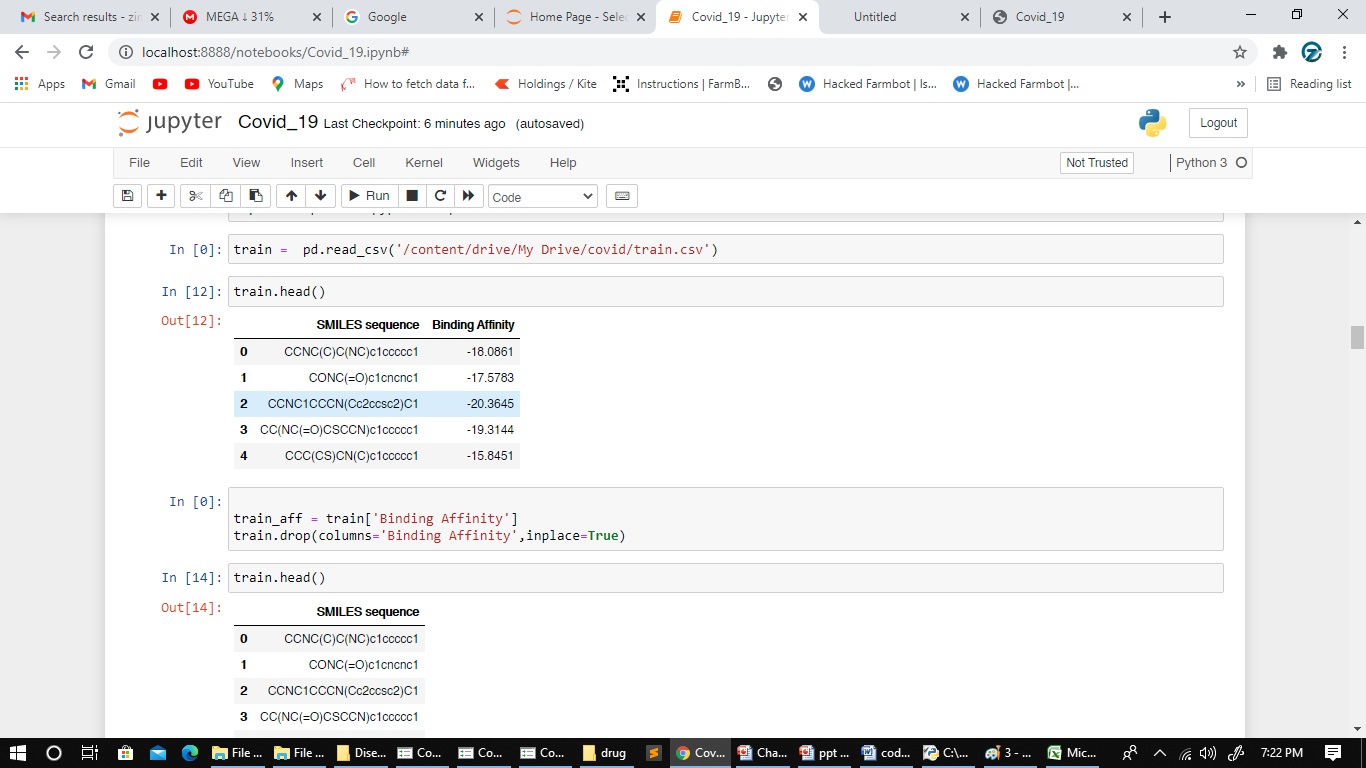
**APPENDICES**

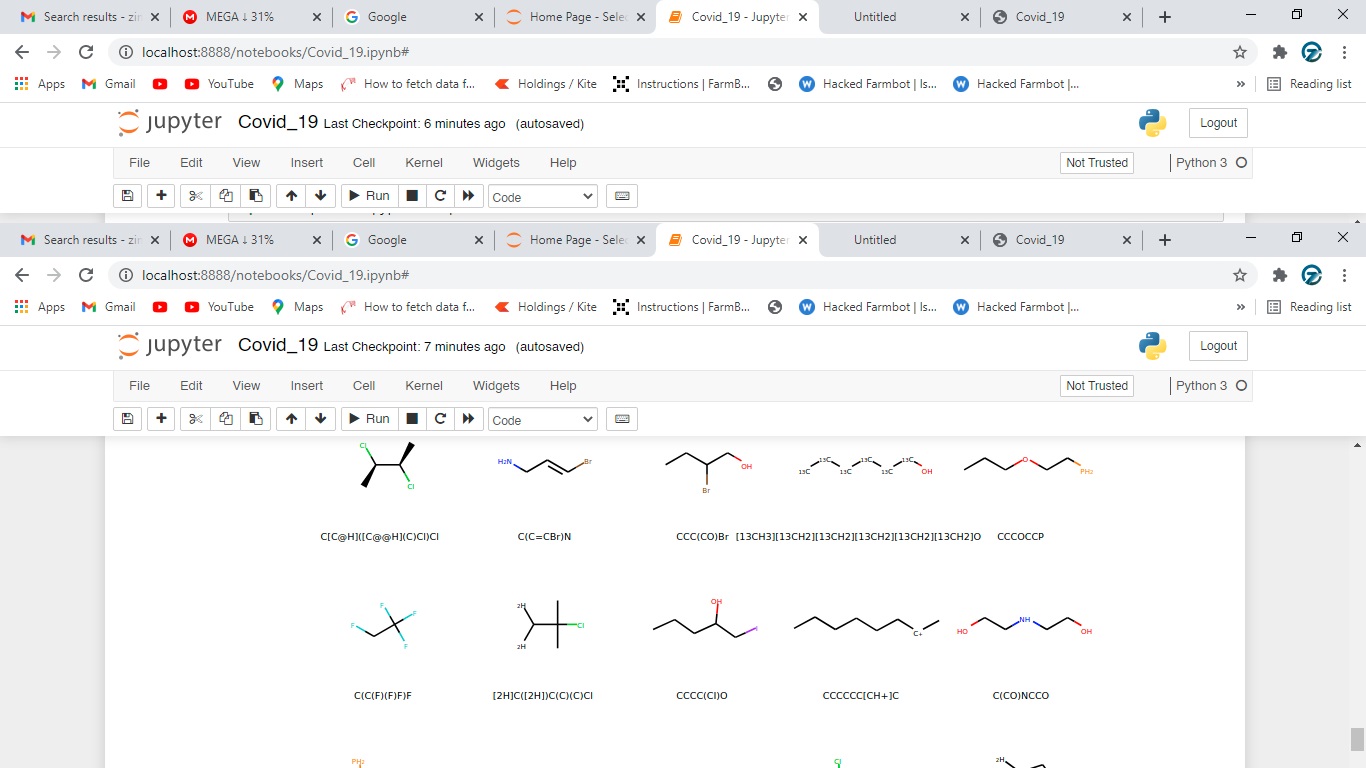
**SAMPLE SCREENS**











**REFERENCE**

1. Fayyad, J.; Jaradat, M.A.; Gruyer, D.; Najjaran, H. Deep Learning Sensor Fusion for Autonomous Vehicle Perception and Localization: A Review. Sensors 2020, 20, 4220. [CrossRef] [PubMed]

2. Deng, L.; Li, X. Machine Learning Paradigms for Speech Recognition: An Overview. IEEE Trans. Audio Speech Lang. Process. 2013, 21, 1060–1089. [CrossRef]

3. Afouras, T.; Chung, J.S.; Senior, A.; Vinyals, O.; Zisserman, A. Deep Audio-visual Speech Recognition. IEEE Trans. Pattern Anal. Mach. Intell. 2018, 1. [CrossRef] [PubMed]

4. Joachims, T.; Radlinski, F. Search Engines that Learn from Implicit Feedback. Computer 2007, 40, 34–40. [CrossRef]

5. Morgan, S.; Grootendorst, P.; Lexchin, J.; Cunningham, C.; Greyson, D. The cost of drug development: A systematic review. Health Policy 2011, 100, 4–17. [CrossRef] [PubMed]

6. Ng, H.W.; Zhang, W.; Shu, M.; Luo, H.; Ge, W.; Perkins, R.; Tong, W.; Hong, H. Competitive molecular docking approach for predicting estrogen receptor subtype α agonists and antagonists. BMC Bioinform. 2014, 15, S4. [CrossRef] [PubMed]

7. Ng, H.W.; Shu, M.; Luo, H.; Ye, H.; Ge, W.; Perkins, R.; Tong, W.; Hong, H. Estrogenic activity data extraction and in silico prediction show the endocrine disruption potential of bisphenol A replacement compounds. Chem. Res. Toxicol. 2015, 28, 1784–1795. [CrossRef] [PubMed]

8. Hong, H.; Neamati, N.; Winslow, H.E.; Christensen, J.L.; Orr, A.; Pommier, Y.; Milne, G.W.A. Identification of HIV-1 integrase inhibitors based on a four-point pharmacophore. Antivir. Chem. Chemother. 1998, 9, 461–472. [CrossRef] [PubMed]

9. Hong, H.; Tong, W.; Xie, Q.; Fang, H.; Perkins, R. An in silico ensemble method for lead discovery: Decision forest. SAR QSAR Environ. Res. 2005, 16, 339–347. [CrossRef]

10. Hong, H.; Fang, H.; Xie, Q.; Perkins, R.; Sheehan, D.M.; Tong, W. Comparative molecular field analysis (CoMFA) model using a large diverse set of natural, synthetic and environmental chemicals for binding to the androgen receptor. SAR QSAR Environ. Res. 2003, 14, 373–388. [CrossRef]

11. Lo, Y.C.; Rensi, S.E.; Torng, W.; Altman, R.B. Machine learning in chemoinformatics and drug discovery. Drug Discov. Today 2018, 23, 1538–1546. [CrossRef] [PubMed]

12. Talevi, A.; Morales, J.F.; Hather, G.; Podichetty, J.T.; Kim, S.; Bloomingdale, P.C.; Kim, S.; Burton, J.; Brown, J.D.; Winterstein, A.G.; et al. Machine Learning in Drug Discovery and Development Part 1: A Primer. CPT Pharmacomet. Syst. Pharmacol. 2020, 9, 129–142. [CrossRef] [PubMed]

13. Gertrudes, J.C.; Maltarollo, V.G.; Silva, R.A.; Oliveira, P.R.; Honório, K.M.; da Silva, A.B. Machine learning techniques and drug design. Curr. Med. Chem. 2012, 19, 4289–4297. [CrossRef]

14. Agarwal, S.; Dugar, D.; Sengupta, S. Ranking chemical structures for drug discovery: A new machine learning approach. J. Chem. Inf. Model. 2010, 50, 716–731. [CrossRef] [PubMed]

15. Rodrigues, T.; Bernardes, G.J.L. Machine learning for target discovery in drug development. Curr. Opin. Chem. Biol. 2020, 56, 16–22. [CrossRef]