NOVA IMS Information Management School

MGI

Mestrado em Gestão de Informação
Master Program in Information Management

Using PubChem's database with Data Mining and Machine Learning Algorithms for the prediction of EGFR inhibitors

A comparative study

Liliana Monteiro Rosa

Dissertation presented as partial requirement for obtaining the Master's degree in Information Management

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa


2017
Title: Using PubChem's database and Machine Learning Algorithms for the prediction of EGFR inhibitors
Subtitle: A comparative study
Student
full name: Liliana Monteiro Rosa
MGI


2017
Title: Using PubChem's database and Machine Learning Algorithms for the prediction of EGFR inhibitors
Subtitle: A comparative study
Student
full name: Liliana Monteiro Rosa
MGI

NOVA Information Management School

Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa

USING PUBCHEM'S DATABASE WITH DATA MINING AND MACHINE LEARNING ALGORITHMS FOR THE PREDICTION OF EGFR INHIBITORS: A COMPARATIVE STUDY

by

Liliana Monteiro Rosa

Dissertation presented as partial requirement for obtaining the Master's degree in Information Management, with a specialization in Knowledge Management and Business Intelligence.

Advisor: Mauro Castelli

August 2017

ACKNOWLEDGEMENTS

Firstly, I would like to thank Professor Mauro Castelli for accepting being my Advisor in this thesis and for contributing to my interest in this topic with his classes.

Thank you to all my family but most of all to my mother, who made me who I am and to whom I owe everything that I have accomplished, academic or otherwise.

I also need to thank my aunt Clarisse who was a second mother to me and provided me with incredible support throughout my academic path.

To Ana, for being my person, both in solidarity in the path for completing a Master's degree and in everything else in life.

To Rúben, whose support was ever present both in the decision to enroll in this Master's degree and during its completion and in so many other times, even to his own detriment.

Thank you all.

ABSTRACT

Data Mining and Machine Learning algorithms and methods have become increasingly important for several industries due to the amount of available data that has grown exponentially in recent years and led to the need of effective ways of gaining insights from that data.

In this study, these methods are applied to the prediction of Epidermal Growth Factor Receptor inhibitors using data extracted from PubChem's database. PubChem is a freely accessible chemical repository that contains information submitted from several different sources, and that comprises three databases, one of which provides information about BioAssays, that is, assays with the purpose of screening numerous compounds for activity on a particular biological target. In this work, the dataset used to train and evaluate the developed models resulted from the information gathered from the assays performed to identify inhibitors of EGFR and the source for the features used to characterize the compounds was PubChem's own chemical descriptor, the Substructure Fingerprint.

The work comprises a literature review on this subject and the implementation of a methodology that tests the performance of different types of classifiers for the problem at hand, namely Naïve Bayes, Decision Tree, Logistic Regression, k-Nearest Neighbors, Support Vector Machine, Multilayer Perceptron, Random Forest, Extremely Randomized Trees, Bagging, Boosting and Voting.

Considering both the evaluated quality metrics and the model's computational burden, the Multilayer Perceptron was considered the best model, although some of the other models had close performances.

It was concluded that the used methodology and developed models had good quality, as did PubChem's Substructure Fingerprint as a descriptor, but that there was still room for improvement that could be achieved with further experimentation on different aspects of the methodology.

KEYWORDS

Data Mining; Machine Learning; Epidermal Growth Factor Receptor; PubChem

INDEX

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

DT	Data Mining
ML	Machine Learning
IC_{50}	Half Maximal Inhibitory Concentration
NB	Naïve Bayes
DT	Decision Tree
SVM	Support Vector Machine
MLP	Multilayer Perceptron
ERT	Extremely Randomized Tree
RF	Random Forest
HTS	High Throughput Screening
CV	Cross Validation
NIH	National Institutes of Health

1. INTRODUCTION

The cost of drug development has been increasing, reaching, in 2013, an estimated average out-of-pocket cost per approved new compound of $1395 million (2013 dollars). This study indicates that the total capitalized cost has increased at an annual rate of 8,5% above general price inflation. (DiMasi, Grabowski, & Hansen, 2016)

From these figures arises the question of whether the current rate of investment is sustainable (Berndt, Nass, Kleinrock, & Aitken, 2015), and, as such, there is a need to stimulate basic research in this field outside of the Pharmaceutical Industry.

With the purpose of expanding the availability, flexibility, and use of small-molecule chemical probes for basic research, the Molecular Libraries Initiative (MLI) component of the NIH (National Institutes of Health) Roadmap for Medical Research was created. This initiative had three components: the Molecular Libraries Screening Centers Network; cheminformatics initiatives including a new public compound database (PubChem); and technology development initiatives in chemical diversity, cheminformatics, assay development, screening instrumentation, and predictive ADMET (absorption, distribution, metabolism, excretion and toxicity). (Austin, Brady, Insel, & Collins, 2004)

PubChem ("The PubChem Project," n.d.) provides information about the biological activities of small molecules. It was released in 2004 and is organized in three different databases: PubChem Substance, PubChem Compound, and PubChem BioAssay.

This great amount of information and the way it is kept up to date and organized makes for a great resource when it comes to research regarding the identification of compounds that show potential for activity on a particular biological target, which can be either a protein or a gene.

As such, PubChem allows, amongst other research possibilities, the elaboration of databases of groups of compounds that have a higher probability of having the desired biological activity on a specific target. For this purpose, an *in-silico* screening of the compounds can be performed using predictive models to identify the compounds of interest that can then be physically tested.

1.1. OBJECTIVES AND STUDY RELEVANCE

This study has two main objectives. First, to conduct a review of the studies previously performed using predictive models on PubChem datasets or classifier models for the same purpose as the present study. Second, to conduct a study aiming to obtain a model capable of predicting inhibitory activity of compounds on the Epidermal Growth Factor Receptor (EGFR) using the PubChem database and its Structure Fingerprint as the basis for the feature creation, leading to a final comparison of the performance of the different tested algorithms.

The Epidermal Growth Factor Receptor is an important drug development target as its overexpression has been linked to numerous Human cancers (Yarden & Sliwkowski, 2001).

Although there have been a small number of previous works where classifier models were built for the purpose of identifying inhibitors of EGFR that achieved good results, which will be presented ahead, these have all used datasets of much smaller size than PubChem's Bioassay database for this target, accounting for a lower diversity in the chemical groups analyzed. These studies have also used different

molecular descriptors from various sources. In this work, the aim is to evaluate the effectiveness of predicting models built for the screening of PubChem's database considering its vast size, using its own molecular descriptor and freely available and common data science tools.

The previous studies have also used a small number of algorithms each, while here the intent is to use a wider variety of models, some of which have not been used before, that will follow the same methodology. This will allow a comparison of their quality and the election of the most suitable one for the problem at hand.

As such, this study intends to contribute with information regarding a methodology that could be applied in similar studies, and that could potentially cut costs and resources usage for drug development.

1.2. Document Structure

This document will be comprised of two main parts.

The first part will be the Literature Review that will be divided into four sections:

- PubChem – Brief description of PubChem and its characteristics that make it a valuable resource for drug research and in particular for virtual screening.
- Review of Data Mining/Machine Learning predictive studies performed using PubChem Data for virtual screening of compounds of interest.
- EGFR – Description of the Epidermal Growth Factor Receptor and of the importance of having the capability of inhibiting it.
- Data Mining and Predictive Algorithms – Theoretical framework of data mining usage for predictive classification and of the algorithms used in this study.

The second part of the document will describe the work done, namely, its methodology, results, discussion, and conclusions.

2. LITERATURE REVIEW

2.1. PUBCHEM

PubChem is a repository of information about chemical substances and compounds and their biological activity tests, that is open to the public. It was first launched in 2004 with the objective of discovering chemical probes through high-throughput screening of small molecules that modulate the activity of gene products. It was developed and is maintained by the U.S NIH. (Y. Wang et al., 2009)

High-throughput screening (HTS) is a technique that has become widely used for the discovery of new drugs in the last 20 years. It is a method that allows the screening and assaying of a large number of compounds against selected targets. These assays can be used for screening different types of libraries, including combinatorial chemistry, genomics, and protein libraries and are performed by both the pharmaceutical industry and academic researchers. This process saves time and decreases costs in drug development by allowing the screening of a vast number of compounds in a short period of time. High-throughput screening is also used to characterize the new drugs in terms of their metabolic, pharmacokinetic and toxicological data. (Szymánski, Markowicz, & Mikiciuk-Olasik, 2012)

For this process, plates are used with the biological target placed in the wells and put in contact with the several compounds to be tested. The activity against the target is then assessed using different techniques that depend on the type of target. Initially, primary screens are conducted that are less quantitative than subsequent confirmatory assays and that provide an identification of positive results. These positive results usually give rise to a secondary screening where quantitative values, such as the IC_{50}, (the value of the needed concentration of the compound to inhibit 50 % of the biological process), are determined. (Szymánski et al., 2012)

With the use of both HTS and combinational chemistry, the production of bioactivity data is now available at a lower cost which allows this activity to be performed by small research laboratories and academic institutions, while before it was mostly confined to the pharmaceutical industry. Both the adoption of data mining techniques that allow an easier collection of chemical information from various sources such as scientific articles and patent documents, and the policies that foment data sharing that have been implemented by funding agencies and journal publishers, have given rise to a great increase in the amount of chemical data that is available to the public (S. Kim, 2016). This, together with the emergence of big data, has led to a rising interest from the research community in the use of virtual screening (S. Kim, 2016). As such, virtual screening has been an increasingly used method to identify new compounds of interest even though it is not free of pitfalls that can compromise efficiency and that have been documented (McInnes, 2007).

Taking this landscape into consideration, PubChem presents itself as a major source of data suitable for virtual screening as it contains chemical information from more than 525 sources ("PubChem Data Sources," n.d.) at the time of writing.

The data found in PubChem is mostly about small molecules, but other chemical structures can also be found, including, amongst others, small-interfering RNAs and micro-RNAs, proteins, carbohydrates, and lipids. PubChem's data is organized in three databases: Substance, Compound, and Bioassay. As of July 2017, PubChem contains information regarding over 93 million compounds, 234 million substances, and 1.2 million BioAssays. ("The PubChem Project," n.d.)

These databases provide different information, with the Substance database containing the chemical information for the substances submitted by the various data contributors while the Compound database stores the information regarding the unique chemical structures derived from the submitted substances through the PubChem standardization process (S. Kim et al., 2016). In the Bioassay database is stored the information about the individual Bioassays, that is, about the studies performed to determine the bioactivity of molecules on a specific target, and the respective results, providing datasets with the tested compounds and their classification as active or inactive (Y. Wang et al., 2014). The substances, compounds, and bioassays included in PubChem's databases all have unique identifiers, respectively: SID (Substance ID), CID (Compound ID), and AID (Assay ID) (S. Kim, 2016).

The origin and organization of PubChem's data are presented in the following figure:

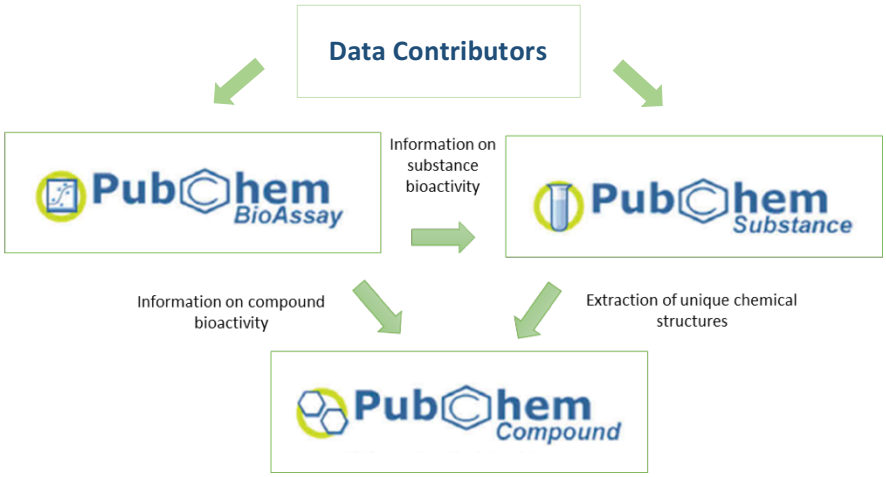

Figure 1 - PubChem Structure

There are several features that make PubChem a valuable resource for the implementation of virtual screening and subsequent physical testing of the compounds of interest identified (Kim, 2016):

- The complementary bioactivity information to the bioassay data that is extracted from scientific articles and is provided by contributors that include ChEMBL (Bento et al., 2014), PDBbind (Liu et al., 2015), BindingDB (Gilson et al., 2016), and IUPHAR/BPS Guide to Pharmacology (Southan et al., 2016).

- The availability of compound related information that complements the bioactivity data. This includes properties of FDA approved and investigational drugs provided by DrugBank (Law et al., 2014) such as their indications, mechanisms of action, target macromolecules, interactions with proteins and genes, and ADMET properties; toxicology data provided by the Hazardous Substances Data Bank (Fonger, Hakkinen, Jordan, & Publicker, 2014) and 3-D structures of small molecules provided by the Molecular Modelling Database (Madej et al., 2014);

- Assurance of compound availability for subsequent testing. There is always at least one data contributor for each compound in PubChem who claims to have it. If the compound becomes unavailable for whatever reason, it becomes nonlive, meaning that it cannot be searched but is still present in the database.

- Easy identification of existent patents for compounds. Given the importance of being able to patent potential drugs in research programs, PubChem offers links between patent documents from U.S., Europe and Word Intellectual Property Organization (WIPO) and unique chemical structures.

- Possibility of inferring, for less studied compounds, their chemical characteristics through the comparison with structurally similar compounds. PubChem makes this process easier by providing a precomputed list of such molecules, called "neighbors". This similarity can be computed in relation to 2-D or 3-D characteristics, giving rise to 2-D or 3-D neighbors.

- Programmatic access to PubChem. There are several programmatic ways by which it is possible to access PubChem's data. These include Entrez Utilities, Power User Gateway (PUG), PUG-SOAP, and PUG-REST. It is also possible to download the databases in different file formats including XML and SDF (Spatial Data File – a type of chemical data file format) through the File Transfer Protocol.

2.2. REVIEW OF WORKS USING VIRTUAL SCREENING

There have been several published works where virtual screening of PubChem was performed with different approaches. Here will be mentioned the ones found most relevant in terms of similarity of objectives with the study at hand.

Chen and Wild (2010) have conducted a study where a Naïve Bayes model was created to predict activity using 1133 bioassays from the PubChem database. A particular type of molecular fingerprints (FCFP_6 circular) and their encoded structural features were used for the model creation. The authors concluded that Bayesian models generated from PubChem Datasets are reasonably accurate but that the variability in their accuracy is still quite higher than that observed in the more traditional QSAR (Quantitative Structure-Activity Relationship) modeling used for drug research (Cherkasov et al., 2014). This fact makes them suitable for virtual screening where the main objective is to obtain a smaller number of potentially active compounds to be tested, but not for prediction where high levels of accuracy are needed. The authors point that a way to improve the accuracy would be to include more information about inactive compounds. (Chen & Wild, 2010)

In 2008, Weis, Visco and Faulon (2008) developed a Support Vector Machine (SVM) classifier to identify compounds that act as Factor XIa inhibitors. The classifier was trained on one of the bioassays conducted for the identification of Factor XIa inhibitors, using the Signature molecular descriptor. The bioassay used was chosen considering two factors: the fact that it was a confirmatory assay which diminishes the number of false positives present and that it had a balanced number of positive and negative compounds which eliminates a problem that is common in HTS data. This problem resides in the fact that the sets are usually highly imbalanced with a much smaller number of active than inactive compounds, leading to classifiers that may have high accuracy but are unable to identify the positive compounds, a problem that will also be further mentioned in this study. After implementation of the authors' version of recursive cluster elimination for feature selection, the final model's 10-fold cross-validation accuracy was improved from that of a random classifier to 89%, proving its adequacy for the

usage in the case at hand. It was also used to screen the 12 million compounds that were present at the time in PubChem's database. (Weis, Visco, & Faulon, 2008)

Han et al. (2012), evaluated Support Vector Machines for the identification of Src inhibitors in large compound libraries by training and testing the models on 1703 inhibitors and 63318 putative non-inhibitors reported before 2011, with the result of correctly identifying 93.53%~ 95.01% inhibitors and 99.81%~ 99.90% non-inhibitors, in 5-fold cross validation studies. The model correctly identified 70.45% of the 44 inhibitors reported since 2011, with the model being applied both to the complete PubChem database and the MDDR database (A bioactivity database produced by BIOVIA and Thomson Reuters with information gathered from patent literature, journals, meetings, and congresses ("BIOVIA Databases | Bioactivity Databases: MDDR," n.d.)). (B. Han et al., 2012)

A study to predict activity against parasitic nematodes was conducted by Khanna and Ranganathan (2011) where a Support Vector Machine model was trained using data from various sources including PubChem to gather the active compounds, while the inactive compound set was derived from DrugBank (Law et al., 2014). The validation of each model was done using ten-fold stratified cross-validation and the best results, with an accuracy of 81.79% on an independent test set, were obtained using the radial basis function kernel. The authors concluded that the developed model would be able to identify new potentially anthelmintic active compounds. (Khanna & Ranganathan, 2011)

Prediction of activity on human ether-a-go-go related gene (hERG) potassium ion channel was performed by Shen, Su, Esposito, Hopfinger and Tseng (2011) using an SVM model and different hERG Bioassay datasets for training and validation, while a test set was derived from literature data. The evaluation was performed with 10-fold cross-validation, and the best model had an accuracy, sensitivity, and specificity of 95%, 90% and 96%, respectively and an overall accuracy for the testing set of 87%. The conclusions drawn by the authors were that the model was able to predict "predisposition" to block hERG ion channels and that it was robust across the structural diversity of the training set. (Shen, Su, Esposito, Hopfinger, & Tseng, 2011)

Cheng et al. (2011) conducted a study to identify inhibitory activity of compounds on cytochrome P450 (CYP) since its inhibition is an important factor in drug-drug interactions. For this purpose, the authors used a data set composed of 24700 compounds extracted from PubChem and a combined classifier algorithm. This algorithm is an ensemble of different independent machine learning classifiers: Support Vector Machine, C4.5 Decision Tree, k-Nearest Neighbors And Naïve Bayes, fused by a back-propagation Neural Network. The models were validated by 5-fold cross-validation and separate validation set. The results obtained led to the conclusion that these models are applicable to the virtual screening of inhibitors of the different isoforms of CYP. (Cheng et al., 2011)

Another study to predict inhibitory activity on CYP was conducted by Su et al. (2015). The authors used a rule-based C5.0 algorithm and different descriptors, including, amongst others, PubChem's Substructure fingerprints. An algorithm of rational sampling was also developed to select compounds from the training set in order to enhance the performance of the models. The optimized model showed improvements in relation to previously existing models, being useful for the screening of large data sets of compounds and also providing the most important rules to identify probable inhibitors which can give new insights about the structural features that are important for this activity. (Su et al., 2015)

Han, Wang and Bryant (2008) developed Decision Tree (DT) models to predict activity of 5HT1a agonists, antagonists, and HIV-1 RT-RNase H inhibitors using PubChem Bioassay data and compound fingerprints. The models were evaluated by 10-fold cross-validation and obtained sensitivity, specificity and Mathews Correlation Coefficient (MCC) in the ranges 57.2%-80.5%, 97.3%-99.0% and 0.4-0.5 respectively, with the conclusion that the DT models developed can be used for virtual screening as well as a complement to other more traditional approaches to activity prediction. (L. Han, Wang, & Bryant, 2008)

Schierz (2009) used Weka's (Hall et al., 2009) cost-sensitive implementation of four classifiers (Support Vector Machines, C4.5 Decision Tree, Naïve Bayes And Random Forest) that were applied to data from several Bioassays on different targets. The authors concluded that Weka's implementations of the Support Vector Machine and C4.5 Decision Tree learner performed relatively well and that care should be taken with the use of primary screenings for their high number of false positives as well as with the use of Weka's cost-sensitive classifiers as "across the board misclassification costs based on class ratios should not be used when comparing differing classifiers for the same dataset." (Schierz, 2009, p. 21)

A consensus model using the k-Nearest Neighbor algorithm was developed by Chavan, Abdelaziz, Wiklander and Nicholls (2016) for the classification of hERG potassium channel blockers. The authors first constructed 8 models based on 8 different kinds of signatures, one of which was PubChem's Substructure Signature, that were obtained for a data set of 172 channel blockers that was created based on information retrieved from OCHEM ("Online Chemical Modeling Environment," n.d.) and Fenichel ("Receptor Binding," n.d.). The authors then created consensus models based on majority rule and on the sensitivity of the individual models (as in this case it was more important the ability to identify active compounds) using 3, 5 and 7 different signatures. The final consensus model showed sensitivity and specificity of 0.78 and 0.61 for the internal dataset compounds and 0.63 and 0.54 for an external validation set of PubChem data. (Chavan, Abdelaziz, Wiklander, & Nicholls, 2016)

Wang, Xie, Wang, Zhu and Niu (2016) applied four different Machine Learning algorithms to the prediction of selective estrogen receptor beta (ER-β) agonist activity: Naïve Bays, k-Nearest Neighbor, Random Forest and Support Vector Machine. The data about the active chemical structures was retrieved from public chemogenomics databases and five types of chemical descriptors were used, including PubChem's fingerprint. The models were evaluated by 5-fold cross-validation with a reported range of classification accuracies between 77.10% and 88.34%, and a range of area under the ROC (receiver operating characteristic) curve between 0.8151 and 0.9475. The study suggests that the Random Forest and the Support Vector Machine classifiers are more suited for the classification of selective ER-β agonists than the other evaluated classifiers. (S. Wang, Xie, Wang, Zhu, & Niu, 2016)

Novotarskyi, Sushko, Körner, Pandey and Tetko (2011) conducted a study that compared different models for their efficacy in predicting bioactivity on CYP1A2. These models were built using various combinations of Machine Learning methods and chemical descriptors. The ML algorithms used were Associative Neural Networks ("a combination of an ensemble of feed-forward neural networks and KNN" (Novotarskyi, Sushko, Körner, Pandey, & Tetko, 2011)), k-Nearest Neighbors, Random Tree, C4.5 Decision Tree and Support Vector Machine with all of the models being also used in combination with the Bagging technique. The different combinations of these methods with the different chemical descriptors and the usage of either the full set of descriptors or a subset of selected ones resulted in 80 evaluated models. The authors concluded that descriptor selection did not improve the quality of

the models and that the best performing model was ASNN with the full descriptor set with 83% and 68% of accuracy in the internal and external test sets, respectively. (Novotarskyi et al., 2011)

In a work conducted by Pouliot, Chiang and Butte (2011) the authors built Logistic Regression models with the goal of correlating postmarketing adverse reactions (ADRs) with screening data from PubChem's Bioassay database. The developed pipeline used 508 BioAssays of the PubChem database with 485 different drug components. The ADRs were grouped in different system organ classes and models were built for each of these. The models were evaluated using Leave One Out Cross-Validation (LOOCV) and the authors report a better performance than expected given the simplicity of the Logistic Regression models with half of the models having an AUC of ≥ 0.7 and all of the models having AUC $\geq$ 0.6. (Pouliot, Chiang, & Butte, 2011)

Recently, Yu, Shi, Tian, Gao and Li (2017) presented a study with the objective of developing a model for the classification of CYP450 1A2 inhibitors and non-inhibitors using a multi-tiered deep belief network (DBN) on a large dataset. The authors used a dataset of over 13000 compounds from PubChem and 245 molecular descriptors including both 2D and 3D descriptors that were calculated by molecular computational software. With the objective of improving the classifier's performance and decrease the computational time a descriptor selection was performed by implementation of three rules: removal of descriptors with too many zeros, with small standard deviation values (< 0.5%) and with correlation coefficients higher than 0.9. For comparison purposes, shallow machine learning models were also trained, namely Support Vector Machine and Artificial Neural Network. All models were run several times to determine the best parameters to use and evaluated by 5-fold cross validation and by an external dataset. The best results were obtained by the DBN model using both 2D and 3D descriptors with an internal overall accuracy of 83.6% and an external accuracy of 77.0%. (Yu, Shi, Tian, Gao, & Li, 2017)

In the study by Bilsland et al. (2015), Artificial Neural Networks were used for virtual screening of Selective G1-Phase Benzimidazolone inhibitors. The dataset used was derived from PubChem Bioassay data with the authors opting to reduce the number of inactive compounds by applying similarity filters using chemical software to obtain a balanced dataset. The final training set contained 3924 compounds of which 1859 were active and 2065 were inactive. The descriptors used included PubChem's fingerprints amongst others, resulting in a group of 2780 features. Successive runs were performed to identify the best parameter selection, to eliminate compounds that were consistently misclassified and to determine the optimal subset of descriptors. The authors opted to use as a final model an ensemble of 10 networks trained using the optimal combination of parameters, compounds and descriptor set. The overall sensitivity, specificity and accuracy evaluated by 10-fold cross validation were, respectively, 83.1%, 82.4% and 82.7% with the authors considering these results to indicate very good predictive performance. (Bilsland et al., 2015)

While most of the QSAR studies for this target have used regression techniques, the works described below have used classification models to predict the inhibitory activity of compounds on EGFR:

In the work published by Kong, Qu, Chen, Gong and Yan (2016) the authors have developed models for the classification of compounds as inhibitors or non-inhibitors of EGFR by using Kohonen's Self-Organizing Map (SOM) and Support Vector Machine (SVM) algorithms. The used dataset was compiled from CHEMBL (Bento et al., 2014) keeping only the compounds with inhibitory concentration (IC_{50}) under 10 μM, resulting in 1248 inhibitors. For the inactive compounds 3093 decoys where gathered

from the DUD database (Huang, Shoichet, & Irwin, 2006). A PCA analysis was performed on some properties to determine that there was overlapping in the active and inactive compounds as to make the identification challenging. The final dataset was divided into training and test set to perform evaluation of the model. For the molecular descriptors, ADRIANA.Code (Gasteiger†, 2006) descriptors were calculated and then a subset selected based on correlation with activity. The authors reported that the final models had prediction accuracies on training and testing set, respectively, of 98.5% and 96.3% for the SOM model and 99.0% and 97.0% for the SVM model, and sensitivity, specificity and MCC, respectively, of 94.0%, 97,3% and 0.91 for the SOM model and 94.2%, 98.2% and 0.93 for the SVM model, concluding that both models had good performance when distinguishing between inhibitors and decoys of EGFR. (Kong, Qu, Chen, Gong, & Yan, 2016)

Zhao et al. (2017) conducted a recently published study where the authors constructed 2D and 3D-QSAR models with the 2D model being built using a Support Vector Machine classifier. The used dataset was constructed using 100 inhibitors retrieved from the literature and 185 inhibitors from the DUD database. For the 2D study the dataset was divided into three training sets which accounted for 75%, 70% and 50% of the whole dataset. Forty-five molecular descriptors where calculated using ChemOffice (Irwin*, 2005) and a subset of 9 descriptors was selected using Correlation-Based Feature Selection combined with Genetic Search algorithms. The training of the model was conducted with the three different training sets with the authors opting to use the dataset accounting for 70% of the data which led to the higher accuracy. The final model presented sensitivity, specificity, accuracy and MCC of 98.55%, 99.23%, 98,99% and 0.978, respectively on the training set evaluated by ten-fold cross-validation and 96.77%, 98.18%, 97.67% and 0.950 on the test set, indicating good performance of the model. (Zhao et al., 2017)

Singh et al. (2015) developed a model for the classification of compounds as inhibitors or non-inhibitors of EGFR. For this purpose, the authors obtained 3528 anti-EGFR compounds and their inhibitory concentration (IC_{50}) from a database that was also developed by the authors and that contains information gathered from around 350 research articles. With these compounds, the authors built three different data sets with different proportions of active and inactive compounds according to the chosen IC_{50} level threshold: EGFR10, EGFR100, EGFR1000. As the chemical descriptor, the authors used PubChem's Substructure fingerprints calculated for the compounds present in the datasets built by the authors. Each of the datasets was divided into training and validation sets for the purpose of model evaluation and different ML algorithms implemented in Weka were applied, including IBK (an implementation of k-Nearest Neighbors), Naïve Bayes, Support Vector Machine and Random Forest. The authors found that the best performing model was the Random Forest with an accuracy, sensitivity, specificity, and MCC of 83.66%, 69.89%, 86.03% and 0.49, respectively, when evaluated on the EGFR10 dataset which had the lowest threshold of IC_{50} and thus a smaller proportion of active compounds. (Singh et al., 2015)

2.3. Epidermal Growth Factor Receptor (EGFR)

The Epidermal Growth Factor Receptor (EGFR) or erbB1/HER1 is a transmembrane glycoprotein (Herbst, 2004). It is a member of the erbB/human epidermal growth factor receptor family of tyrosine kinases, which also includes erbB2/HER2, erbB3/HER3 and erbB4/HER4 (Troiani et al., 2012). EGFR is found not only in the plasma membrane, but its expression levels are also high in the nucleus, endosomes, lysosomes, and mitochondria (H. Li, You, Xie, Pan, & Han, 2017).

The EGFR signaling pathway is of extreme importance in mammalian cells, having roles in growth, survival, proliferation, and differentiation of cells (Oda, Matsuoka, Funahashi, & Kitano, 2005). It is also overexpressed in a variety of cancers: EGFR is overexpressed in 50–80% of non-small cell lung cancers and ErbB2 and ErbB3 are overexpressed in 25–30% and 63% of breast cancers, respectively (Scharadin et al., 2017). It is also related to increasing resistance to chemotherapy and radiation therapy of tumor cells (Herbst, 2004), and is overexpressed in cancers of very poor prognosis such as pancreatic cancer (Troiani et al., 2012) making it a critical drug target, being its inhibition of particular interest.

Currently, therapies directed at EGFR are included in two general categories: monoclonal antibodies that target the extracellular domain and small molecule tyrosine kinase inhibitors that show effectiveness but eventually lead to resistance (Scharadin et al., 2017) which justifies that the search for new potential inhibitors is continually necessary.

Because of its high prevalence in a number of pathways both healthy and pathogenic that make it a potential drug target of such importance and of the development of resistance, the research for compounds with bioactivity on this target has been great and continuous since the discovery of its role in cancer in the 1980's (Vastag, 2005). This has led to the existence of vast literature on the subject which makes data on known inhibitors freely available (Singh et al., 2015).

All the previously mentioned characteristics, mainly the importance as a drug target, the ever-increasing need for new inhibitors driven by resistance, and the availability of data makes EGFR a great target for virtual screening.

2.4. Data Mining and Machine Learning Algorithms Theoretical Framework

The term Data Mining (DM) was initially a derogatory term that meant the act of searching for an insight that was not supported by the data (Leskovec, Rajaraman, & Ullman, 2011). With the increase of readily available data both in quantity and size, it has taken a positive meaning and can be described as a process to discover patterns and relationships in data that can bring previously unknown insights and allow the making of valid predictions (Edelstein, 1999). It results from the crossing of several fields including Database Management, Artificial Intelligence, Machine Learning, Pattern Recognition, and Data Visualization (Friedman & Friedman, 1997). Data Mining has applications in any industry. These applications include customer segmentation and targeting, credit scoring, fraud detection and drug effect identification in drug trials ("Data Mining From A to Z," n.d.).

Some authors consider Data Mining and Machine Learning (ML) as synonyms and Data Mining does use algorithms from ML in its processes of Knowledge discovery (Leskovec et al., 2011). But Machine Learning can be said to be the discipline that aims at making computers modify or adapt their actions (which can be making predictions or others) so that these actions become more accurate in terms of the goal (Marsland, 2015) using metrics that evaluate this adaptation to guide the process.

When it comes to the kind of tasks related to Data Science, we can say that Machine Learning refers to the creation and use of models that are learned from Data, which will typically have as a goal the prediction of a certain outcome (Grus, 2015).

In the last decade, the multidisciplinary nature of Machine Learning has become apparent. Concepts from Neuroscience and Biology, Statistics, Mathematics and Physics have all contributed to the development of processes to make computers learn (Marsland, 2015), which is apparent in algorithms

such as Artificial Neural Networks (Rumelhart, Widrow, & Lehr, 1994), and Genetic algorithms (Koza, 1992).

2.4.1. Types of Data Mining Problems

2.4.1.1. Regression/Classification Problems

Data mining problems can be classified as either Regression or Classification problems. Both the input and predicted variables used to develop a model can be either quantitative or categorical. In a general sense, we can say that a problem that aims to predict a quantitative value is a Regression problem while problems that aim to assign a record to a certain category are Classification problems (James, Witten, Hastie, & Tibshirani, 2013).

2.4.1.2. Types of Learning

Data Mining problems can also be classified in terms of the amount and type of supervision they get during training. These include:

- **Supervised Learning:** A training set of records with their correct responses (targets or labels) is provided (Marsland, 2015). In this case, the aim is to fit a model that relates the predictors with the response, in order to be able to predict the response for new observations (prediction) and/or to better understand the relationship between the response and the predictors (inference) (James et al., 2013).

- **Unsupervised Learning:** On the other hand, in unsupervised learning, the training dataset does not include labels for a target variable, and in this case, the aim is to find relationships between the variables or between the observations (James et al., 2013).

- **Semi-supervised Learning:** In this kind of learning there is a large amount of unlabeled data and a smaller amount of labeled data. Most semi-supervised systems consist of combinations of supervised and unsupervised algorithms (Géron, 2017).

- **Evolutionary Learning:** Learning systems inspired by biological evolution using the concept of fitness as a measure of how good is the current solution (Marsland, 2015).

- **Reinforcement Learning:** In reinforcement learning the algorithm is told when the answer is wrong but not how to get the right one. In this case, the algorithm or agent can observe the environment and experiment with different solutions that get rewarded, with the agent having to figure out which is the best policy to increase these rewards (Géron, 2017; Marsland, 2015).

In this study, the problem at hand is a Classification problem with supervised learning since the objective is to classify the compounds in PubChem's database or others as either active or inactive and the algorithms used will be provided with the correct category for the training examples.

2.4.2. Theoretical background of the used algorithms

2.4.2.1. Decision Trees

A simple way to describe Decision Trees is that they are a way of representing a set of rules that when applied to an observation can lead to a class or value (Edelstein, 1999). They are composed of a root node, decision nodes and leaf nodes that are connected by branches:

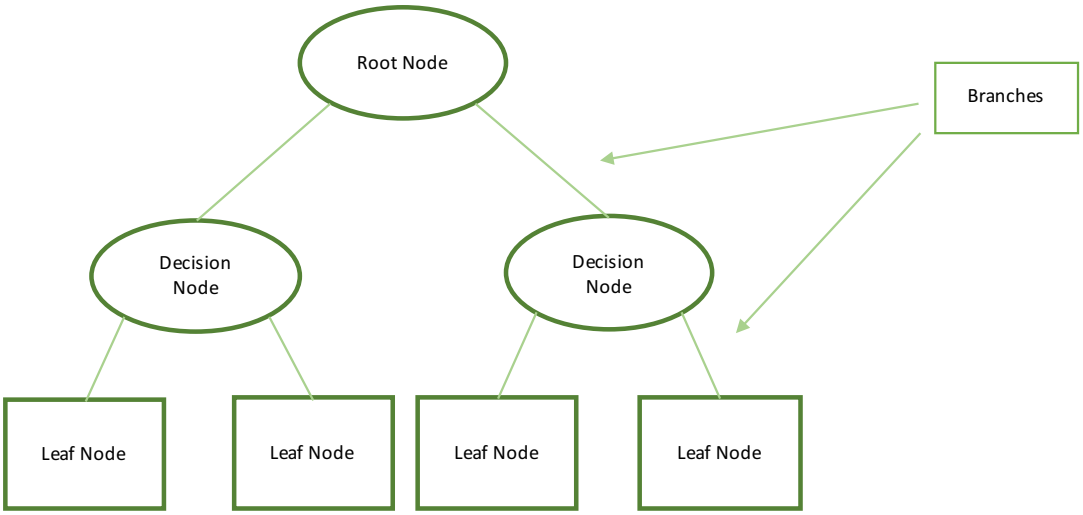

Figure 2 - Decision Tree Structure

At the root (the first split) and at the decision nodes, each attribute is evaluated according to a rule which is applied in each branch until a leaf node is reached where there are no more attributes to evaluate on (Daniel T. Larose, 2015). The greater the purity of the leaf nodes and the distance between them the better (Edelstein, 1999).

Decision Trees have the advantages of being simple and allowing easy interpretation, but usually don't perform as well as other more complex algorithms in terms of accuracy (James et al., 2013) and overfit to the training set very easily which means less generalization capacity (Grus, 2015).

In the process of building decision trees, it is necessary to determine which questions are being asked in each decision node and in what order (Grus, 2015). To achieve this goal, it is necessary to have measures of purity of the nodes before and after the partition is applied and to calculate that difference which will be the measure of how much information will be gained by applying a certain partition. The two most used measures for this purpose are Entropy and Gini Impurity (Marsland, 2015):

- Entropy of a set of probabilities p_i:

$$Entropy = -\sum_i p_i \log_2 p_i \quad (1)$$

- Gini Impurity for a particular feature *k*:

$$G_k = 1 - \sum_{i=1}^{c} N(i)^2 , \quad (2)$$

where c is the total number of classes and N(i) is the fraction of records that belong to class i.

Trees that are allowed to grow indefinitely will overfit to the training data. In order to avoid this, stopping rules must be applied. Common stopping rules are simply limiting the maximum depth that the tree can reach or establishing a lower limit to the number of records in a node. Alternatively, it is possible to prune the tree, where the tree is allowed to grow to full size and then is pruned back to the smallest size that does not compromise accuracy. (Edelstein, 1999)

There are different algorithms that can be used to build a tree that include ID3, C4.5, C5.0, and CART. In this work, the algorithm used is CART.

The CART algorithm produces binary trees, that is, for each feature, it splits the training set into two subsets based on a certain threshold value for that feature. The feature and threshold used are chosen so that the purest subsets are obtained. It follows these steps recursively until the stopping condition is met. CART is a greedy algorithm which means that the optimum splitting choice is made at each level without checking if it will lead to the purest subsets in the levels below. This usually leads to a good solution but doesn't guaranty an optimal one (Géron, 2017).

2.4.2.2. Naïve Bayes

Studies have found the Naïve Bayes classifier to have comparable performance to Decision Trees and some Neural Network classifiers (J. Han, Kamber, & Pei, 2012). It is called Naïve Bayes because it assumes that the observation variables are independent of each other which will most of the times not be true. This classifier is based on Bayes' theorem that states that (Marsland, 2015):

$$P(H|X) = \frac{P(X|H)P(H)}{P(X)}, \tag{3}$$

where $P(H|X)$ is the conditional probability of H given X, $P(X|H)$ is the conditional probability of X given H and $P(H)$ and $P(X)$ are the *a priori* probabilities of H and X, respectively. For the purpose of classifiers, we can consider $P(H)$ to be the probability of the hypothesis that the observation X belongs to a certain class, and $P(X)$ the probability that an observation X is equal to a certain vector of variables.

With the use of the assumption of independence of the variables in the dataset, we come to a simplified equation that states that the probability of an observation X_j being equal to a certain vector of variables given that it belongs to class C_i ($P(X_j|C_i) = P(X_j^1, X_j^2, \dots X_j^n|C_i)$, where the superscripts of X represent the index of the variables of the vector), is equal to the product of the individual probabilities:

$$\prod_K P(X_j^k = a_k|C_i) = P(X_j^1 = a_1|C_i) \times P(X_j^2 = a_2|C_i) \times \dots \times P(X_j^n = a_n|C_i), \tag{4}$$

and the classifier will select the class C_i for which the following computation is the maximum:

$$P(C_i) \prod_k P(X_j^k = a_k|C_i). \tag{5}$$

Although this computation is the result of an obviously incorrect assumption, several empirical studies

show that Bayesian classifiers perform well and are comparable to other more complex algorithms (J. Han et al., 2012) as the assumption made tends to not hurt classification performance (Foster & Fawcett, 2013).

This classifier is very efficient in terms of used storage space and computational time and it is also a natural "incremental learner" as it can update its model one example at a time without having to reprocess all past training examples (Foster & Fawcett, 2013).

In theory, Bayesian classifiers will have the minimum error rate in comparison to other models which not always happens in practice due to the inaccuracies resulting from the assumptions made (J. Han et al., 2012). Nonetheless, it is a very commonly used classifier to serve as a baseline to which other models are compared (Foster & Fawcett, 2013).

2.4.2.3. Logistic Regression

Linear Regression aims to approximate the relationship that exists between a set of variables and a continuous response, but when the response is categorical Linear Regression is not applicable. However, an analogous method can be used, Logistic Regression (Daniel T. Larose, 2015). It is mostly used to predict binary variables but can also be applied to the prediction of multi-class variables (Edelstein, 1999).

In a classification problem, we want the examples that are further away from the boundary between classes to have a higher probability of belonging to that class. The problem of using Linear Regression is that the distance from the border can range from $-\infty$ to $+\infty$ while the probabilities should be in the range zero to one (Foster & Fawcett, 2013).

Since the target variable is discrete and it is not possible to directly model using linear regression, instead of predicting if the event itself will happen the logistic model predicts the logarithm of the odds of its occurrence (Edelstein, 1999).

To meet the objective of getting probabilities between zero and one we can use the logistic function (James et al., 2013):

$$p(X) = \frac{e^{\beta_0+\beta_1 X}}{1 + e^{\beta_0+\beta_1 X}}, \tag{6}$$

where the β_0 and β_1, represent the coefficients for a single predictor X.

With some manipulation of the formula we arrive at:

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X, \tag{7}$$

with the left side being the *log-odds* or *logit* that is linear to X.

For a regression with multiple predictors the previous equation can be generalized to:

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p. \tag{8}$$

To estimate the coefficients in the logistic regression model, the *maximum likelihood* method is used, where the coefficients sought are the ones that will make the predicted probability of each individual be as close as possible to their actual observed category. This can be formalized in the following equation called a *likelihood function* where the estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ are chosen to maximize the following function (James et al., 2013):

$$l(\beta_0, \beta_1) = \prod_{i:y_i=1} p(x_i) \prod_{i':y_i'=0} \left(1 - p(x_{i'})\right). \tag{9}$$

Logistic Regression is a very powerful modeling tool, but it does assume that the target variable (the log odds) is linear in the coefficients of the predictor variables. Also, the optimization of the model may depend on the choice of the right inputs and their functional relationship to the target variable by the modeler, who would also have to explicitly add terms for any possible interactions which requires skill and experience of the analyst (Edelstein, 1999).

2.4.2.4. *k*-Nearest Neighbors

The *k*-Nearest Neighbors (*k*-NN) algorithm uses the similarity between examples to classify them. It places an example in the class to which most of their neighbors belong to (Edelstein, 1999).

Each of the training records described by *n* attributes represents a point in an *n*-dimensional space. When presented with a new example, the classifier searches the pattern space for the k examples that are closer to it. For this purpose, it is necessary to have a measure of distance between the training records. The most commonly used distance measure is the Euclidean distance that can be represented by (J. Han et al., 2012):

$$dist(X_1, X_2) = \sqrt{\sum_{i=1}^{n} (x_{1i} - x_{2i})^2}\,, \tag{9}$$

where X_1 and X_2 are two points represented by the tuples: $X_1 = (x_{11}, x_{12}, \dots, x_{1n})$ and $X_2 = (x_{21}, x_{22}, \dots, x_{2n})$.

The determination of which class to assign the new example to can be made by voting, that is, assigning the new example to the class where the majority of its *k* neighbors belong to, as previously described, or it can be determined by averaging the values of its *k* nearest neighbors which gives an estimate of the probability of the example belonging to that class. It is also possible to use weighted voting where the most distant neighbors will be assigned a smaller weight than the closer ones (Foster & Fawcett, 2013).

There is no direct answer to what the value *k* should be, although odd numbers are advantageous for majority voting purpose. A too small value of *k* will make the classifier too sensitive to noise while a value that is too large will reduce accuracy as points that are very far from the example will be considered (Marsland, 2015). The most common approach would be determining it experimentally by running the model with several incremental values of *k* and using the *k* that gives the lowest error rate on the testing set (J. Han et al., 2012).

k-NN models have a large computational load because the calculation time increases as the factorial of the total number of points. It is necessary to make a new calculation for each new presented case. On the other hand, they are very easy to understand when there are few predictor variables and are useful for non-standard data types, such as text, as it is only required the existence of an appropriate metric. (Edelstein, 1999)

2.4.2.5. Support Vector Machine

Support Vector Machine (SVM) is an algorithm that can be used to classify both linear and nonlinear data. It works by applying a nonlinear mapping to transform the original dataset into a higher dimension where it searches for the linear optimal separating hyperplane, the *maximum margin hyperplane*. This hyperplane is determined by using support vectors and the margins that they define (J. Han et al., 2012).

Linearly Separable Data

For data that has linearly separable classes, an hyperplane exists that classifies all instances correctly and the maximum margin hyperplane will be the one that achieves this and gives the greatest separation between the classes (J. Han et al., 2012).

A hyperplane separating two classes can be expressed by:

$$0 = w_0 + w_1 a_1 + w_2 a_2, \tag{10}$$

in a case with two attributes a_1 and a_2 and three weights to be determined.

Adjusting this equation to the hyperplanes that define the limits of the margin we get:

$$H_1: w_0 + w_1 a_1 + w_2 a_2 \geq 1 \text{ for } y_i = 1 \tag{11}$$

$$H_2: w_0 + w_1 a_1 + w_2 a_2 \leq 1 \text{ for } y_i = -1 \tag{12}$$

Or, combining the two inequalities:

$$y_i(w_0 + w_1 a_1 + w_2 a_2) \geq 1, \forall_i \tag{13}$$

The instances located on the hyperplanes that define the margins are called Support Vectors, and for each class, there is always at least one. They are located at the same distance from the maximum margin hyperplane and are the most important in terms of the classification, since after being defined all other instances could be removed and the model would still be the same (Witten & Frank, 2005).

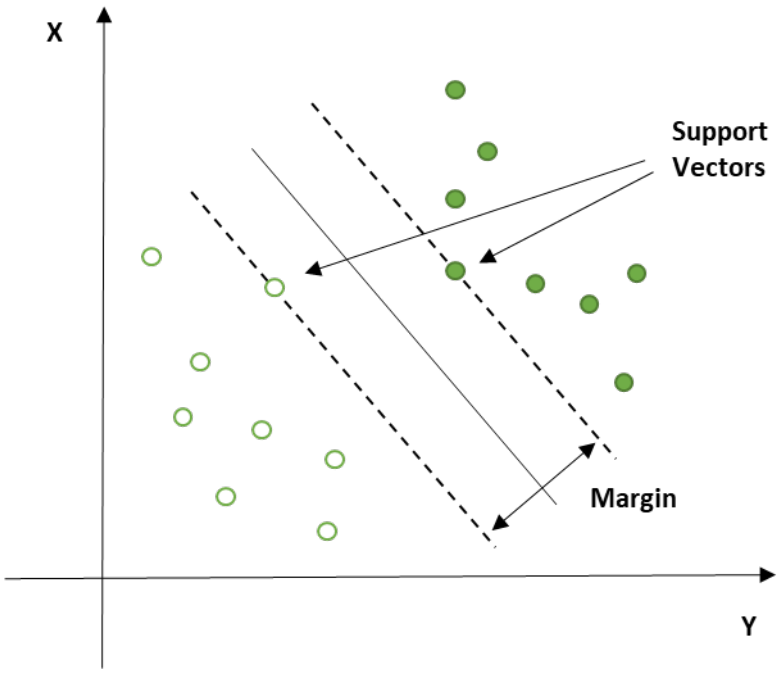

Figure 3 - Representation of Maximum Marginal Hyperplane

The maximum margin hyperplane equation can be given by:

$$x = b + \sum \alpha_i \, y_i a(i) \cdot \, a \tag{14}$$

where y_i is the class value of training instance $a(i)$, b and α_i are parameters determined by the fitting of the algorithm, and the term $a(i) \cdot \, a$ represents the dot product of a test instance with one of the support vectors, and so we achieve a constrained quadratic optimization problem (Witten & Frank, 2005).

Non-linearly Separable Data

To apply SVMs to nonlinear data, an extension of the linear approach can be used where the original data is transformed into a higher dimensional space by applying real functions ϕ on an input vector, where the algorithm will search for a linear separating hyperplane that corresponds to a nonlinear separating hypersurface in the original space (J. Han et al., 2012).

This computation is very expensive as the dot product involves one multiplication and one addition for each attribute and the total number of attributes in the new space can be immense (Witten & Frank, 2005). However, instead of computing the dot product on the transformed input vectors, it is mathematically equivalent to apply a *kernel* function $K(X_i, X_j)$ to the original input as in:

$$K(X_i, X_j) = \phi(X_i) \cdot \, \phi(X_j), \tag{15}$$

where $\phi(X)$ is the nonlinear mapping function used to transform the training tuples.

As such, the dot products can be substituted by one of the less computationally expensive kernel functions that include the following (J. Han et al., 2012):

- Polynomial kernel of degree h:

$$K(X_i, X_j) = (X_i \cdot X_j + 1)^h \tag{16}$$

- Gaussian radial basis function kernel:

$$K(X_i, X_j) = e^{-\|X_i - X_j\|^2 / 2\sigma^2} \tag{17}$$

- Sigmoid kernel:

$$K(X_i, X_j) = \tanh(\kappa X_i \cdot X_j - \delta) \tag{18}$$

Support Vector Machines are very useful as they produce very accurate classifiers but are quite slow when compared to other algorithms (Witten & Frank, 2005) and, as such, it is a major research goal to make SMVs faster both in the training and testing phases, so that they can be more easily applied to very large datasets (J. Han et al., 2012).

2.4.2.6. Neural Networks

Artificial Neural Networks are predictive models inspired by the way the brain works. The brain is composed of neurons wired together where each neuron receives inputs from other neurons and (according to the activation threshold) either fires and produces a response to other neurons or doesn't (Grus, 2015).

Neural Networks are algorithms that are efficient in the modeling of large and complex problems and that may be used both in classification and regression problems (Edelstein, 1999). They were first introduced in the 60's, and since then there have been several waves of new interest towards them (Géron, 2017).

The simplest implementation of a Neural Network is a Perceptron. It is composed of an input layer with one node for each input variable and an output layer where each node is connected to all nodes in the input layer and, usually, a bias feature. The perceptron calculates a weighted sum of the input vector and then applies a step function to the result of the sum and outputs that result. The most commonly used step function is the Heaviside step function (Géron, 2017):

$$Heaviside\ (z) = \begin{cases} 0\ if\ z\ < 0 \\ 1\ if\ z\ \geq 0 \end{cases} \tag{19}$$

The Perceptron is trained by making a prediction for each example and, after comparing the result with the target, increasing the weights from the inputs that would have contributed to the right prediction. Perceptrons are capable of solving simple linear binary classification problems but are unable to solve some other trivial problems such as the XOR problem (Géron, 2017).

However, some limitations of the Perceptron can be eliminated by stacking multiple Perceptrons giving rise to what is called a Multilayer Perceptron (Géron, 2017):

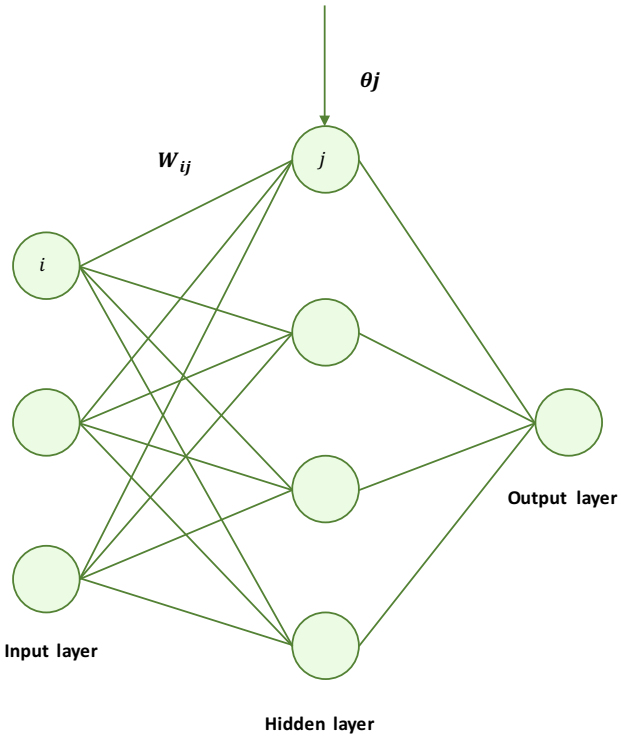

Figure 4 - Simplified representation of Multilayer Perceptron, where W_{ij} is the weight between units i and j and θ_j is the bias of unit j

In the case of the Multilayer Perceptron, one or more hidden layers are located between the input and the output layers, where each node is connected to all the nodes in the previous layer and calculates the weighted sum of the inputs. Each output unit applies a nonlinear activation function to the result of its weighted input. The number of hidden layers and the number of neurons in each layer define the topology of the network. There are no simple rules to determine the best topology for a particular problem and the approach to determine it is experimental (J. Han et al., 2012).

The training of Multilayer Perceptrons is made by the Backpropagation algorithm that consists of several steps that are described next (J. Han et al., 2012):

1. The first step is the initialization of the weights which are small numbers randomly selected.

2. Forward propagation: After the inputs are passed through the input layer, the input and output of each unit in the hidden layer are computed. The net input of the hidden and output layers is computed as a linear combination of its inputs

$$I_j = \sum_i w_{ij} O_i + \theta_j, \tag{20}$$

where I_j is the net input, O_i is the output from the previous unit, w_{ij} is the weight of the connection between the previous unit i and the current unit j, and θ_j is the bias of the current unit j.

3. Application of the activation function: Each unit in the hidden and output layers applies the activation function to its net input. Most commonly the logistic or sigmoid functions are used, and this step allows the algorithm to classify nonlinear data. It is also referred to as a smashing function as it smashes a wide range input into values between 0 and 1. This produces the output of the unit and in the case of the output layer, gives the prediction of the network.

4. Error backpropagation: The goal of the backpropagation of the error is to update both the weights and the biases to reflect the error made in the network's prediction. The error of a unit in the output layer is given by:

$$Err_j = O_j(1 - O_j)(T_j - O_j), \tag{21}$$

where O_j is the actual output of unit j and T_j is the target value for the particular example. The error of a hidden unit j is given by:

$$Err_j = O_j(1 - O_j)\sum_k Err_k w_{jk}\,, \tag{22}$$

where w_{jk} is the weight of the connection from unit j to unit k in the next layer and Err_k is the error of unit k.

5. Update weights and biases. Weights are updated by applying:

$$\Delta w_{ij} = (l) Err_j O_i \tag{23}$$

$$w_{ij} = w_{ij} + \Delta w_{ij} \tag{24}$$

where l is the learning rate that is used to avoid the algorithm getting stuck on a local optimum. Biases are updated by applying:

$$\Delta\theta_j = (l) Err_j \tag{25}$$

$$\theta_j = \theta_j + \Delta\theta_j \tag{26}$$

In this case weights are being updated for each example which is called case updating, but they can also be stored in a variable and be updated after all the examples in the training set have been presented to the network, which is called epoch updating, as one iteration through the training set is named an epoch.

6. Termination of the algorithm: The training stops when a termination condition is met. This can be either when the weights of the previous epoch or the error of the network are below a certain specified value, or when a specified number of epochs have been run.

Although Neural Networks are versatile, powerful, scalable and appropriate for large and highly complex tasks (Géron, 2017), they have some disadvantages. These include their tendency to overfit, the fact that they are nearly impossible to interpret, being what is called a "black box", and that they

have a long training time, although the predictions are provided quickly (Edelstein, 1999).

2.4.2.7. Ensemble methods

Ensemble methods are based on a principle similar to that of the *wisdom of the crowd* which is a phenomenon where the average answer of a large number of people to a certain question is often better than the single answer of an expert (Géron, 2017).

An ensemble model for classification is a composite model that results from combining different classifiers. Ensemble classifiers tend to perform better than their composing models individually (J. Han et al., 2012).

Voting Classifiers

The simplest form of Ensemble methods is the Voting method where the predictions for the individual classifiers are aggregated, and the prediction with the majority of votes from the composing classifiers is the one chosen.

This strategy, albeit its simplicity, will generally outperform the best classifier in the ensemble and tends to achieve good performance even if all the composing classifiers are weak learners, that is, only slightly better than random guessing. This happens because of the *law of large numbers*, that states that over a large number of trials the average result will be close to the expected value. In the case of classifiers, this translates to the fact that if all classifiers have an accuracy of above 50%, as their number increases, the accuracy of the combined models will also increase. (Géron, 2017)

Bagging

Bagging is a method that also uses majority vote to get the final prediction of the ensemble, but it introduces the variety of classifiers in a different way than described previously. The several individual classifiers are built using the same algorithm but on different samples taken from the original dataset with replacement. This implies that a certain sample will likely exclude some examples and duplicate others from the original dataset. The aim of this process is to decrease a source of error for individual classifiers that arises from the use of a particular training set that is inevitably finite and not completely representative of the whole population. That is, it intends to diminish the variance component of the error of the classifier in the bias-variance trade-off. Because of this, Bagging is usually most useful when used with algorithms that are by themselves unstable, such as Decision Trees. (Witten & Frank, 2005)

Boosting

In Boosting, weights are assigned to each record and, after a classifier is trained, these weights are updated so that the following classifier will give more importance to the records that were misclassified. In the end, the ensemble chooses the correct prediction based on voting with each classifier's vote weight being a function of its accuracy. (J. Han et al., 2012)

AdaBoost

The most commonly used algorithm to perform boosting is AdaBoost (from adaptive boosting). It can be described by the following (J. Han et al., 2012):

Considering we have a dataset D, of d labeled tuples (X_1, y_1), (X_2, y_2), …, (X_d, y_d) where y_i is the class label of X_i, AdaBoost will firstly assign a weight of $1/d$ to each instance. Following this first step, k rounds are performed for k generated classifiers. On round i a sample D_i, of size d is generated with replacement to form the training set so each tuple may appear more than once and their probability of selection is dependent on its assigned weight. A classifier, M_i, is generated and its error is calculated using D_i as a test set. This error is calculated using:

$$error(M_i) = \sum_{j=1}^{d} w_j \times err(X_j), \tag{27}$$

where $err(X_j)$ is the misclassification error of instance X_j and is equal to 1 if the instance was misclassified and 0 otherwise. The weights for each correctly classified tuple are then updated multiplying by:

$$\frac{error(M_i)}{1 - error(M_i)}. \tag{28}$$

After this step, all weights are normalized, including those of the misclassified tuples, which results in an increase of the weights of the misclassified tuples and a decrease of the weights of the correctly classified ones.

At the end of the k rounds, the final prediction is made by voting, as previously mentioned, and the weight given to each classifier's vote is given by:

$$\log \frac{1 - error(M_i)}{error(M_i)}. \tag{29}$$

For each class, the weights of each classifier that voted for it as the correct one are summed, and the class with the highest sum is the prediction made by the ensemble.

In comparison with Bagging, Boosting tends to achieve higher accuracy but has a higher risk of overfitting (J. Han et al., 2012).

Random Forest

Random Forests are an ensemble of Decision Trees generally trained using Bagging (Géron, 2017). If so, they are built using Bagging in combination with random attribute selection at each node to determine the split. With a training set D, of size d, to generate k Decision Trees, for each iteration $i (i = 1,2, \dots, k)$, a sample of size d is generated with replacement. At each node, an F number of attributes is randomly selected as candidates for the split with F being much smaller than the number of total attributes. The used algorithm to grow the trees is CART, and the trees are not pruned. Random Forest is comparable to AdaBoost in terms of accuracy but is more robust to errors and outliers (J. Han et al., 2012).

Extremely Randomized Trees

Extremely Randomized Trees are similar to Random Forests, but besides choosing a random number of features to be considered for splitting, the threshold used for each feature is also random instead of searching for the best possible one. This will trade bias for variance, that is, will increase the error

that arises from differences in the training dataset and decrease the error that arises from the assumptions made by the model. Since the choice of threshold is random, Extremely Randomized Trees train much faster than normal Random Forest but is not possible to tell beforehand which will perform better and both need to be applied and compared to determine the best one. (Géron, 2017)

3. METHODOLOGY

The process of defining the methodology for this work was an iterative and recursive one as it is for most Data Mining/Machine Learning projects. Its major parts involved the experimentation and selection of the best tools, the gathering and treatment of the data, and the choice of algorithms to use, of their parameters' values and of how to evaluate them, each step consisting of a process of trial and error to achieve the best results.

3.1. USED TOOLS

The tool of choice for this work was the Python ("Welcome to Python.org," n.d.) programming language and its data analysis and data science libraries, mainly Numpy ("NumPy — NumPy," n.d.), Pandas ("Python Data Analysis Library — pandas: Python Data Analysis Library," n.d.) and Scikit-learn ("scikit-learn: machine learning in Python — scikit-learn 0.19.0 documentation," n.d.). After considering other tools, the choice of using Python was due to several reasons. This language and its libraries are one the most popular tools for data science because they have several advantages. They are suited to the fast and easy handling of large amounts of data, they are intuitive to use and easy to learn. Particularly in the case of Scikit-learn, the computations are fast, the models are easy to implement while still offering flexibility and user customization, and most processes necessary in a data science project are covered.

3.2. DATA GATHERING AND TREATMENT

As previously mentioned, the data that was used in this study came from the PubChem database, namely from the Bioassay database where the information about all the assays that have been uploaded is gathered, and datasets are available for each bioassay. These datasets contain the information about the type of assay or source, which could be a primary screening, a confirmatory assay or a literature based dataset, the tested compounds and their classification as active or inactive.

In the case of EGFR, the data available was quite extensive in comparison to some other compounds and a choice was made to use the dataset that gathered all the available information for this target as almost all of the assays were confirmatory or the information was collected from the literature. This gives confidence in the quality of the results and that the number of false positives will be proportionally low compared to the case of primary screen assays where a large number of compounds is tested and the IC_{50} threshold for the compound to be considered active is higher, while for confirmatory assays the threshold is tighter to confirm the findings of the primary screens. The total dataset was composed of 13116 compounds of which 4692 are labeled as active.

The datasets are available for download in different formats and contain information about the assays and the compounds that were tested, together with their classification as active or inactive. For this study, the only information that was kept was the CIDs for the compounds (their PubChem identifiers that allow the gathering of information about them from the database) and their classification.

In this work, the chemical descriptors that were chosen to characterize the compounds and be the features used in the model building were PubChem's own particular descriptors named Substructure Fingerprints that consist of a binary string of 881 bits where each bit codes the presence or absence of

a chemical structure. The coding for each bit can be found in the annexes section of this work. This fingerprint is available as a base64 encoded string.

To extract these fingerprints, the CIDs were used and were gathered from PubChem using a Python library, PubChemPy ("PubChemPy documentation — PubChemPy 1.0.4 documentation," n.d.), that uses PubChem's API to allow the user to get only the information needed programmatically.

As such, the process to obtain the final dataset consisted of the download and treatment of the dataset, the import of the fingerprints in base64, the conversion of those fingerprints to binary, removal of the padding, inputting each of the resulting 881 bits to a separate feature, and assigning the value one to the active compounds and zero to the inactive ones, so that the resulting dataset consisted of 881 binary input variables, one binary target variable, and 13116 records.

This resulting dataset was then divided into the training and testing sets in a proportion of 70 and 30% respectively, as will be explained in the section about model evaluation.

3.3. Choice of Algorithms

Since one of the objectives of this study was the comparison of several algorithms for their performance on the problem at hand, the option followed was to use a wide diversity of classifiers known to be able to give good results, that could be applied and evaluated using the same methodology and tool, which meant the implementations of Scikit-learn. As such, the classifiers that were applied in this work were:

- Decision Tree with CART algorithm,
- Gaussian Naïve Bayes, where the likelihood of the features is assumed to be Gaussian and Bernoulli Naïve Bayes that assumes the data is distributed according to multivariate Bernoulli distributions and as such, requires samples to be represented as binary-valued feature vectors which is the case for the used dataset ("scikit-learn: machine learning in Python — scikit-learn 0.19.0 documentation," n.d.),
- Logistic Regression,
- K-Nearest Neighbors,
- Support Vector Machine,
- Neural Network – Multilayer Perceptron,
- Ensemble Methods:
    - Random Forest,
    - Extremely Randomized Tree,
    - Bagging,
    - Boosting with AdaBoost algorithm,
    - Voting.

3.4. Feature Selection

It is known that feature selection is an important part of most Data Mining or Machine Learning projects, and its importance is increasing in recent times as large amounts of data become easily available, and data scientists are faced with datasets that often contain variables that are either

irrelevant to model the problem being studied or are redundant, that is, convey information that is already encoded in other variables (Fernandez-Lozano et al., 2013).

There are several techniques that can be used to perform feature reduction that have the purpose of finding an optimal subset of features that are needed to find the solution for a problem. The benefits of applying these techniques include the need for a smaller number of samples to obtain an optimal result, less running time (Fernandez-Lozano et al., 2013) and better performance of the model by diminishing the risk of overfitting (Grus, 2015).

Considering these advantages, different techniques were tried and applied to the data that was used to build the predictive models. The experimented techniques were the Scikit-learn implementations of:

- Feature removal based on low variance – this technique eliminates all the features that don't meet a certain variance threshold, that is, that have the same value for a certain proportion of the examples;
- Principal Component Analysis – PCA is an unsupervised approach that uses rotation methods to group the variables in such a way that the total variance explained is maximum, resulting in a reduced set of variables (the Principal Components) that are linear combinations of the original ones, ordered by their variance (Das, Chattopadhyay, & Gupta, 2016).
- Univariate feature selection – This technique uses statistic testing to select the k features with the strongest relationship with the target variable. In this case, the test used was the $\varkappa^2$ test.
- Linear models with regularization – These methods use coefficients from linear models to select the best features since, if the variables are on the same scale, the most important ones for the model will have the higher coefficients, and the ones uncorrelated to the target variable will have coefficients close to zero. The use of regularization adds a penalty to the loss function to avoid overfitting.
- Tree-based feature selection – these models use the measure of purity increase resulting from the tree models' partitions to select the best features.

After the experimentation with these different techniques, it was verified that the obtained results, depending on the model, were either similar or worse than was the case with the use of the total number of features. As such, and because the difference in computation times was not very significant, the option taken was to use the full set of features.

These results may be related to the nature of the dataset where the features all encode chemical structures which are the base of molecular target activity which is what is being modeled.

3.5. TREATMENT OF IMBALANCED DATA

HTS data is usually characterized by highly imbalanced data, that is, data that has a much higher proportion of one class than another, as for each test there will usually be a much smaller number of active compounds than inactive compounds. This type of datasets may lead to a weakened performance of models as it may skew the accuracy (Q. Li, Wang, & Bryant, 2009). If the records of a dataset are composed almost only of one of the classes, the model can have a high accuracy by ignoring one of the classes which might defeat the purpose of the project.

In this work, the dataset used was not highly imbalanced as it was mostly based on confirmatory assays. Nevertheless, a more balanced dataset was obtained. Both undersampling and oversampling techniques were tried to achieve this objective.

To undersample the dataset, a simple method of randomly removing a proportion of the non-active records was used, but the best results were achieved by oversampling the minority class, the active compounds, using the SMOTE (Synthetic Minority Over Sampling) method.

The SMOTE method creates synthetic records of the minority class by selecting for each instance of the minority class k instances closest in Euclidean distance, calculating the difference between the instance and its neighbors, multiplying that value by a random number and adding the result to the variables of the minority record. The value of k is dependent on the intended ratio between classes. (Ramezankhani et al., 2016)

For this study, different ratios were tried, and although the differences in results with models using the dataset before and after constructing a more balanced dataset were not very large, the best results were achieved using the SMOTE method of oversampling to create a dataset with a ratio of 0.7 between the active and inactive classes. The SMOTE method was applied after the separation of the dataset in training and testing set so as to keep a set where the ratio was the initial one and where the generalization capacity of the model could be assessed without bias.

3.6. Model Evaluation

To choose the best model to solve a prediction problem it is necessary to have measures of its performance that reflect the different aspects of the quality of the models.

3.6.1. The Confusion Matrix

At the basis of the measures of quality of classification problems is the confusion matrix. It is simply a matrix with the labels of all the possible classes listed both horizontally and vertically, with the predicted classes listed on one orientation and the actual classes listed perpendicularly. So, if we have two classes for the target variable, C_1 and C_2, we would have:

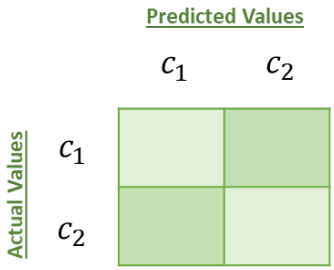

Figure 5 – Confusion Matrix

Where, we would have on the top left corner the number of instances correctly classified as belonging to C_1, on the bottom left corner the number of instances classified as C_1 that actually belong to class C_2, on the top right the number of instances classified as C_2 that actually belong to C_1 and on the bottom right the number of instances correctly classified as C_2.

So, if we would have C_1 as the "negative" class and C_2 as the "positive" class we would have:

TN	FP
FN	TP

Figure 6 – Confusion Matrix for Binary Classification

Where TN, FP, FN, and TP mean true negatives, false positives, false negatives and true positives, respectively.

3.6.2. Accuracy Measures

The most generally used measure to evaluate the quality of a classifier is its accuracy. It provides a way to measure the general predictive capability of the classifier, and its formula is:

$$\frac{TP + TN}{TP + TN + FP + FN} \tag{30}$$

Although accuracy is a valuable indicator of the model's quality, it does not provide a comprehensive view of the model's performance, and other metrics are necessary to make that interpretation. The ones that were used in this study are described below:

$$Sensitivity = \frac{TP}{TP + FN} \tag{31}$$

$$Specificity = \frac{TN}{TN + FP} \tag{32}$$

$$Precision = \frac{TP}{TP + FP} \tag{33}$$

$$F_1 = 2 \cdot \frac{precision.recall}{precision + recall} \tag{34}$$

Sensitivity (or recall as it is named in Scikit-learn) gives a metric to evaluate how well the model "captures" the positive instances in that it is the proportion of the number of correctly predicted positives to the total number of positives while specificity is the same metric applied to negatives. Precision, on the other hand, allows the evaluation of what proportion of the records classified as positive are actually positive. F_1 gives a measure of the balance between precision and recall.

A measure that can be used even in the case of highly imbalanced datasets and gives a good overall measure of the quality of the model is the Mathew's Correlation Coefficient:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{35}$$

3.7. OVERFITTING

Overfitting corresponds to the error of not following the Occam's Razor principle which is a parsimony principle that states that the used model should only have the complexity necessary to model the studied problem and no more. This additional complexity can arise from different factors such as the inclusion of irrelevant components as, for example, using a polynomial of excessive degree, the use of irrelevant predictors or running the learning phase for too long. (Hawkins*, 2003)

This excessive complexity of the models leads to a great problem that is the incapacity of the model to generalize to unseen data which means that the model might have a very high performance on training data and not perform any better than a random classifier on unseen data.

To prevent this problem and correctly assess the generalization capacity of models, different techniques may be applied, and in this study, two approaches were used as to be able to confidently make this assessment. Here, the used techniques were 10-fold Cross Validation and keeping an independent test set that was unaltered and never seen by the model during training.

k-Fold cross validation is used to assess the models' generalization capacity and provide a more accurate metric of the models' performance. It works by dividing the dataset in k folds and train and test the model k times, each time testing on one of the folds while the training is done on the remaining $k-1$ folds all together. The assessment metrics are then obtained by calculating the average of the individual results on each of the folds when used as a testing set. In this case, 10-fold cross validation was chosen since it is the one most commonly used because of its empirically demonstrated good results (J.-H. Kim, 2009).

A scheme for this process is:

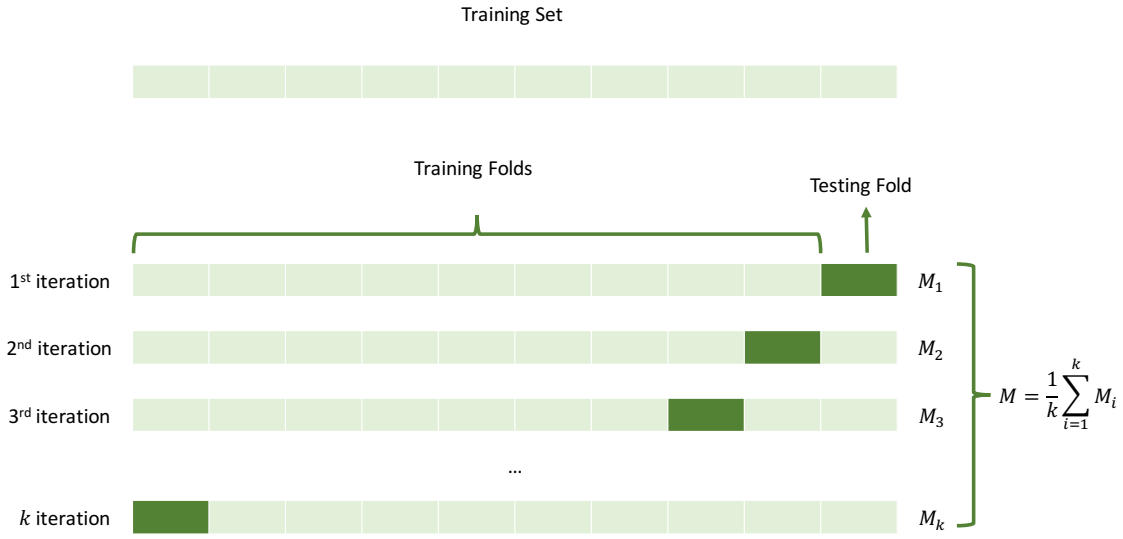

Figure 7 - k -fold Cross Validation

To provide an added level of confidence on the assessment of the performance of the models on unseen data, a test set with 30% of the data was initially split from the original dataset, that was never shown to the model during training, and that was not oversampled and, as such, provides a good estimate of the quality of the models in terms of their generalization capability.

3.8. MODELS' PARAMETER OPTIMIZATION

Most classifiers will have parameters that influence the performance of the model. These parameters can determine the configuration and complexity of the model and, as such, determine its learning and whether it under or overfits the data.

To optimize the choice of these parameters as to get the best model possible from the applied algorithm, the simplest approach would be to iterate the training of the model using different parameters and then compare the results, but there are more automatic methods of making that selection.

In this work, both approaches were used. Firstly, an iterative process of running the learning of the models several times with parameters with a wider range was performed, to then obtain a smaller interval of values that are then subject to the process of automatic search of the optimal value.

The automated part of the search was conducted using an implementation from Scikit-learn of a Grid Search evaluated by cross-validation. This method tests all possible combinations of parameters, evaluating each combination through cross-validation, which avoids the choice of models of unnecessary high complexity that could lead to overfitting, and using a scoring function that is maximized. The user determines the parameter grid, the number of folds to use in the cross validation and the scoring function that by default is accuracy. In this case, the option was to use 10-fold cross validation for the reasons previously mentioned, and recall as the scoring function because of the nature of the problem, considering that the ultimate goal is to screen large databases for the active compounds that usually will be in much smaller number than the non-active ones and, as such, the most important feature of the model will be its ability to identify the largest proportion possible out of the actual positive instances that exist in the database.

3.9. STUDY WORKFLOW

Although above the different parts of the study were presented separately, the work was done continuously and iteratively. Bellow, a workflow of the study is presented, although the order of the work was at times not as linear and more recursive than is presented in the scheme.

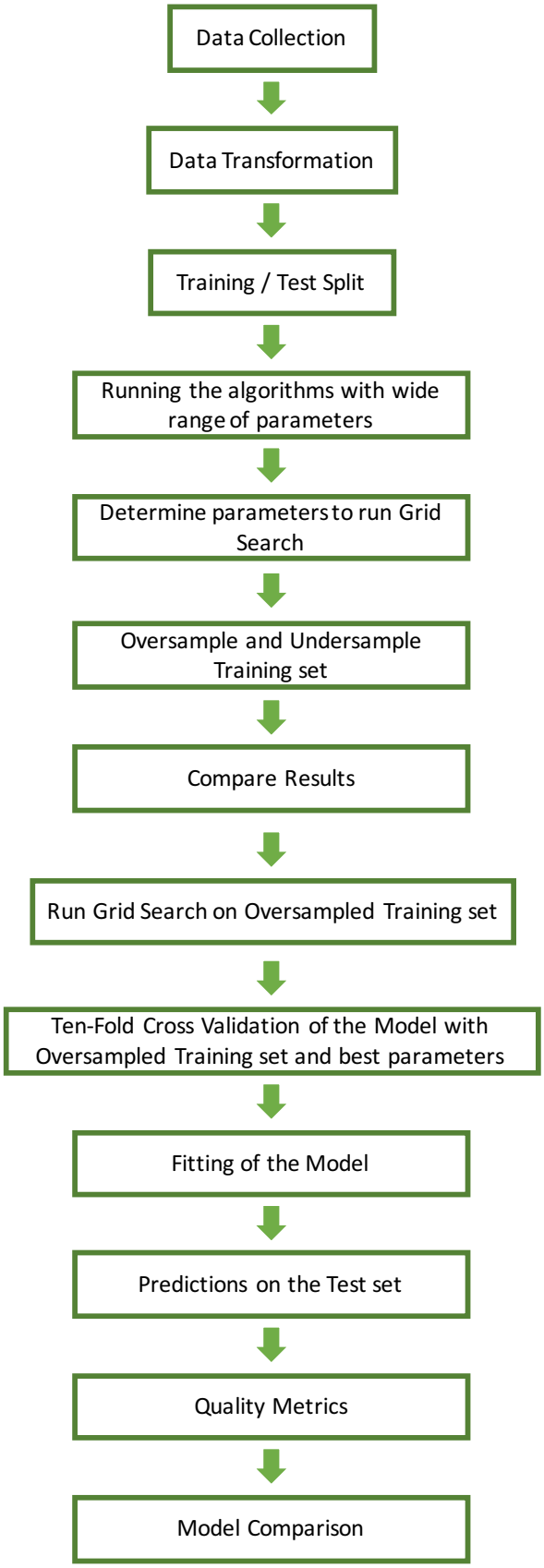

Figure 8 - Study Workflow

4. RESULTS AND DISCUSSION

4.1. RESULTS

4.1.1. Results from Cross Validated Grid Search for Best Parameters

Table 1 - Optimal Parameters Selected by Cross Validated Grid Search

Classifier	Selected Parameters
k-NN	Number of k neighbors = 5
Logistic Regression	C = 10
SVM	Kernel = RBF, C = 100
MLP	Hidden Layer Sizes = 400, Initial Learning Rate = 0.001
Random Forest	Number of Estimators = 25
ERT	Number of Estimators = 15
AdaBoost	Number of Estimators = 1600

4.1.2. Results from ten-fold Cross Validation

Table 2 - Average Quality Metrics from 10-fold Cross Validation

Classifier	Average Accuracy	Average Sensitivity	Average Precision
Gaussian NB	0.55	0.95	0.48
Bernoulli NB	0.70	0.70	0.62
Decision Tree	0.87	0.83	0.84
k-NN	0.87	0.88	0.83
Logistic Regression	0.84	0.82	0.81
SVM Linear Kernel	0.84	0.82	0.81
SVM RBF Kernel	0.89	0.89	0.84
MLP	0.89	0.89	0.85
Random Forest	0.89	0.86	0.87
ERT	0.89	0.86	0.87
AdaBoost	0.84	0.79	0.81
Bagging with DT as base classifer	0.88	0.84	0.87
Bagging with MLP as base classifer	0.89	0.89	0.86
Voting	0.89	0.87	0.87

4.1.3. Result from Test Set

Table 3 - Accuracy, Sensitivity, Specificity, and Precision from test set

Classifier	Accuracy	Sensitivity	Specificity	Precision
Gaussian NB	0.50	0.94	0.25	0.42
Bernoulli NB	0.70	0.69	0.71	0.58
Decision Tree	0.86	0.79	0.90	0.81
k-NN	0.87	0.86	0.87	0.80
Logistic Regression	0.84	0.79	0.87	0.78
SVM Linear Kernel	0.84	0.80	0.86	0.77
SVM RBF Kernel	0.88	0.86	0.89	0.82
MLP	0.87	0.90	0.85	0.78
Random Forest	0.88	0.84	0.90	0.83
ERT	0.87	0.81	0.91	0.83
AdaBoost	0.82	0.73	0.87	0.77
Bagging with DT as base classifer	0.87	0.82	0.90	0.83
Bagging with MLP as base classifer	0.88	0.83	0.91	0.84
Voting	0.88	0.83	0.91	0.84

Table 4 - F1 and MCC from Test Set

Classifier	F1	MCC
Gaussian NB	0.58	0.24
Bernoulli NB	0.63	0.39
Decision Tree	0.80	0.69
k-NN	0.83	0.73
Logistic Regression	0.79	0.66
SVM Linear Kernel	0.78	0.65
SVM RBF Kernel	0.84	0.74
MLP	0.84	0.74
Random Forest	0.83	0.74
ERT	0.82	0.72
AdaBoost	0.75	0.61
Bagging with DT as base classifer	0.82	0.72
Bagging with MLP as base classifer	0.83	0.74
Voting	0.83	0.74

4.2. Discussion

4.2.1. Individual Classifier Discussion

4.2.1.1. Naïve Bayes

For the Naïve Bayes classifiers, there were no parameters to adjust. In this case, we see that the choice of the probability distribution function deeply influences the performance of the classifier with the Gaussian NB having a poorer performance overall, with a very high sensitivity but at the expense of having a very large number of false positives. The Bernoulli NB had a more balanced performance but worse results than the majority of the other classifiers. Both had very small running times which is also an advantage of this classifier.

4.2.1.2. Decision Tree

For the Decision Tree classifier, there was no optimization of parameters as the decision made was not to impose a limit on the maximum number of features to consider when looking for the best split nor on the maximum depth of the tree. The classifier had good performance but the sensitivity, that is the most valued metric for this problem, is somewhat low.

4.2.1.3. k-Nearest Neighbors

The optimized parameter in the k-Nearest Neighbors classifier was obviously the number of neighbors to use since the chosen number has great importance as it's the majority of the classes of these neighbors that will determine the class of the instance being categorized. After an initial experimentation with small and high values, a range between 5 and 15 was determined as a good search space and the value of 5 was selected by the grid search method. The model's quality was good since it obtained high values for accuracy and sensitivity with slightly lower value for precision which is not deemed as important for the present study.

4.2.1.4. Logistic Regression

In the case of the Logistic Regression classifier the optimized parameter was the parameter C, that, as is the case for the SVM algorithm, introduces more regulation and can be increased in case of overfitting of the model. The optimal value determined was 10. This classifier showed lower sensitivity making it not as useful for its objective.

4.2.1.5. Support Vector Machine

The SVM algorithm is the one that took the longest to run the grid search and train, which is a disadvantage and, because of this fact, instead of having a parameter grid with all possible kernel functions and respective parameters, the models with different kernel functions were optimized and trained separately, with the polynomial kernel becoming rapidly apparent to be much worse than the other two and, as such, was not applied. Both the other kernel functions produced models with good performance, but the best SVM model was the one with 'RBF' kernel and C equal to 100. This configuration produced a model with high quality measures overall.

4.2.1.6. Multilayer Perceptron

Optimizing the topology of the Multilayer Perceptron classifier was the most time consuming of all the algorithms because it is possible to create virtually endless combinations of number of hidden layers and number of units in each hidden layer. After several tries, the configurations that seemed to produce the best results were either a single layer with a higher number of units or two layers with a smaller number of units. The final grid consisted of two single hidden layer topologies and one with two hidden layers combined with two different values for the learning rate as this is also an important parameter on the tuning of Multilayer Perceptrons since it influences the algorithm's ability to escape local optimums. The final chosen parameters were a single hidden layer of 400 units and learning rate of 0.001. The model had high values for the quality metrics overall with the Precision slightly lower than the rest of the metrics.

4.2.1.7. Random Forest

The optimized parameter on this classifier was the number of trees in the forest. Following the same approach as previously mentioned, the grid search was conducted using the range between 10 and 30 estimators, and the optimal number determined was 25. This ensemble considerably improved over the use of a single Decision Tree, producing good results.

4.2.1.8. Extremely Random Tree

With this ensemble method, as was also the case with the AdaBoost algorithm, the grid search method would always choose the highest value for the number of estimators but it was verified that from a certain value the results were similar, which defined the upper level of the search space that was a range between 1 and 15, to confirm if there was no smaller value that would produce better results. The resulting model with 15 estimators had good performance but lower sensitivity, considered more important in this study.

4.2.1.9. AdaBoost

As was mentioned for the Extremely Random Tree, the grid search for the AdaBoost apparently would always choose the highest possible value for the number of estimators but in this case the upper limit where there were no more improvements was much higher and the difference in the number of estimators also had to be higher to lead to differences in the results, so the range used for the optimization was between 700 and 1600 with intervals of 100, to confirm the best value that was indeed the highest. This algorithm also produced a model with a smaller value for sensitivity.

4.2.1.10. Bagging

The Bagging ensemble was tested both with the default base estimator, a Decision Tree and with the Multilayer Perceptron. Both produced good results, but the one using MLP as the base estimator produced better ones, although apparently not improving over the MLP classifier by itself while the one using DT did improve over the DT alone, but not as much as the Random Forest.

4.2.1.11. Voting

The voting classifier was built combining several different individual models, namely Logistic Regression, Random Forest, Gaussian Naïve Bayes, k-Nearest Neighbors, Support Vector Machine and

Multilayer Perceptron, since usually ensemble algorithms have better performance with diversified models. This resulting ensemble had good overall quality metrics but didn't seem to particularly improve over the best performing individual models.

4.2.2. Model Comparison

To better understand the differences in the results of the different models, the following graphical representations were constructed:

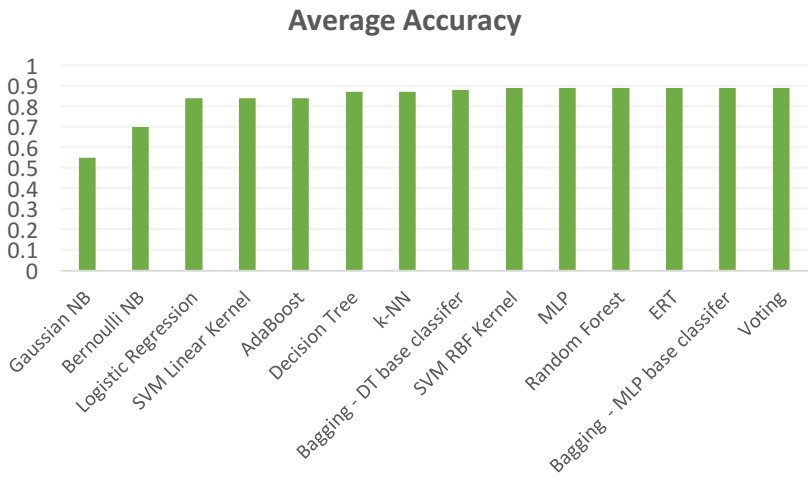

Figure 9 - Average Accuracy for 10-fold Cross Validation

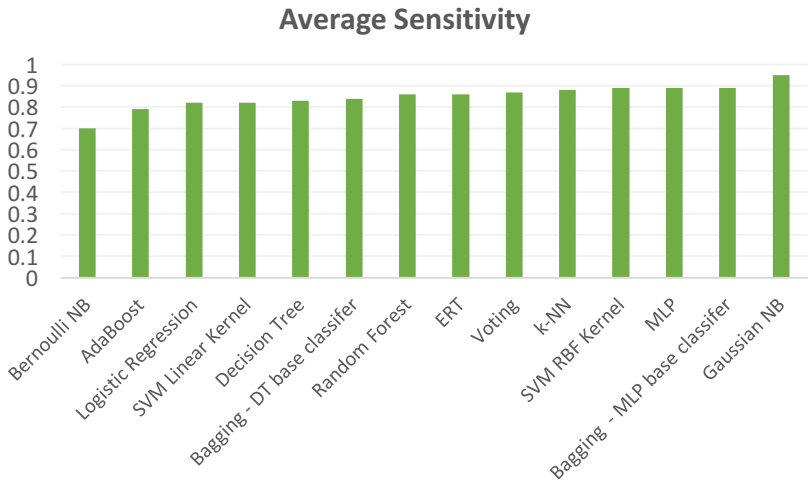

Figure 10 - Average Sensitivity for 10-fold Cross Validation

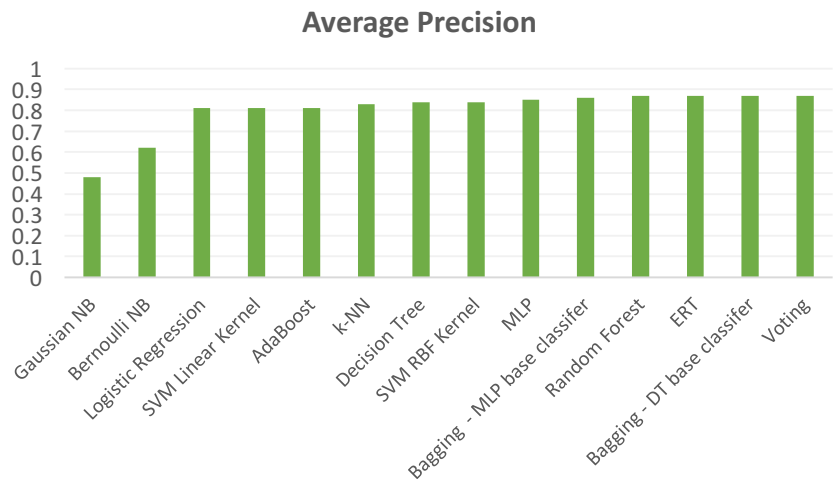

Figure 11 - Average Precision for 10-fold Cross Validation

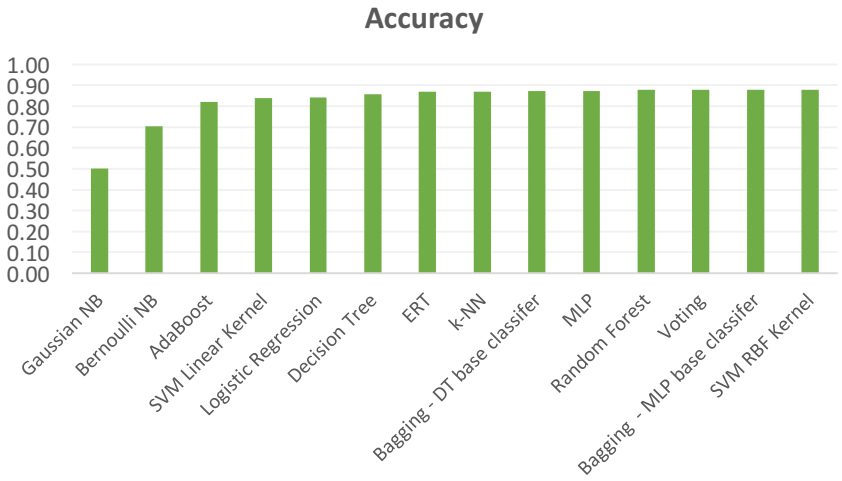

Figure 12 - Accuracy on Test Set

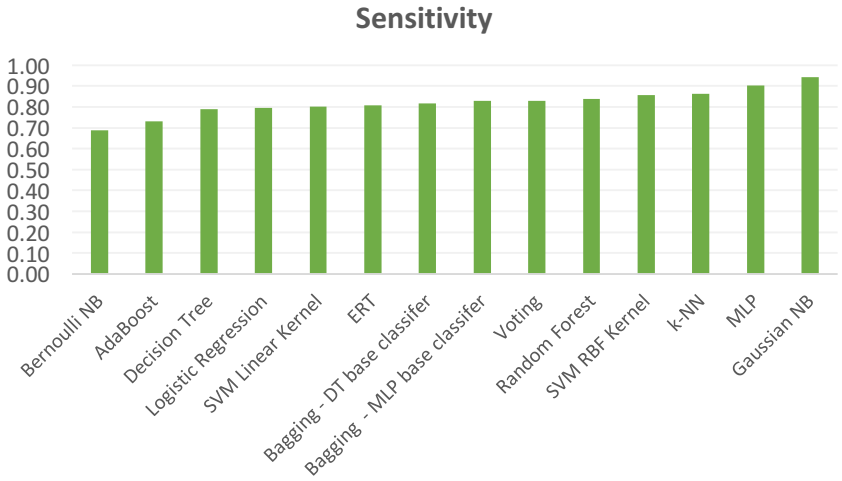

Figure 13 - Sensitivity on Test Set

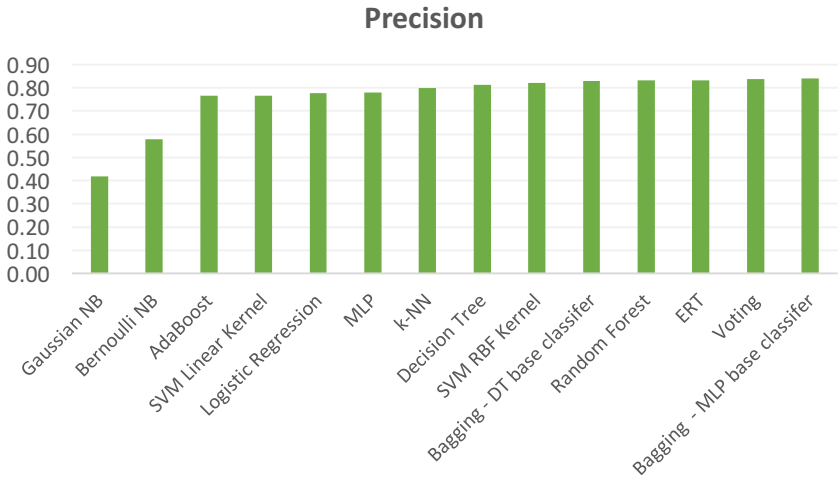

Figure 14 - Precision on Test Set

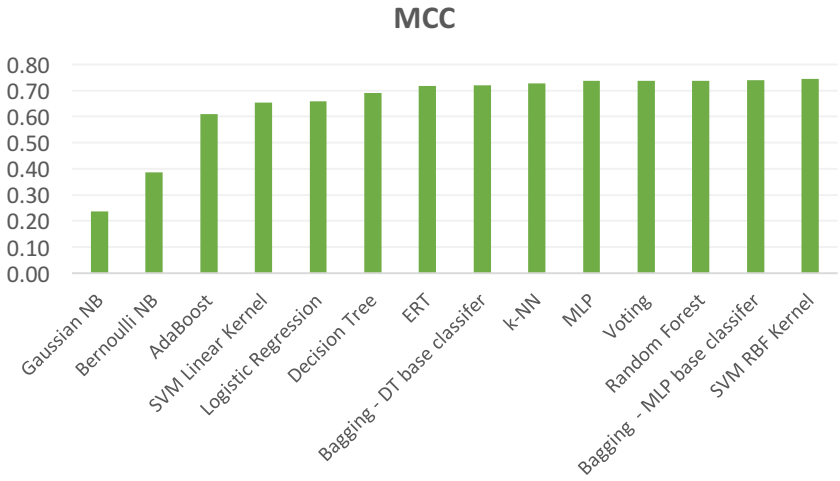

Figure 15 - MCC on Test Set

The charts used were built for the metrics that were considered most important. In particular Sensitivity as the measure of ability to 'capture' the active compounds and Accuracy and MCC as measures of overall quality of the models.

We can see from all the measures that the worst performing classifiers are the Naïve Bayes, the AdaBoost, the Linear SVM, Logistic Regression and Decision Tree while ERT, Bagging with DT base classifier and k-NN are in the mid-range.

As such, in general terms, we can say that the models that performed better were SVM with RBF kernel and MLP from the individual models and Random Forest, Bagging with MLP as the base classifier and Voting from the ensemble models.

Some other considerations can be made, such as the fact that although the Gaussian NB had the highest Sensitivity, the measure considered of most importance, it was not considered a good performer as that came at the cost of a very high number of false positives which would defeat the purpose of the model. On the other hand, we can see that the models that have high sensitivity have

a lower value of precision and vice-versa which is expected, and in this trade-off the ones with higher sensitivity will be preferred for this study, as the ultimate objective is the ability to screen large databases where the number of positives is expected to be quite lower than the number of negatives.

Also, the results obtained show that the differences in performance from the best performing models are not very relevant, with very similar values both in the average measures from the cross validation and in the test set. As such, considering that the ensemble and SVM models take longer to run and that the MLP classifier showed the best results in terms of the sensitivity in the test set and has a smaller time of prediction, this could be considered the best model to be used to screen the entire PubChem database, subsections of it or other databases.

4.2.3. General Discussion

Some considerations can be made taking into account the results of this study. In a general way, we can say that the best models obtained good performances as classifiers of EGFR inhibitors with the MLP classifier, considered the best, having high values for the metrics considered most important, namely accuracy and sensitivity. These were 89% for both the average accuracy and sensitivity for 10-fold CV and 87% and 90%, respectively on the test set and also an MCC of 0.74.

Previous studies using Machine Learning and Data Mining methods to build predictive models of EGFR inhibitors have had results both below (Singh et al., 2015) and above (Kong et al., 2016; Zhao et al., 2017) the ones reported in this study. This could be related with several factors in the methodology such as the chemical descriptors used.

Nevertheless, it is difficult to make a direct comparison to the previous studies as the used datasets were very different and the models created might have a different balance between the errors derived from bias and variance and the latter refers to the sensitivity to the used dataset. On the other hand, the fact that the dataset used in this project comprises a much larger number of compounds and, presumably, more diversity of represented chemical structures, will likely have resulted in models with better generalization capability and less variance derived error. Also, making a general comparison with the studies that had a more similar methodology to this one in terms of the source and size of the used dataset, we see that the results obtained were worse than the ones for the studies aimed at predicting inhibitory activity on EGFR that used small training sets, which corroborates the intuition that the use of different datasets with very disparate sizes will lead to different results.

Additionally, since the values for the quality metrics of this study are calculated based on 10-fold cross validation and on testing on unseen data, it is possible to be confident that the application of these models in other datasets would perform similarly without the risk of having a great decrease in the quality of the predictions because of overfitting.

Finally, it is noteworthy that because the resulting best models are what is considered ‘black boxes’, that is, do not produce interpretable relationships between the input and output variables, and the experimented methods of feature selection did not produce better results, the determination of the most important features for this classification problem was not possible, which could be interesting information, although for these kind of projects, with the objective of screening large databases, what is most important is for the model to be able to make accurate predictions and not so much to understand the underlying relationships in the data.

5. CONCLUSIONS

At the end of this study, it is possible to say that the objectives were met.

The proposed review of similar studies, although not exhaustive, is both comprehensive and relevant, serving as the grounds of the current state of the art both in studies aimed at the prediction of EGFR inhibitors and in general predictive studies that used PubChem's database as a data source.

Another one of the objectives was to establish a methodology that could easily be implemented to similar studies which was achieved, as all the tools used are readily available and allow the user to easily gain the necessary knowledge to use them.

It was verified that PubChem can indeed be used both as a data source for the activity of compounds and for the necessary information to build chemical descriptors. Also, it was shown that PubChem's Substructure Fingerprints are good chemical descriptors to be used as input features considering the quality of the models developed.

On the other hand, the best models developed could easily be used for making predictions both on PubChem's database or others, as using the chemical formula or another identifier of a compound, it is possible to import the Substructure Fingerprints from PubChem and thus create a dataset where the model could make predictions.

Regarding the overall quality of the models produced, as stated earlier, there have been studies for the same purpose both with better and worse results but a direct comparison of the models' performance is not possible for the reasons mentioned, in particular, because of the use of such different training datasets.

As such, it is considered that the best models developed have good performance although with room for improvement, that could be achieved with the extension of this work both in the data, descriptors and algorithms used.

6. LIMITATIONS AND RECOMMENDATIONS FOR FUTURE WORKS

The main limitations of this study were time and computational power constraints. As such, in future works that would extend on this one, there would be approaches that could be tried to improve the results obtained.

In this study, one of the objectives was determining the quality of PubChem's Substructure fingerprints as descriptors for predictive modelling, but with the possibility to extend the study, it would be of interest to also use other chemical descriptors, specially combining 2D and 3D descriptors, to investigate if those would produce a better model, since we have seen other authors use other kinds of chemical descriptors with good results.

Also, since the Multilayer Perceptron was the model that was considered the best for future predictions on unseen data, taking into account both its metrics and running time, it would be of interest to try both the use of other tools that allow more fine tuning of the model to see if it would be possible to improve on it and also other variations of Neural Networks, such as Deep Belief Networks. Other algorithms could also be tried, such as Genetic Algorithms.

In a study not only focused on PubChem's database, the use of additional data about inhibitors not present in the used database, if possible to gather, would probably benefit the model.

On the other hand, it would also be useful to test the model on a different dataset and make predictions on PubChem itself or other databases and analyze the type of compounds predicted as positive for the inhibition of EGFR.

7. BIBLIOGRAPHY

Austin, C. P., Brady, L. S., Insel, T. R., & Collins, F. S. (2004). NIH Molecular Libraries Initiative. *Science (New York, N.Y.)*, *306*(5699), 1138–9. https://doi.org/10.1126/science.1105511

Bento, A. P., Gaulton, A., Hersey, A., Bellis, L. J., Chambers, J., Davies, M., … Overington, J. P. (2014). The ChEMBL bioactivity database: an update. *Nucleic Acids Research*, *42*(D1), D1083–D1090. https://doi.org/10.1093/nar/gkt1031

Berndt, E. R., Nass, D., Kleinrock, M., & Aitken, M. (2015). Decline in economic returns from new drugs raises questions about sustaining innovations. *Health Affairs (Project Hope)*, *34*(2), 245–52. https://doi.org/10.1377/hlthaff.2014.1029

Bilsland, A. E., Pugliese, A., Liu, Y., Revie, J., Burns, S., McCormick, C., … Keith, W. N. (2015). Identification of a Selective G1-Phase Benzimidazolone Inhibitor by a Senescence-Targeted Virtual Screen Using Artificial Neural Networks. *Neoplasia*, *17*(9), 704–715. https://doi.org/10.1016/j.neo.2015.08.009

BIOVIA Databases | Bioactivity Databases: MDDR. (n.d.). Retrieved July 24, 2017, from http://accelrys.com/products/collaborative-science/databases/bioactivity-databases/mddr.html

Chavan, S., Abdelaziz, A., Wiklander, J. G., & Nicholls, I. A. (2016). A k-nearest neighbor classification of hERG K+ channel blockers. *Journal of Computer-Aided Molecular Design*, *30*(3), 229–236. https://doi.org/10.1007/s10822-016-9898-z

Chen, B., & Wild, D. J. (2010). PubChem BioAssays as a data source for predictive models. *Journal of Molecular Graphics and Modelling*, *28*(5), 420–426. https://doi.org/10.1016/j.jmgm.2009.10.001

Cheng, F., Yu, Y., Shen, J., Yang, L., Li, W., Liu, G., … Tang, Y. (2011). Classification of Cytochrome P450 Inhibitors and Noninhibitors Using Combined Classifiers. *Journal of Chemical Information and Modeling*, *51*(5), 996–1011. https://doi.org/10.1021/ci200028n

Cherkasov, A., Muratov, E. N., Fourches, D., Varnek, A., Baskin, I. I., Cronin, M., … Tropsha, A. (2014). QSAR Modeling: Where Have You Been? Where Are You Going To? *Journal of Medicinal Chemistry*, *57*(12), 4977–5010. https://doi.org/10.1021/jm4004285

Daniel T. Larose, C. D. L. (2015). *Data Mining and Predictive Analytics (Wiley Series on Methods and Applications in Data Mining)* (Second Edi, Vol. 25). John Wiley & Sons, Inc. https://doi.org/10.1007/s13398-014-0173-7.2

Das, G., Chattopadhyay, M., & Gupta, S. (2016). A comparison of self-organising maps and principal components analysis. *International Journal of Market Research*, *58*(6), 815. https://doi.org/10.2501/IJMR-2016-039

Data Mining From A to Z. (n.d.). Retrieved July 29, 2017, from https://www.sas.com/en_us/whitepapers/data-mining-from-a-z-104937.html

DiMasi, J. A., Grabowski, H. G., & Hansen, R. W. (2016). Innovation in the pharmaceutical industry: New estimates of R&D costs. *Journal of Health Economics*, *47*, 20–33. https://doi.org/10.1016/j.jhealeco.2016.01.012

Edelstein, H. A. (1999). *Introduction to Data Mining and Knowledge Discovery* (3rd ed.). Two Crows

Corporation.

Fernandez-Lozano, C., Canto, C., Gestal, M., Andrade-Garda, J. M., Rabuñal, J. R., Dorado, J., & Pazos, A. (2013). Hybrid model based on Genetic Algorithms and SVM applied to variable selection within fruit juice classification. *TheScientificWorldJournal*, *2013*, 982438. https://doi.org/10.1155/2013/982438

Fonger, G. C., Hakkinen, P., Jordan, S., & Publicker, S. (2014). The National Library of Medicine's (NLM) Hazardous Substances Data Bank (HSDB): background, recent enhancements and future plans. *Toxicology*, *325*, 209–16. https://doi.org/10.1016/j.tox.2014.09.003

Foster, P., & Fawcett, T. (2013). *Data Science for Business* (First Edit). O'Reilly Media, Inc.

Friedman, J. H., & Friedman, J. H. (1997). Data mining and statistics: What's the connection. *PROCEEDINGS OF THE 29TH SYMPOSIUM ON THE INTERFACE BETWEEN COMPUTER SCIENCE AND STATISTICS*. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.374.7489

Gasteiger†, J. (2006). Of Molecules and Humans. https://doi.org/10.1021/JM0608964

Géron, A. (2017). *Hands-On Machine Learning with Scikit-Learn and TensorFlow* (First Edit). O'Reilly Media, Inc.

Grus, J. (2015). *Data Science from Scratch* (First Edit). O'Reilly Media, Inc.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA data mining software. *ACM SIGKDD Explorations Newsletter*, *11*(1), 10. https://doi.org/10.1145/1656274.1656278

Han, B., Ma, X., Zhao, R., Zhang, J., Wei, X., Liu, X., … Chen, Y. (2012). Development and experimental test of support vector machines virtual screening method for searching Src inhibitors from large compound libraries. *Chemistry Central Journal*, *6*(1), 139. https://doi.org/10.1186/1752-153X-6-139

Han, J., Kamber, M., & Pei, J. (2012). *Data Mining Concepts and Techniques* (Third Edit). Elsevier.

Han, L., Wang, Y., & Bryant, S. H. (2008). Developing and validating predictive decision tree models from mining chemical structural fingerprints and high–throughput screening data in PubChem. *BMC Bioinformatics*, *9*(1), 401. https://doi.org/10.1186/1471-2105-9-401

Hawkins*, D. M. (2003). The Problem of Overfitting. https://doi.org/10.1021/CI0342472

Herbst, R. S. (2004). Review of epidermal growth factor receptor biology. *International Journal of Radiation Oncology*Biology*Physics*, *59*(2), S21–S26. https://doi.org/10.1016/j.ijrobp.2003.11.041

Huang, N., Shoichet, B. K., & Irwin, J. J. (2006). Benchmarking Sets for Molecular Docking. *Journal of Medicinal Chemistry*, *49*(23), 6789–6801. https://doi.org/10.1021/jm0608356

Irwin*, J. J. (2005). Software Review: ChemOffice 2005 Pro by CambridgeSoft. https://doi.org/10.1021/CI050286X

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning*. Springer Science+Business Media.

Khanna, V., & Ranganathan, S. (2011). In silico approach to screen compounds active against parasitic

nematodes of major socio-economic importance. *BMC Bioinformatics*, *12*(Suppl 13), S25. https://doi.org/10.1186/1471-2105-12-S13-S25

Kim, J.-H. (2009). Estimating classification error rate: Repeated cross-validation, repeated hold-out and bootstrap. *Computational Statistics & Data Analysis*, *53*(11), 3735–3745. https://doi.org/10.1016/j.csda.2009.04.009

Kim, S. (2016). Getting the most out of PubChem for virtual screening. *Expert Opinion on Drug Discovery*, *11*(9), 843–855. https://doi.org/10.1080/17460441.2016.1216967

Kim, S., Thiessen, P. A., Bolton, E. E., Chen, J., Fu, G., Gindulyte, A., … Bryant, S. H. (2016). PubChem Substance and Compound databases. *Nucleic Acids Research*, *44*(D1), D1202–D1213. https://doi.org/10.1093/nar/gkv951

Kong, Y., Qu, D., Chen, X., Gong, Y.-N., & Yan, A. (2016). Self-Organizing Map (SOM) and Support Vector Machine (SVM) Models for the Prediction of Human Epidermal Growth Factor Receptor (EGFR/ ErbB-1) Inhibitors. *Combinatorial Chemistry & High Throughput Screening*, *19*(5), 400–411. https://doi.org/10.2174/1386207319666160414105044

Koza, J. R. (1992). *Genetic programming : on the programming of computers by means of natural selection*. MIT Press.

Law, V., Knox, C., Djoumbou, Y., Jewison, T., Guo, A. C., Liu, Y., … Wishart, D. S. (2014). DrugBank 4.0: shedding new light on drug metabolism. *Nucleic Acids Research*, *42*(Database issue), D1091-7. https://doi.org/10.1093/nar/gkt1068

Leskovec, J., Rajaraman, A., & Ullman, J. D. (2011). *Mining of Massive Datasets*. *Lecture Notes for Stanford CS345A Web Mining* (First Edit, Vol. 67). Cambridge University Press. https://doi.org/10.1017/CBO9781139058452

Li, H., You, L., Xie, J., Pan, H., & Han, W. (2017). The roles of subcellularly located EGFR in autophagy. *Cellular Signalling*, *35*, 223–230. https://doi.org/10.1016/j.cellsig.2017.04.012

Li, Q., Wang, Y., & Bryant, S. H. (2009). A novel method for mining highly imbalanced high-throughput screening data in PubChem. *Bioinformatics*, *25*(24), 3310–3316. https://doi.org/10.1093/bioinformatics/btp589

Madej, T., Lanczycki, C. J., Zhang, D., Thiessen, P. A., Geer, R. C., Marchler-Bauer, A., & Bryant, S. H. (2014). MMDB and VAST+: tracking structural similarities between macromolecular complexes. *Nucleic Acids Research*, *42*(Database issue), D297-303. https://doi.org/10.1093/nar/gkt1208

Marsland, S. (2015). *Machine Learning : An ALgorithmic Perpective* (Second). CRC Press.

McInnes, C. (2007). Virtual screening strategies in drug discovery. *Current Opinion in Chemical Biology*, *11*(5), 494–502. https://doi.org/10.1016/j.cbpa.2007.08.033

Novotarskyi, S., Sushko, I., Körner, R., Pandey, A. K., & Tetko, I. V. (2011). A comparison of different QSAR approaches to modeling CYP450 1A2 inhibition. *Journal of Chemical Information and Modeling*, *51*(6), 1271–1280. https://doi.org/10.1021/ci200091h

NumPy — NumPy. (n.d.). Retrieved August 23, 2017, from http://www.numpy.org/

Oda, K., Matsuoka, Y., Funahashi, A., & Kitano, H. (2005). A comprehensive pathway map of epidermal growth factor receptor signaling. *Molecular Systems Biology*, *1*, 2005.0010. https://doi.org/10.1038/msb4100014

Online Chemical Modeling Environment. (n.d.). Retrieved August 14, 2017, from https://ochem.eu/home/show.do

Pouliot, Y., Chiang, A. P., & Butte, A. J. (2011). Predicting Adverse Drug Reactions Using Publicly Available PubChem BioAssay Data. *Clinical Pharmacology & Therapeutics*, *90*(1), 90–99. https://doi.org/10.1038/clpt.2011.81

PubChem Data Sources. (n.d.). Retrieved July 9, 2017, from https://pubchem.ncbi.nlm.nih.gov/sources/

PubChemPy documentation — PubChemPy 1.0.4 documentation. (n.d.). Retrieved August 30, 2017, from https://pubchempy.readthedocs.io/en/latest/

Python Data Analysis Library — pandas: Python Data Analysis Library. (n.d.). Retrieved August 23, 2017, from http://pandas.pydata.org/

Ramezankhani, A., Pournik, O., Shahrabi, J., Azizi, F., Hadaegh, F., & Khalili, D. (2016). The Impact of Oversampling with SMOTE on the Performance of 3 Classifiers in Prediction of Type 2 Diabetes. *Medical Decision Making*, *36*(1), 137–144. https://doi.org/10.1177/0272989X14560647

Receptor Binding. (n.d.). Retrieved July 25, 2017, from http://www.fenichel.net/pages/Professional/subpages/QT/Tables/pbydrug.htm

Rumelhart, D. E., Widrow, B., & Lehr, M. A. (1994). The basic ideas in neural networks. *Communications of the ACM*, *37*(3), 87–92. https://doi.org/10.1145/175247.175256

Scharadin, T. M., He, W., Yiannakou, Y., Tomilov, A. A., Saldana, M., Cortopassi, G. A., … Henderson, P. T. (2017). Synthesis and biochemical characterization of EGF receptor in a water-soluble membrane model system. *PLOS ONE*, *12*(6), e0177761. https://doi.org/10.1371/journal.pone.0177761

Schierz, A. C. (2009). Virtual screening of bioassay data. *Journal of Cheminformatics*, *1*(1), 21. https://doi.org/10.1186/1758-2946-1-21

scikit-learn: machine learning in Python — scikit-learn 0.19.0 documentation. (n.d.). Retrieved August 23, 2017, from http://scikit-learn.org/stable/

Shen, M., Su, B.-H., Esposito, E. X., Hopfinger, A. J., & Tseng, Y. J. (2011). A Comprehensive Support Vector Machine Binary hERG Classification Model Based on Extensive but Biased End Point hERG Data Sets. *Chemical Research in Toxicology*, *24*(6), 934–949. https://doi.org/10.1021/tx200099j

Singh, H., Singh, S., Singla, D., Agarwal, S. M., Raghava, G. P. S., & Agarwal, S. (2015). QSAR based model for discriminating EGFR inhibitors and non-inhibitors using Random forest. *Biology Direct 2015 10:1*, *8*(1), 28. https://doi.org/10.1186/1745-6150-8-28

Su, B.-H., Tu, Y., Lin, C., Shao, C.-Y., Lin, O. A., & Tseng, Y. J. (2015). Rule-Based Prediction Models of Cytochrome P450 Inhibition. *Journal of Chemical Information and Modeling*, *55*(7), 1426–1434. https://doi.org/10.1021/acs.jcim.5b00130

Szymański, P., Markowicz, M., & Mikiciuk-Olasik, E. (2012). Adaptation of high-throughput screening in drug discovey-toxicological screening tests. *International Journal of Molecular Sciences*, *13*(1), 427–452. https://doi.org/10.3390/ijms13010427

The PubChem Project. (n.d.). Retrieved July 9, 2017, from https://pubchem.ncbi.nlm.nih.gov/

Troiani, T., Martinelli, E., Capasso, A., Morgillo, F., Orditura, M., De Vita, F., & Ciardiello, F. (2012). Targeting EGFR in pancreatic cancer treatment. *Current Drug Targets*, *13*(6), 802–10. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/22458527

Vastag, B. (2005). For EGFR Research, New Targeted Drugs Mean New Questions. *JNCI Journal of the National Cancer Institute*, *97*(9), 628–630. https://doi.org/10.1093/jnci/97.9.628

Wang, S., Xie, L., Wang, H., Zhu, B., & Niu, A. (2016). Prediction of selective estrogen receptor beta agonist using open data and machine learning approach. *Drug Design, Development and Therapy*, *Volume 10*, 2323–2331. https://doi.org/10.2147/DDDT.S110603

Wang, Y., Suzek, T., Zhang, J., Wang, J., He, S., Cheng, T., … Bryant, S. H. (2014). PubChem BioAssay: 2014 update. *Nucleic Acids Research*, *42*(D1), D1075–D1082. https://doi.org/10.1093/nar/gkt978

Wang, Y., Xiao, J., Suzek, T. O., Zhang, J., Wang, J., & Bryant, S. H. (2009). PubChem: a public information system for analyzing bioactivities of small molecules. *Nucleic Acids Research*, *37*(Web Server issue), W623-33. https://doi.org/10.1093/nar/gkp456

Weis, D. C., Visco, D. P., & Faulon, J. L. (2008). Data mining PubChem using a support vector machine with the Signature molecular descriptor: Classification of factor XIa inhibitors. *Journal of Molecular Graphics and Modelling*, *27*(4), 466–475. https://doi.org/10.1016/j.jmgm.2008.08.004

Welcome to Python.org. (n.d.). Retrieved August 23, 2017, from https://www.python.org/

Witten, I. H., & Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. *Complementary literature None* (Second Edi). Elsevier. https://doi.org/0120884070, 9780120884070

Yarden, Y., & Sliwkowski, M. X. (2001). Untangling the ErbB signalling network. *Nature Reviews Molecular Cell Biology*, *2*(2), 127–137. https://doi.org/10.1038/35052073

Yu, L., Shi, X., Tian, S., Gao, S., & Li, L. (2017). Classification of Cytochrome P450 1A2 Inhibitors and Noninhibitors Based on Deep Belief Network. *International Journal of Computational Intelligence and Applications*, *16*(1), 1750002. https://doi.org/10.1142/S146902681750002X

Zhao, M., Wang, L., Zheng, L., Zhang, M., Qiu, C., Zhang, Y., … Niu, B. (2017). 2D-QSAR and 3D-QSAR Analyses for EGFR Inhibitors. *BioMed Research International*, *2017*, 1–11. https://doi.org/10.1155/2017/4649191

8. ANNEXES

8.1. CODE

8.1.1. Gaussian Naïve Bays

In [1]:

```
import numpy as np
import pandas as pd
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_validate
from sklearn.cross_validation import train_test_split
from sklearn.metrics import accuracy_score, recall_score, precision_score
from sklearn.metrics import classification_report,confusion_matrix
from imblearn.over_sampling import SMOTE
```

```
C:\Users\Liliana Rosa\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: D
eprecationWarning: This module was deprecated in version 0.18 in favor of the model
_selection module into which all the refactored classes and functions are moved. Al
so note that the interface of the new CV iterators are different from that of this
module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [2]:

```
df = pd.read_csv('Dataset_EGFR.csv', index_col = False)
X = df.drop(['PUBCHEM_ACTIVITY_OUTCOME'],axis=1)
y = df['PUBCHEM_ACTIVITY_OUTCOME']
X = X.dropna(axis=1)
```

In [3]:

```
from sklearn.model_selection import GridSearchCV
```

In [4]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_st
ate=42)
sm = SMOTE(random_state=12, ratio = 0.7)
X_train_res, y_train_res = sm.fit_sample(X_train, y_train)
```

In [5]:

```
cls = GaussianNB()
```

In [6]:

```
scoring = ['accuracy', 'recall', 'precision']
```

In [7]:

```
scores = cross_validate(cls, X_train_res, y_train_res, scoring=scoring, cv=10)
```

In [8]:

```
print('\nAverage Accuracy %.2f +/- %.2f' % (np.mean(scores['test_accuracy']), np.std(scores['test_accuracy'])))

print('\nAverage Recall %.2f +/- %.2f' % (np.mean(scores['test_recall']), np.std(scores['test_recall'])))

print('\nAverage Precision %.2f +/- %.2f' % (np.mean(scores['test_precision']), np.std(scores['test_precision'])))
```

```
Average Accuracy 0.55 +/- 0.01

Average Recall 0.95 +/- 0.01

Average Precision 0.48 +/- 0.01
```

In [9]:

```
model = cls.fit(X_train_res, y_train_res)
```

In [10]:

```
train_acc = accuracy_score(y_true=y_train_res, y_pred=model.predict(X_train_res))

train_rec = recall_score(y_true=y_train_res, y_pred=model.predict(X_train_res))

train_prec = precision_score(y_true=y_train_res, y_pred=model.predict(X_train_res))

test_acc = accuracy_score(y_true=y_test, y_pred=model.predict(X_test))

test_rec = recall_score(y_true=y_test, y_pred=model.predict(X_test))

test_prec = precision_score(y_true=y_test, y_pred=model.predict(X_test))

predictions = model.predict(X_test)

print('Training accuracy: %.2f' % train_acc)

print('Training recall: %.2f' % train_rec)

print('Training precision: %.2f' % train_prec)

print('Test accuracy: %.2f' % test_acc)

print('Test recall: %.2f' % test_rec)

print('Test precision: %.2f' % test_prec)

print(classification_report(y_test,predictions))

print(confusion_matrix(y_test,predictions))
```

```
Training accuracy: 0.55
Training recall: 0.97
Training precision: 0.48
Test accuracy: 0.50
Test recall: 0.94
Test precision: 0.42
             precision    recall  f1-score   support

          0       0.88      0.25      0.38      2497
          1       0.42      0.94      0.58      1438

avg / total       0.71      0.50      0.46      3935

[[ 613 1884]
 [  84 1354]]
```

8.1.2. Bernoulli Naïve Bayes

In [1]:

```
import numpy as np
import pandas as pd
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_validate
from sklearn.cross_validation import train_test_split
from sklearn.metrics import accuracy_score, recall_score, precision_score
from sklearn.metrics import classification_report,confusion_matrix
from imblearn.over_sampling import SMOTE
from sklearn.naive_bayes import BernoulliNB
```

```
C:\Users\Liliana Rosa\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: D
eprecationWarning: This module was deprecated in version 0.18 in favor of the model
_selection module into which all the refactored classes and functions are moved. Al
so note that the interface of the new CV iterators are different from that of this
module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [2]:

```
df = pd.read_csv('Dataset_EGFR.csv', index_col = False)
X = df.drop(['PUBCHEM_ACTIVITY_OUTCOME'],axis=1)
y = df['PUBCHEM_ACTIVITY_OUTCOME']
X = X.dropna(axis=1)
```

In [3]:

```
from sklearn.model_selection import GridSearchCV
```

In [4]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_st
ate=42)
sm = SMOTE(random_state=12, ratio = 0.7)
X_train_res, y_train_res = sm.fit_sample(X_train, y_train)
```

In [5]:

```
cls = BernoulliNB()
```

In [6]:

```
scoring = ['accuracy', 'recall', 'precision']
```

In [7]:

```
scores = cross_validate(cls, X_train_res, y_train_res, scoring=scoring, cv=10)
```

In [8]:

```
print('\nAverage Accuracy %.2f +/- %.2f' % (np.mean(scores['test_accuracy']), np.st
d(scores['test_accuracy'])))
```

```
print('\nAverage Recall %.2f +/- %.2f' % (np.mean(scores['test_recall']), np.std(sc
ores['test_recall'])))
print('\nAverage Precision %.2f +/- %.2f' % (np.mean(scores['test_precision']), np.
std(scores['test_precision'])))
```

```
Average Accuracy 0.70 +/- 0.01

Average Recall 0.70 +/- 0.03

Average Precision 0.62 +/- 0.02
```

In [9]:

```
model = cls.fit(X_train_res, y_train_res)
```

In [10]:

```
train_acc = accuracy_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
train_rec = recall_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
train_prec = precision_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
test_acc = accuracy_score(y_true=y_test, y_pred=model.predict(X_test))
test_rec = recall_score(y_true=y_test, y_pred=model.predict(X_test))
test_prec = precision_score(y_true=y_test, y_pred=model.predict(X_test))
predictions = model.predict(X_test)
print('Training accuracy: %.2f' % train_acc)
print('Training recall: %.2f' % train_rec)
print('Training precision: %.2f' % train_prec)
print('Test accuracy: %.2f' % test_acc)
print('Test recall: %.2f' % test_rec)
print('Test precision: %.2f' % test_prec)
print(classification_report(y_test,predictions))
print(confusion_matrix(y_test,predictions))
```

```
Training accuracy: 0.70
Training recall: 0.70
Training precision: 0.63
Test accuracy: 0.70
Test recall: 0.69
Test precision: 0.58
             precision    recall  f1-score   support

          0       0.80      0.71      0.75      2497
          1       0.58      0.69      0.63      1438

avg / total       0.72      0.70      0.71      3935

[[1771  726]
 [ 447  991]]
```

8.1.3. Decision Tree

In [2]:

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

```
from sklearn.tree import DecisionTreeClassifier

from sklearn.model_selection import cross_validate

from sklearn.cross_validation import train_test_split

from sklearn.metrics import accuracy_score, recall_score, precision_score

from sklearn.metrics import classification_report,confusion_matrix

from imblearn.over_sampling import SMOTE
```

```
C:\Users\Liliana Rosa\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: D
eprecationWarning: This module was deprecated in version 0.18 in favor of the model
_selection module into which all the refactored classes and functions are moved. Al
so note that the interface of the new CV iterators are different from that of this
module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [3]:

```
df = pd.read_csv('Dataset_EGFR.csv', index_col = False)

X = df.drop(['PUBCHEM_ACTIVITY_OUTCOME'],axis=1)

y = df['PUBCHEM_ACTIVITY_OUTCOME']

X = X.dropna(axis=1)
```

In [4]:

```
from sklearn.model_selection import GridSearchCV
```

In [5]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_st
ate=42)

sm = SMOTE(random_state=12, ratio = 0.7)

X_train_res, y_train_res = sm.fit_sample(X_train, y_train)
```

In [6]:

```
cls = DecisionTreeClassifier()
```

In [7]:

```
scoring = ['accuracy', 'recall', 'precision']
```

In [8]:

```
scores = cross_validate(cls, X_train_res, y_train_res, scoring=scoring, cv=10)
```

In [9]:

```
print('\nAverage Accuracy %.2f +/- %.2f' % (np.mean(scores['test_accuracy']), np.st
d(scores['test_accuracy'])))

print('\nAverage Recall %.2f +/- %.2f' % (np.mean(scores['test_recall']), np.std(sc
ores['test_recall'])))

print('\nAverage Precision %.2f +/- %.2f' % (np.mean(scores['test_precision']), np.
std(scores['test_precision'])))
```

```
Average Accuracy 0.87 +/- 0.03

Average Recall 0.83 +/- 0.05
```

```
Average Precision 0.84 +/- 0.02
```

In [10]:

```
model = cls.fit(X_train_res, y_train_res)
```

In [12]:

```
train_acc = accuracy_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
train_rec = recall_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
train_prec = precision_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
test_acc = accuracy_score(y_true=y_test, y_pred=model.predict(X_test))
test_rec = recall_score(y_true=y_test, y_pred=model.predict(X_test))
test_prec = precision_score(y_true=y_test, y_pred=model.predict(X_test))
predictions = model.predict(X_test)
print('Training accuracy: %.2f' % train_acc)
print('Training recall: %.2f' % train_rec)
print('Training precision: %.2f' % train_prec)
print('Test accuracy: %.2f' % test_acc)
print('Test recall: %.2f' % test_rec)
print('Test precision: %.2f' % test_prec)
print(classification_report(y_test,predictions))
print(confusion_matrix(y_test,predictions))
```

```
Training accuracy: 0.96
Training recall: 0.94
Training precision: 0.96
Test accuracy: 0.86
Test recall: 0.79
Test precision: 0.81
             precision    recall  f1-score   support

          0       0.88      0.90      0.89      2497
          1       0.81      0.79      0.80      1438

avg / total       0.86      0.86      0.86      3935

[[2237  260]
 [ 304 1134]]
```

8.1.4. *k*-Nearest Neighbors

In [1]:

```
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_validate
from sklearn.cross_validation import train_test_split
from sklearn.metrics import accuracy_score, recall_score, precision_score
from sklearn.metrics import classification_report,confusion_matrix
from imblearn.over_sampling import SMOTE
```

```
C:\Users\Liliana Rosa\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: D
eprecationWarning: This module was deprecated in version 0.18 in favor of the model
_selection module into which all the refactored classes and functions are moved. Al
so note that the interface of the new CV iterators are different from that of this
module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [2]:

```
df = pd.read_csv('Dataset_EGFR.csv', index_col = False)

X = df.drop(['PUBCHEM_ACTIVITY_OUTCOME'],axis=1)

y = df['PUBCHEM_ACTIVITY_OUTCOME']

X = X.dropna(axis=1)
```

In [3]:

```
from sklearn.model_selection import GridSearchCV
```

In [4]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_st
ate=42)

sm = SMOTE(random_state=12, ratio = 0.7)

X_train_res, y_train_res = sm.fit_sample(X_train, y_train)
```

In [5]:

```
cls = KNeighborsClassifier()
```

In [6]:

```
K = list(range(5,15))

param_grid = {'n_neighbors': K}

scoring = ['accuracy', 'recall', 'precision']
```

In [7]:

```
gs = GridSearchCV(estimator=cls,
                        param_grid=param_grid,
                        scoring='recall',
                        n_jobs=-1,
                        cv=10)
```

In [8]:

```
gs_fit = gs.fit(X_train_res, y_train_res)

print('Best parameters %s' % gs_fit.best_params_)
```

```
Best parameters {'n_neighbors': 5}
```

In [10]:

```
cls_best = KNeighborsClassifier(n_neighbors=5)
```

In [11]:

```
scores = cross_validate(cls_best, X_train_res, y_train_res, scoring=scoring, cv=10)
```

In [12]:

```
print('\nAverage Accuracy %.2f +/- %.2f' % (np.mean(scores['test_accuracy']), np.std(scores['test_accuracy'])))
print('\nAverage Recall %.2f +/- %.2f' % (np.mean(scores['test_recall']), np.std(scores['test_recall'])))
print('\nAverage Precision %.2f +/- %.2f' % (np.mean(scores['test_precision']), np.std(scores['test_precision'])))
```

```
Average Accuracy 0.87 +/- 0.02

Average Recall 0.88 +/- 0.03

Average Precision 0.83 +/- 0.02
```

In [13]:

```
model = cls_best.fit(X_train_res, y_train_res)
```

In [14]:

```
train_acc = accuracy_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
train_rec = recall_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
train_prec = precision_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
test_acc = accuracy_score(y_true=y_test, y_pred=model.predict(X_test))
test_rec = recall_score(y_true=y_test, y_pred=model.predict(X_test))
test_prec = precision_score(y_true=y_test, y_pred=model.predict(X_test))
predictions = model.predict(X_test)
print('Training accuracy: %.2f' % train_acc)
print('Training recall: %.2f' % train_rec)
print('Training precision: %.2f' % train_prec)
print('Test accuracy: %.2f' % test_acc)
print('Test recall: %.2f' % test_rec)
print('Test precision: %.2f' % test_prec)
print(classification_report(y_test,predictions))
print(confusion_matrix(y_test,predictions))
```

```
Training accuracy: 0.91
Training recall: 0.92
Training precision: 0.87
Test accuracy: 0.87
Test recall: 0.86
Test precision: 0.80
             precision    recall  f1-score   support

          0       0.92      0.87      0.90      2497
          1       0.80      0.86      0.83      1438

avg / total       0.87      0.87      0.87      3935

[[2184  313]
 [ 198 1240]]
```

8.1.5. Logistic Regression

In [1]:

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_validate
from sklearn.cross_validation import train_test_split
from sklearn.metrics import accuracy_score, recall_score, precision_score
from sklearn.metrics import classification_report,confusion_matrix
from imblearn.over_sampling import SMOTE
```

```
C:\Users\Liliana Rosa\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: D
eprecationWarning: This module was deprecated in version 0.18 in favor of the model
_selection module into which all the refactored classes and functions are moved. Al
so note that the interface of the new CV iterators are different from that of this
module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [2]:

```
df = pd.read_csv('Dataset_EGFR.csv', index_col = False)
X = df.drop(['PUBCHEM_ACTIVITY_OUTCOME'],axis=1)
y = df['PUBCHEM_ACTIVITY_OUTCOME']
X = X.dropna(axis=1)
```

In [3]:

```
from sklearn.model_selection import GridSearchCV
```

In [4]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_st
ate=42)
sm = SMOTE(random_state=12, ratio = 0.7)
X_train_res, y_train_res = sm.fit_sample(X_train, y_train)
```

In [5]:

```
cls = LogisticRegression(solver='sag', max_iter=4000)
```

In [6]:

```
param_grid ={'C': [ 0.1, 1, 10]}
scoring = ['accuracy', 'recall', 'precision']
```

In [7]:

```
gs = GridSearchCV(estimator=cls,
                        param_grid=param_grid,
                        scoring='recall',
                        n_jobs=-1,
                        cv=10)
```

In [8]:

```
gs_fit = gs.fit(X_train_res, y_train_res)
print('Best parameters %s' % gs_fit.best_params_)
```

```
Best parameters {'C': 10}
```

In [9]:

```
cls_best = LogisticRegression(C = 10, solver='sag', max_iter=4000)
```

In [10]:

```
scores = cross_validate(cls_best, X_train_res, y_train_res, scoring=scoring, cv=10)
```

In [11]:

```
print('\nAverage Accuracy %.2f +/- %.2f' % (np.mean(scores['test_accuracy']), np.st
d(scores['test_accuracy'])))
print('\nAverage Recall %.2f +/- %.2f' % (np.mean(scores['test_recall']), np.std(sc
ores['test_recall'])))
print('\nAverage Precision %.2f +/- %.2f' % (np.mean(scores['test_precision']), np.
std(scores['test_precision'])))
```

```
Average Accuracy 0.84 +/- 0.01

Average Recall 0.82 +/- 0.02

Average Precision 0.81 +/- 0.01
```

In [12]:

```
model = cls_best.fit(X_train_res, y_train_res)
```

In [13]:

```
train_acc = accuracy_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
train_rec = recall_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
train_prec = precision_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
test_acc = accuracy_score(y_true=y_test, y_pred=model.predict(X_test))
test_rec = recall_score(y_true=y_test, y_pred=model.predict(X_test))
test_prec = precision_score(y_true=y_test, y_pred=model.predict(X_test))
predictions = model.predict(X_test)
print('Training accuracy: %.2f' % train_acc)
print('Training recall: %.2f' % train_rec)
print('Training precision: %.2f' % train_prec)
print('Test accuracy: %.2f' % test_acc)
print('Test recall: %.2f' % test_rec)
print('Test precision: %.2f' % test_prec)
print(classification_report(y_test,predictions))
print(confusion_matrix(y_test,predictions))
```

```
Training accuracy: 0.87
Training recall: 0.85
Training precision: 0.83
Test accuracy: 0.84
```

```
Test recall: 0.79
Test precision: 0.78
             precision    recall  f1-score   support

          0       0.88      0.87      0.87      2497
          1       0.78      0.79      0.79      1438

avg / total       0.84      0.84      0.84      3935

[[2167  330]
 [ 295 1143]]
```

8.1.6. SVM with Linear Kernel

In [28]:

```
import numpy as np
import pandas as pd
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from sklearn.model_selection import cross_validate
from sklearn.cross_validation import train_test_split
from sklearn.metrics import accuracy_score, recall_score, precision_score
from sklearn.metrics import classification_report,confusion_matrix
from imblearn.over_sampling import SMOTE
```

In [29]:

```
df = pd.read_csv('Dataset_EGFR.csv', index_col = False)
X = df.drop(['PUBCHEM_ACTIVITY_OUTCOME'],axis=1)
y = df['PUBCHEM_ACTIVITY_OUTCOME']
X = X.dropna(axis=1)
```

In [30]:

```
from sklearn.model_selection import GridSearchCV
```

In [31]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_st
ate=42)
sm = SMOTE(random_state=12, ratio = 0.7)
X_train_res, y_train_res = sm.fit_sample(X_train, y_train)
```

In [43]:

```
cls = LinearSVC(random_state=24)
```

In [44]:

```
param_grid = {'C': [1, 10, 100]},


scoring = ['accuracy', 'recall', 'precision']
```

In [45]:

```
gs = GridSearchCV(estimator=cls,
                        param_grid=param_grid,
                        scoring='recall',
                        n_jobs=-1,
                        cv=10)
```

In [46]:

```
gs_fit = gs.fit(X_train_res, y_train_res)
print('Best parameters %s' % gs_fit.best_params_)
```

```
Best parameters {'C': 1}
```

In [47]:

```
cls_best = LinearSVC(C=1, random_state= 24)
```

In [48]:

```
scores = cross_validate(cls_best, X_train_res, y_train_res, scoring=scoring, cv=10)
```

In [49]:

```
print('\nAverage Accuracy %.2f +/- %.2f' % (np.mean(scores['test_accuracy']), np.st
d(scores['test_accuracy'])))
print('\nAverage Recall %.2f +/- %.2f' % (np.mean(scores['test_recall']), np.std(sc
ores['test_recall'])))
print('\nAverage Precision %.2f +/- %.2f' % (np.mean(scores['test_precision']), np.
std(scores['test_precision'])))
```

```
Average Accuracy 0.84 +/- 0.01

Average Recall 0.82 +/- 0.02

Average Precision 0.81 +/- 0.02
```

In [50]:

```
model = cls.fit(X_train_res, y_train_res)
```

In [51]:

```
train_acc = accuracy_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
train_rec = recall_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
train_prec = precision_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
test_acc = accuracy_score(y_true=y_test, y_pred=model.predict(X_test))
test_rec = recall_score(y_true=y_test, y_pred=model.predict(X_test))
test_prec = precision_score(y_true=y_test, y_pred=model.predict(X_test))
predictions = model.predict(X_test)
print('Training accuracy: %.2f' % train_acc)
print('Training recall: %.2f' % train_rec)
print('Training precision: %.2f' % train_prec)
print('Test accuracy: %.2f' % test_acc)
print('Test recall: %.2f' % test_rec)
```

```
print('Test precision: %.2f' % test_prec)

print(classification_report(y_test,predictions))

print(confusion_matrix(y_test,predictions))
```

```
Training accuracy: 0.87
Training recall: 0.85
Training precision: 0.83
Test accuracy: 0.84
Test recall: 0.80
Test precision: 0.77
             precision    recall  f1-score   support

          0       0.88      0.86      0.87      2497
          1       0.77      0.80      0.78      1438

avg / total       0.84      0.84      0.84      3935

[[2147  350]
 [ 287 1151]]
```

8.1.7. SVM with RBF Kernel

In [22]:

```
import numpy as np

import pandas as pd

from sklearn.svm import SVC

from sklearn.svm import LinearSVC

from sklearn.model_selection import cross_validate

from sklearn.cross_validation import train_test_split

from sklearn.metrics import accuracy_score, recall_score, precision_score

from sklearn.metrics import classification_report,confusion_matrix

from imblearn.over_sampling import SMOTE
```

In [23]:

```
df = pd.read_csv('Dataset_EGFR.csv', index_col = False)

X = df.drop(['PUBCHEM_ACTIVITY_OUTCOME'],axis=1)

y = df['PUBCHEM_ACTIVITY_OUTCOME']

X = X.dropna(axis=1)
```

In [24]:

```
from sklearn.model_selection import GridSearchCV
```

In [25]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_st
ate=42)

sm = SMOTE(random_state=12, ratio = 0.7)

X_train_res, y_train_res = sm.fit_sample(X_train, y_train)
```

In [30]:

```
cls = SVC(kernel='rbf', random_state=24)
```

In [31]:

```
param_grid ={'C': [1, 10, 100]}


scoring = ['accuracy', 'recall', 'precision']
```

In [32]:

```
gs = GridSearchCV(estimator=cls,
                        param_grid=param_grid,
                        scoring='recall',
                        n_jobs=-1,
                        cv=10)
```

In [33]:

```
gs_fit = gs.fit(X_train_res, y_train_res)
print('Best parameters %s' % gs_fit.best_params_)
```

```
Best parameters {'C': 100}
```

In [35]:

```
cls_best = SVC(kernel='rbf', C=100, random_state=24)
```

In [36]:

```
scores = cross_validate(cls_best, X_train_res, y_train_res, scoring=scoring, cv=10)
```

In [37]:

```
print('\nAverage Accuracy %.2f +/- %.2f' % (np.mean(scores['test_accuracy']), np.st
d(scores['test_accuracy'])))
print('\nAverage Recall %.2f +/- %.2f' % (np.mean(scores['test_recall']), np.std(sc
ores['test_recall'])))
print('\nAverage Precision %.2f +/- %.2f' % (np.mean(scores['test_precision']), np.
std(scores['test_precision'])))
```

```
Average Accuracy 0.89 +/- 0.01

Average Recall 0.89 +/- 0.03

Average Precision 0.84 +/- 0.01
```

In [39]:

```
model = cls_best.fit(X_train_res, y_train_res)
```

In [40]:

```
train_acc = accuracy_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
train_rec = recall_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
train_prec = precision_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
test_acc = accuracy_score(y_true=y_test, y_pred=model.predict(X_test))
test_rec = recall_score(y_true=y_test, y_pred=model.predict(X_test))
```

```
test_prec = precision_score(y_true=y_test, y_pred=model.predict(X_test))

predictions = model.predict(X_test)

print('Training accuracy: %.2f' % train_acc)

print('Training recall: %.2f' % train_rec)

print('Training precision: %.2f' % train_prec)

print('Test accuracy: %.2f' % test_acc)

print('Test recall: %.2f' % test_rec)

print('Test precision: %.2f' % test_prec)

print(classification_report(y_test,predictions))

print(confusion_matrix(y_test,predictions))
```

```
Training accuracy: 0.91
Training recall: 0.91
Training precision: 0.88
Test accuracy: 0.88
Test recall: 0.86
Test precision: 0.82
             precision    recall  f1-score   support

          0       0.92      0.89      0.90      2497
          1       0.82      0.86      0.84      1438

avg / total       0.88      0.88      0.88      3935

[[2230  267]
 [ 207 1231]]
```

8.1.8. Multilayer Perceptron

In [1]:

```
import numpy as np

import pandas as pd

from sklearn.neural_network import MLPClassifier

from sklearn.cross_validation import cross_val_score

from sklearn.model_selection import cross_validate

from sklearn.cross_validation import train_test_split

from sklearn.metrics import accuracy_score, recall_score, precision_score

from sklearn.metrics import classification_report,confusion_matrix

from imblearn.over_sampling import SMOTE
```

```
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/sklea
rn/cross_validation.py:41: DeprecationWarning: This module was deprecated in versio
n 0.18 in favor of the model_selection module into which all the refactored classes
and functions are moved. Also note that the interface of the new CV iterators are d
ifferent from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [2]:

```
df = pd.read_csv('Dataset_EGFR.csv', index_col = False)

X = df.drop(['PUBCHEM_ACTIVITY_OUTCOME'],axis=1)

y = df['PUBCHEM_ACTIVITY_OUTCOME']
```

```
X = X.dropna(axis=1)
```

In [3]:

```
from sklearn.model_selection import GridSearchCV
```

In [4]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_st
ate=42)

sm = SMOTE(random_state=12, ratio = 0.7)

X_train_res, y_train_res = sm.fit_sample(X_train, y_train)
```

In [5]:

```
cls = MLPClassifier(random_state=6409)
```

In [6]:

```
layer_sizes = [(100,100),(300),(400)]

learning_rate_init = [0.0001,0.001]

param_grid = {'hidden_layer_sizes': layer_sizes, 'learning_rate_init' : learning_ra
te_init}

scoring = ['recall','accuracy', 'precision']
```

In [7]:

```
gs = GridSearchCV(estimator=cls,

                        param_grid=param_grid,

                        scoring='recall',

                        n_jobs=-1,

                        cv=10)
```

In [8]:

```
gs_fit = gs.fit(X_train_res, y_train_res)

print('Best parameters %s' % gs_fit.best_params_)
```

```
Best parameters {'hidden_layer_sizes': 400, 'learning_rate_init': 0.001}
```

In [9]:

```
cls_best = MLPClassifier(hidden_layer_sizes=(400), learning_rate_init=0.001, random
_state=6409)
```

In [10]:

```
scores = cross_validate(cls_best, X_train_res, y_train_res, scoring=scoring, cv=10)
```

In [11]:

```
print('\nAverage Accuracy %.2f +/- %.2f' % (np.mean(scores['test_accuracy']), np.st
d(scores['test_accuracy'])))

print('\nAverage Recall %.2f +/- %.2f' % (np.mean(scores['test_recall']), np.std(sc
ores['test_recall'])))
```

```
print('\nAverage Precision %.2f +/- %.2f' % (np.mean(scores['test_precision']), np.
std(scores['test_precision'])))
```

```
Average Accuracy 0.89 +/- 0.01

Average Recall 0.89 +/- 0.04

Average Precision 0.85 +/- 0.01
```

In [12]:

```
model = cls_best.fit(X_train_res, y_train_res)
```

In [13]:

```
train_acc = accuracy_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
train_rec = recall_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
train_prec = precision_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
test_acc = accuracy_score(y_true=y_test, y_pred=model.predict(X_test))
test_rec = recall_score(y_true=y_test, y_pred=model.predict(X_test))
test_prec = precision_score(y_true=y_test, y_pred=model.predict(X_test))
predictions = model.predict(X_test)
print('Training accuracy: %.2f' % train_acc)
print('Training recall: %.2f' % train_rec)
print('Training precision: %.2f' % train_prec)
print('Test accuracy: %.2f' % test_acc)
print('Test recall: %.2f' % test_rec)
print('Test precision: %.2f' % test_prec)
print(classification_report(y_test,predictions))
print(confusion_matrix(y_test,predictions))
```

```
Training accuracy: 0.92
Training recall: 0.98
Training precision: 0.86
Test accuracy: 0.87
Test recall: 0.90
Test precision: 0.78
             precision    recall  f1-score   support

          0       0.94      0.85      0.89      2497
          1       0.78      0.90      0.84      1438

avg / total       0.88      0.87      0.87      3935

[[2129  368]
 [ 139 1299]]
```

8.1.9. Random Forest

In [1]:

```
import numpy as np
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_validate
```

```
from sklearn.cross_validation import train_test_split
from sklearn.metrics import accuracy_score, recall_score, precision_score
from sklearn.metrics import classification_report,confusion_matrix
from imblearn.over_sampling import SMOTE
```

```
C:\Users\Liliana Rosa\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: D
eprecationWarning: This module was deprecated in version 0.18 in favor of the model
_selection module into which all the refactored classes and functions are moved. Al
so note that the interface of the new CV iterators are different from that of this
module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [2]:

```
df = pd.read_csv('Dataset_EGFR.csv', index_col = False)
X = df.drop(['PUBCHEM_ACTIVITY_OUTCOME'],axis=1)
y = df['PUBCHEM_ACTIVITY_OUTCOME']
X = X.dropna(axis=1)
```

In [3]:

```
from sklearn.model_selection import GridSearchCV
```

In [4]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_st
ate=42)
sm = SMOTE(random_state=12, ratio = 0.7)
X_train_res, y_train_res = sm.fit_sample(X_train, y_train)
```

In [5]:

```
cls = RandomForestClassifier(random_state=944)
```

In [6]:

```
estimators = list(range(10,31))
param_grid = {'n_estimators': estimators}
scoring = ['accuracy', 'recall', 'precision']
```

In [7]:

```
gs = GridSearchCV(estimator=cls,
                      param_grid=param_grid,
                      scoring='recall',
                      n_jobs=-1,
                      cv=10)
```

In [8]:

```
gs_fit = gs.fit(X_train_res, y_train_res)
print('Best parameters %s' % gs_fit.best_params_)
```

```
Best parameters {'n_estimators': 25}
```

In [9]:

```
cls_best = RandomForestClassifier(n_estimators=25, random_state=944)
```

In [10]:

```
scores = cross_validate(cls_best, X_train_res, y_train_res, scoring=scoring, cv=10)
```

In [11]:

```
print('\nAverage Accuracy %.2f +/- %.2f' % (np.mean(scores['test_accuracy']), np.st
d(scores['test_accuracy'])))

print('\nAverage Recall %.2f +/- %.2f' % (np.mean(scores['test_recall']), np.std(sc
ores['test_recall'])))

print('\nAverage Precision %.2f +/- %.2f' % (np.mean(scores['test_precision']), np.
std(scores['test_precision'])))
```

```
Average Accuracy 0.89 +/- 0.02

Average Recall 0.86 +/- 0.05

Average Precision 0.87 +/- 0.02
```

In [12]:

```
model = cls_best.fit(X_train_res, y_train_res)
```

In [13]:

```
train_acc = accuracy_score(y_true=y_train_res, y_pred=model.predict(X_train_res))

train_rec = recall_score(y_true=y_train_res, y_pred=model.predict(X_train_res))

train_prec = precision_score(y_true=y_train_res, y_pred=model.predict(X_train_res))

test_acc = accuracy_score(y_true=y_test, y_pred=model.predict(X_test))

test_rec = recall_score(y_true=y_test, y_pred=model.predict(X_test))

test_prec = precision_score(y_true=y_test, y_pred=model.predict(X_test))

predictions = model.predict(X_test)

print('Training accuracy: %.2f' % train_acc)

print('Training recall: %.2f' % train_rec)

print('Training precision: %.2f' % train_prec)

print('Test accuracy: %.2f' % test_acc)

print('Test recall: %.2f' % test_rec)

print('Test precision: %.2f' % test_prec)

print(classification_report(y_test,predictions))

print(confusion_matrix(y_test,predictions))
```

```
Training accuracy: 0.96
Training recall: 0.96
Training precision: 0.94
Test accuracy: 0.88
Test recall: 0.84
Test precision: 0.83
             precision    recall  f1-score   support

          0       0.91      0.90      0.90      2497
          1       0.83      0.84      0.83      1438

avg / total       0.88      0.88      0.88      3935
```

```
[[2252  245]
 [ 234 1204]]
```

8.1.10. Extremely Randomized Tree

In [1]:

```
import numpy as np
import pandas as pd
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import cross_validate
from sklearn.cross_validation import train_test_split
from sklearn.metrics import accuracy_score, recall_score, precision_score
from sklearn.metrics import classification_report,confusion_matrix
from imblearn.over_sampling import SMOTE
```

```
C:\Users\Liliana Rosa\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: D
eprecationWarning: This module was deprecated in version 0.18 in favor of the model
_selection module into which all the refactored classes and functions are moved. Al
so note that the interface of the new CV iterators are different from that of this
module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [2]:

```
df = pd.read_csv('Dataset_EGFR.csv', index_col = False)
X = df.drop(['PUBCHEM_ACTIVITY_OUTCOME'],axis=1)
y = df['PUBCHEM_ACTIVITY_OUTCOME']
X = X.dropna(axis=1)
```

In [3]:

```
from sklearn.model_selection import GridSearchCV
```

In [4]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_st
ate=42)
sm = SMOTE(random_state=12, ratio = 0.7)
X_train_res, y_train_res = sm.fit_sample(X_train, y_train)
```

In [5]:

```
cls = ExtraTreesClassifier(random_state=1)
```

In [29]:

```
estimators = list(range(1,16))
param_grid = {'n_estimators': estimators}
scoring = ['accuracy', 'recall', 'precision']
```

In [30]:

```
gs = GridSearchCV(estimator=cls,
                      param_grid=param_grid,
                      scoring='recall',
```

```
                        n_jobs=-1,
                        cv=10)
```

In [31]:

```
gs_fit = gs.fit(X_train_res, y_train_res)
print('Best parameters %s' % gs_fit.best_params_)
```

```
Best parameters {'n_estimators': 15}
```

In [43]:

```
cls_best = ExtraTreesClassifier(n_estimators=19, random_state=1)
```

In [44]:

```
scores = cross_validate(cls_best, X_train_res, y_train_res, scoring=scoring, cv=10)
```

In [45]:

```
print('\nAverage Accuracy %.2f +/- %.2f' % (np.mean(scores['test_accuracy']), np.st
d(scores['test_accuracy'])))
print('\nAverage Recall %.2f +/- %.2f' % (np.mean(scores['test_recall']), np.std(sc
ores['test_recall'])))
print('\nAverage Precision %.2f +/- %.2f' % (np.mean(scores['test_precision']), np.
std(scores['test_precision'])))
```

```
Average Accuracy 0.89 +/- 0.02

Average Recall 0.86 +/- 0.05

Average Precision 0.87 +/- 0.02
```

In [46]:

```
model = cls_best.fit(X_train_res, y_train_res)
```

In [47]:

```
train_acc = accuracy_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
train_rec = recall_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
train_prec = precision_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
test_acc = accuracy_score(y_true=y_test, y_pred=model.predict(X_test))
test_rec = recall_score(y_true=y_test, y_pred=model.predict(X_test))
test_prec = precision_score(y_true=y_test, y_pred=model.predict(X_test))
predictions = model.predict(X_test)
print('Training accuracy: %.2f' % train_acc)
print('Training recall: %.2f' % train_rec)
print('Training precision: %.2f' % train_prec)
print('Test accuracy: %.2f' % test_acc)
print('Test recall: %.2f' % test_rec)
print('Test precision: %.2f' % test_prec)
print(classification_report(y_test,predictions))
print(confusion_matrix(y_test,predictions))
```

```
Training accuracy: 0.96
Training recall: 0.94
Training precision: 0.96
Test accuracy: 0.87
Test recall: 0.81
Test precision: 0.83
             precision    recall  f1-score   support

          0       0.89      0.91      0.90      2497
          1       0.83      0.81      0.82      1438

avg / total       0.87      0.87      0.87      3935

[[2261  236]
 [ 277 1161]]
```

8.1.11. AdaBoost

In [1]:

```
import numpy as np

import pandas as pd

from sklearn.ensemble import AdaBoostClassifier

from sklearn.cross_validation import cross_val_score

from sklearn.model_selection import cross_validate

from sklearn.cross_validation import train_test_split

from sklearn.metrics import accuracy_score, recall_score, precision_score

from sklearn.metrics import classification_report,confusion_matrix

from imblearn.over_sampling import SMOTE
```

```
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/sklea
rn/cross_validation.py:41: DeprecationWarning: This module was deprecated in versio
n 0.18 in favor of the model_selection module into which all the refactored classes
and functions are moved. Also note that the interface of the new CV iterators are d
ifferent from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [2]:

```
df = pd.read_csv('Dataset_EGFR.csv', index_col = False)

X = df.drop(['PUBCHEM_ACTIVITY_OUTCOME'],axis=1)

y = df['PUBCHEM_ACTIVITY_OUTCOME']

X = X.dropna(axis=1)
```

In [3]:

```
from sklearn.model_selection import GridSearchCV
```

In [4]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_st
ate=42)

sm = SMOTE(random_state=12, ratio = 0.7)

X_train_res, y_train_res = sm.fit_sample(X_train, y_train)
```

In [5]:

```
cls = AdaBoostClassifier(random_state=113)
```

In [11]:

```
estimators = list(range(700,1700,100))
param_grid = {'n_estimators': estimators}
scoring = ['recall','accuracy', 'precision']
```

In [7]:

```
gs = GridSearchCV(estimator=cls,
                        param_grid=param_grid,
                        scoring='accuracy',
                        n_jobs=-1,
                        cv=10)
```

In [8]:

```
gs_fit = gs.fit(X_train_res, y_train_res)
print('Best parameters %s' % gs_fit.best_params_)
```

```
Best parameters {'n_estimators': 1600}
```

In [9]:

```
cls_best = AdaBoostClassifier(n_estimators=1600)
```

In [12]:

```
scores = cross_validate(cls_best, X_train_res, y_train_res, scoring=scoring, cv=10)
```

In [13]:

```
print('\nAverage Accuracy %.2f +/- %.2f' % (np.mean(scores['test_accuracy']), np.st
d(scores['test_accuracy'])))
print('\nAverage Recall %.2f +/- %.2f' % (np.mean(scores['test_recall']), np.std(sc
ores['test_recall'])))
print('\nAverage Precision %.2f +/- %.2f' % (np.mean(scores['test_precision']), np.
std(scores['test_precision'])))
```

```
Average Accuracy 0.84 +/- 0.03

Average Recall 0.79 +/- 0.08

Average Precision 0.81 +/- 0.02
```

In [14]:

```
model = cls_best.fit(X_train_res, y_train_res)
```

In [15]:

```
train_acc = accuracy_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
train_rec = recall_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
train_prec = precision_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
test_acc = accuracy_score(y_true=y_test, y_pred=model.predict(X_test))
test_rec = recall_score(y_true=y_test, y_pred=model.predict(X_test))
test_prec = precision_score(y_true=y_test, y_pred=model.predict(X_test))
predictions = model.predict(X_test)
```

```
print('Training accuracy: %.2f' % train_acc)
print('Training recall: %.2f' % train_rec)
print('Training precision: %.2f' % train_prec)
print('Test accuracy: %.2f' % test_acc)
print('Test recall: %.2f' % test_rec)
print('Test precision: %.2f' % test_prec)
print(classification_report(y_test,predictions))
print(confusion_matrix(y_test,predictions))
```

```
Training accuracy: 0.85
Training recall: 0.81
Training precision: 0.83
Test accuracy: 0.82
Test recall: 0.73
Test precision: 0.77
             precision    recall  f1-score   support

          0       0.85      0.87      0.86      2497
          1       0.77      0.73      0.75      1438

avg / total       0.82      0.82      0.82      3935

[[2174  323]
 [ 386 1052]]
```

8.1.12. Bagging with DT base classifier

In [1]:

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

In [2]:

```
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.cross_validation import cross_val_score
from sklearn.cross_validation import train_test_split
from sklearn.metrics import accuracy_score, recall_score, precision_score
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.model_selection import cross_validate
```

```
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packages/sklea
rn/cross_validation.py:41: DeprecationWarning: This module was deprecated in versio
n 0.18 in favor of the model_selection module into which all the refactored classes
and functions are moved. Also note that the interface of the new CV iterators are d
ifferent from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [3]:

```
from imblearn.over_sampling import SMOTE
```

In [4]:

```
df = pd.read_csv('Dataset_EGFR.csv', index_col = False)
X = df.drop(['PUBCHEM_ACTIVITY_OUTCOME'],axis=1)
y = df['PUBCHEM_ACTIVITY_OUTCOME']
X = X.dropna(axis=1)
```

In [5]:

```
from sklearn.model_selection import GridSearchCV
```

In [6]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_st
ate=42)
sm = SMOTE(random_state=12, ratio = 0.7)
X_train_res, y_train_res = sm.fit_sample(X_train, y_train)
```

In [7]:

```
cls = BaggingClassifier()
scoring = ['recall','accuracy', 'precision']
```

In [8]:

```
scores = cross_validate(cls, X_train_res, y_train_res, scoring=scoring, cv=10)
```

In [9]:

```
print('\nAverage Accuracy %.2f +/- %.2f' % (np.mean(scores['test_accuracy']), np.st
d(scores['test_accuracy'])))
print('\nAverage Recall %.2f +/- %.2f' % (np.mean(scores['test_recall']), np.std(sc
ores['test_recall'])))
print('\nAverage Precision %.2f +/- %.2f' % (np.mean(scores['test_precision']), np.
std(scores['test_precision'])))
```

```
Average Accuracy 0.88 +/- 0.02

Average Recall 0.84 +/- 0.05

Average Precision 0.87 +/- 0.01
```

In [10]:

```
model = cls.fit(X_train_res, y_train_res)
```

In [11]:

```
train_acc = accuracy_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
train_rec = recall_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
train_prec = precision_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
test_acc = accuracy_score(y_true=y_test, y_pred=model.predict(X_test))
test_rec = recall_score(y_true=y_test, y_pred=model.predict(X_test))
test_prec = precision_score(y_true=y_test, y_pred=model.predict(X_test))
```

```
predictions = model.predict(X_test)
print('Training accuracy: %.2f' % train_acc)
print('Training recall: %.2f' % train_rec)
print('Training precision: %.2f' % train_prec)
print('Test accuracy: %.2f' % test_acc)
print('Test recall: %.2f' % test_rec)
print('Test precision: %.2f' % test_prec)
print(classification_report(y_test,predictions))
print(confusion_matrix(y_test,predictions))
```

```
Training accuracy: 0.95
Training recall: 0.94
Training precision: 0.94
Test accuracy: 0.87
Test recall: 0.82
Test precision: 0.83
             precision    recall  f1-score   support

          0       0.89      0.90      0.90      2497
          1       0.83      0.82      0.82      1438

avg / total       0.87      0.87      0.87      3935

[[2254  243]
 [ 265 1173]]
```

8.1.13. Bagging with MLP base classifier

In [1]:

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

In [9]:

```
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.cross_validation import cross_val_score
from sklearn.cross_validation import train_test_split
from sklearn.metrics import accuracy_score, recall_score, precision_score
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.model_selection import cross_validate
```

In [6]:

```
from imblearn.over_sampling import SMOTE
```

In [3]:

```
df = pd.read_csv('Dataset_EGFR.csv', index_col = False)
```

```
X = df.drop(['PUBCHEM_ACTIVITY_OUTCOME'],axis=1)
y = df['PUBCHEM_ACTIVITY_OUTCOME']
X = X.dropna(axis=1)
```

In [4]:

```
from sklearn.model_selection import GridSearchCV
```

In [7]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_st
ate=42)
sm = SMOTE(random_state=12, ratio = 0.7)
X_train_res, y_train_res = sm.fit_sample(X_train, y_train)
```

In [20]:

```
cls = BaggingClassifier(base_estimator=(MLPClassifier(hidden_layer_sizes=(400))))
scoring = ['recall','accuracy', 'precision']
```

In [21]:

```
scores = cross_validate(cls, X_train_res, y_train_res, scoring=scoring, cv=10)
```

In [22]:

```
print('\nAverage Accuracy %.2f +/- %.2f' % (np.mean(scores['test_accuracy']), np.st
d(scores['test_accuracy'])))
print('\nAverage Recall %.2f +/- %.2f' % (np.mean(scores['test_recall']), np.std(sc
ores['test_recall'])))
print('\nAverage Precision %.2f +/- %.2f' % (np.mean(scores['test_precision']), np.
std(scores['test_precision'])))
```

```
Average Accuracy 0.89 +/- 0.02

Average Recall 0.89 +/- 0.04

Average Precision 0.86 +/- 0.01
```

In [23]:

```
model = cls.fit(X_train_res, y_train_res)
```

In [24]:

```
train_acc = accuracy_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
train_rec = recall_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
train_prec = precision_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
test_acc = accuracy_score(y_true=y_test, y_pred=model.predict(X_test))
test_rec = recall_score(y_true=y_test, y_pred=model.predict(X_test))
test_prec = precision_score(y_true=y_test, y_pred=model.predict(X_test))
predictions = model.predict(X_test)
print('Training accuracy: %.2f' % train_acc)
print('Training recall: %.2f' % train_rec)
print('Training precision: %.2f' % train_prec)
```

```
print('Test accuracy: %.2f' % test_acc)
print('Test recall: %.2f' % test_rec)
print('Test precision: %.2f' % test_prec)
print(classification_report(y_test,predictions))
print(confusion_matrix(y_test,predictions))
```

```
Training accuracy: 0.94
Training recall: 0.93
Training precision: 0.92
Test accuracy: 0.88
Test recall: 0.83
Test precision: 0.84
             precision    recall  f1-score   support

          0       0.90      0.91      0.91      2497
          1       0.84      0.83      0.83      1438

avg / total       0.88      0.88      0.88      3935

[[2269  228]
 [ 247 1191]]
```

8.1.14. Voting

In [4]:

```
import numpy as np
import pandas as pd
from sklearn.ensemble import AdaBoostClassifier
from sklearn.cross_validation import cross_val_score
from sklearn.model_selection import cross_validate
from sklearn.cross_validation import train_test_split
from sklearn.metrics import accuracy_score, recall_score, precision_score
from sklearn.metrics import classification_report,confusion_matrix
from imblearn.over_sampling import SMOTE
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
```

In [5]:

```
df = pd.read_csv('Dataset_EGFR.csv', index_col = False)
X = df.drop(['PUBCHEM_ACTIVITY_OUTCOME'],axis=1)
```

```
y = df['PUBCHEM_ACTIVITY_OUTCOME']
X = X.dropna(axis=1)
```

In [6]:

```
from sklearn.model_selection import GridSearchCV
```

In [7]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_st
ate=42)
sm = SMOTE(random_state=12, ratio = 0.7)
X_train_res, y_train_res = sm.fit_sample(X_train, y_train)
```

In [11]:

```
scoring = ['recall','accuracy', 'precision']
```

In [9]:

```
clf1 = LogisticRegression(random_state=1, C=10)
clf2 = RandomForestClassifier(random_state=1, n_estimators=25)
clf3 = GaussianNB()
clf4 = KNeighborsClassifier(n_neighbors=5)
clf5 = SVC()
clf6 = MLPClassifier(hidden_layer_sizes=(400))
cls = VotingClassifier(estimators=[('lr', clf1), ('rf', clf2), ('gnb', clf3), ('knn
', clf4), ('svm', clf5), ('mlp', clf6)], voting='hard')
```

In [12]:

```
scores = cross_validate(cls, X_train_res, y_train_res, scoring=scoring, cv=10)
```

In [13]:

```
print('\nAverage Accuracy %.2f +/- %.2f' % (np.mean(scores['test_accuracy']), np.st
d(scores['test_accuracy'])))
print('\nAverage Recall %.2f +/- %.2f' % (np.mean(scores['test_recall']), np.std(sc
ores['test_recall'])))
print('\nAverage Precision %.2f +/- %.2f' % (np.mean(scores['test_precision']), np.
std(scores['test_precision'])))
```

```
Average Accuracy 0.89 +/- 0.02

Average Recall 0.87 +/- 0.04

Average Precision 0.87 +/- 0.01
```

In [15]:

```
model = cls.fit(X_train_res, y_train_res)
```

In [16]:

```
train_acc = accuracy_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
train_rec = recall_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
train_prec = precision_score(y_true=y_train_res, y_pred=model.predict(X_train_res))
```

```
test_acc = accuracy_score(y_true=y_test, y_pred=model.predict(X_test))
test_rec = recall_score(y_true=y_test, y_pred=model.predict(X_test))
test_prec = precision_score(y_true=y_test, y_pred=model.predict(X_test))
predictions = model.predict(X_test)
print('Training accuracy: %.2f' % train_acc)
print('Training recall: %.2f' % train_rec)
print('Training precision: %.2f' % train_prec)
print('Test accuracy: %.2f' % test_acc)
print('Test recall: %.2f' % test_rec)
print('Test precision: %.2f' % test_prec)
print(classification_report(y_test,predictions))
print(confusion_matrix(y_test,predictions))
```

```
Training accuracy: 0.93
Training recall: 0.91
Training precision: 0.91
Test accuracy: 0.88
Test recall: 0.83
Test precision: 0.84
             precision    recall  f1-score   support

          0       0.90      0.91      0.90      2497
          1       0.84      0.83      0.83      1438

avg / total       0.88      0.88      0.88      3935

[[2263  234]
 [ 244 1194]]
```

8.2. PUBCHEM FINGERPRINT

```
PubChem Substructure Fingerprint
V1.3
http://pubchem.ncbi.nlm.nih.gov

The PubChem System generates a binary substructure fingerprint for
chemical structures.  These fingerprints are used by PubChem for
similarity neighboring and similarity searching.

A substructure is a fragment of a chemical structure.  A
fingerprint is an ordered list of binary (1/0) bits.  Each bit
represents a Boolean determination of, or test for, the presence
of, for example, an element count, a type of ring system, atom
pairing, atom environment (nearest neighbors), etc., in a chemical
structure.

The native format of the PubChem Substructure Fingerprint property
is binary data with a four byte integer prefix, where this integer
prefix indicates the length of the bit list.  For the ASN.1 and XML
formatted data, this property is stored in a PC-InfoData container,
as described by the PCSubstance ASN.1 definition or XML schema:
ftp://ftp.ncbi.nlm.nih.gov/pubchem/specifications/

PC-InfoData is able to handle various types of data.  Each PC-
InfoData has a PC-Urn object (urn = universal resource name). Each
property has a unique trio of "label", "name", and "datatype"
definition (e.g., for PubChem Substructure Fingerprint, this is
"Fingerprint", "SubStructure Keys", and "fingerprint",
respectively).  The fingerprint binary data is hex-encoded, when
```

provided in the XML or textual ASN.1 formats.

When exporting fingerprint information in the SD file format, the SD tag for the PubChem Substructure Fingerprint property is "PUBCHEM_CACTVS_SUBGRAPHKEYS". The PubChem Substructure Fingerprint is Base64 encoded to provide a textual representation of the binary data. For a description of the Base64 encoding and decoding algorithm specification, go to: http://www.faqs.org/rfcs/rfc3548.html

Below is the description of each bit represented in the PubChem Substructure Fingerprint. Some fingerprint bit descriptions are written in SMILES or SMARTS notation. For additional information on SMARTS and SMILES, please go to: http://en.wikipedia.org/wiki/Simplified_molecular_input_line_entry_specification

PubChem Substructure Fingerprint Description

Section 1: Hierarchic Element Counts - These bits test for the presence or count of individual chemical atoms represented by their atomic symbol.

Bit Position	Bit Substructure
0	>= 4 H
1	>= 8 H
2	>= 16 H
3	>= 32 H
4	>= 1 Li
5	>= 2 Li
6	>= 1 B
7	>= 2 B
8	>= 4 B
9	>= 2 C
10	>= 4 C
11	>= 8 C
12	>= 16 C
13	>= 32 C
14	>= 1 N
15	>= 2 N
16	>= 4 N
17	>= 8 N
18	>= 1 O
19	>= 2 O
20	>= 4 O
21	>= 8 O
22	>= 16 O
23	>= 1 F
24	>= 2 F
25	>= 4 F
26	>= 1 Na
27	>= 2 Na
28	>= 1 Si
29	>= 2 Si
30	>= 1 P
31	>= 2 P
32	>= 4 P
33	>= 1 S
34	>= 2 S
35	>= 4 S
36	>= 8 S
37	>= 1 Cl
38	>= 2 Cl
39	>= 4 Cl
40	>= 8 Cl
41	>= 1 K
42	>= 2 K

```
43      >= 1 Br
44      >= 2 Br
45      >= 4 Br
46      >= 1 I
47      >= 2 I
48      >= 4 I
49      >= 1 Be
50      >= 1 Mg
51      >= 1 Al
52      >= 1 Ca
53      >= 1 Sc
54      >= 1 Ti
55      >= 1 V
56      >= 1 Cr
57      >= 1 Mn
58      >= 1 Fe
59      >= 1 Co
60      >= 1 Ni
61      >= 1 Cu
62      >= 1 Zn
63      >= 1 Ga
64      >= 1 Ge
65      >= 1 As
66      >= 1 Se
67      >= 1 Kr
68      >= 1 Rb
69      >= 1 Sr
70      >= 1 Y
71      >= 1 Zr
72      >= 1 Nb
73      >= 1 Mo
74      >= 1 Ru
75      >= 1 Rh
76      >= 1 Pd
77      >= 1 Ag
78      >= 1 Cd
79      >= 1 In
80      >= 1 Sn
81      >= 1 Sb
82      >= 1 Te
83      >= 1 Xe
84      >= 1 Cs
85      >= 1 Ba
86      >= 1 Lu
87      >= 1 Hf
88      >= 1 Ta
89      >= 1 W
90      >= 1 Re
91      >= 1 Os
92      >= 1 Ir
93      >= 1 Pt
94      >= 1 Au
95      >= 1 Hg
96      >= 1 Tl
97      >= 1 Pb
98      >= 1 Bi
99      >= 1 La
100     >= 1 Ce
101     >= 1 Pr
102     >= 1 Nd
103     >= 1 Pm
104     >= 1 Sm
105     >= 1 Eu
106     >= 1 Gd
107     >= 1 Tb
108     >= 1 Dy
109     >= 1 Ho
110     >= 1 Er
```

111 >= 1 Tm
112 >= 1 Yb
113 >= 1 Tc
114 >= 1 U

Section 2: Rings in a canonic Extended Smallest Set of Smallest Rings (ESSSR) ring set - These bits test for the presence or count of the described chemical ring system. An ESSSR ring is any ring which does not share three consecutive atoms with any other ring in the chemical structure. For example, naphthalene has three ESSSR rings (two phenyl fragments and the 10-membered envelope), while biphenyl will yield a count of only two ESSSR rings.

Bit Position	Bit Substructure
115	>= 1 any ring size 3
116	>= 1 saturated or aromatic carbon-only ring size 3
117	>= 1 saturated or aromatic nitrogen-containing ring size 3
118	>= 1 saturated or aromatic heteroatom-containing ring size 3
119	>= 1 unsaturated non-aromatic carbon-only ring size 3
120	>= 1 unsaturated non-aromatic nitrogen-containing ring size 3
121	>= 1 unsaturated non-aromatic heteroatom-containing ring size 3
122	>= 2 any ring size 3
123	>= 2 saturated or aromatic carbon-only ring size 3
124	>= 2 saturated or aromatic nitrogen-containing ring size 3
125	>= 2 saturated or aromatic heteroatom-containing ring size 3
126	>= 2 unsaturated non-aromatic carbon-only ring size 3
127	>= 2 unsaturated non-aromatic nitrogen-containing ring size 3
128	>= 2 unsaturated non-aromatic heteroatom-containing ring size 3
129	>= 1 any ring size 4
130	>= 1 saturated or aromatic carbon-only ring size 4
131	>= 1 saturated or aromatic nitrogen-containing ring size 4
132	>= 1 saturated or aromatic heteroatom-containing ring size 4
133	>= 1 unsaturated non-aromatic carbon-only ring size 4
134	>= 1 unsaturated non-aromatic nitrogen-containing ring size 4
135	>= 1 unsaturated non-aromatic heteroatom-containing ring size 4
136	>= 2 any ring size 4
137	>= 2 saturated or aromatic carbon-only ring size 4
138	>= 2 saturated or aromatic nitrogen-containing ring size 4
139	>= 2 saturated or aromatic heteroatom-containing ring size 4
140	>= 2 unsaturated non-aromatic carbon-only ring size 4
141	>= 2 unsaturated non-aromatic nitrogen-containing ring size 4
142	>= 2 unsaturated non-aromatic heteroatom-containing ring size 4
143	>= 1 any ring size 5
144	>= 1 saturated or aromatic carbon-only ring size 5
145	>= 1 saturated or aromatic nitrogen-containing ring size 5
146	>= 1 saturated or aromatic heteroatom-containing ring size 5
147	>= 1 unsaturated non-aromatic carbon-only ring size 5
148	>= 1 unsaturated non-aromatic nitrogen-containing ring size 5
149	>= 1 unsaturated non-aromatic heteroatom-containing ring size 5
150	>= 2 any ring size 5
151	>= 2 saturated or aromatic carbon-only ring size 5
152	>= 2 saturated or aromatic nitrogen-containing ring size 5
153	>= 2 saturated or aromatic heteroatom-containing ring size 5
154	>= 2 unsaturated non-aromatic carbon-only ring size 5
155	>= 2 unsaturated non-aromatic nitrogen-containing ring size 5
156	>= 2 unsaturated non-aromatic heteroatom-containing ring size 5
157	>= 3 any ring size 5
158	>= 3 saturated or aromatic carbon-only ring size 5
159	>= 3 saturated or aromatic nitrogen-containing ring size 5
160	>= 3 saturated or aromatic heteroatom-containing ring size 5
161	>= 3 unsaturated non-aromatic carbon-only ring size 5
162	>= 3 unsaturated non-aromatic nitrogen-containing ring size 5
163	>= 3 unsaturated non-aromatic heteroatom-containing ring size 5

```
164      >= 4 any ring size 5
165      >= 4 saturated or aromatic carbon-only ring size 5
166      >= 4 saturated or aromatic nitrogen-containing ring size 5
167      >= 4 saturated or aromatic heteroatom-containing ring size 5
168      >= 4 unsaturated non-aromatic carbon-only ring size 5
169      >= 4 unsaturated non-aromatic nitrogen-containing ring size 5
170      >= 4 unsaturated non-aromatic heteroatom-containing ring size 5
171      >= 5 any ring size 5
172      >= 5 saturated or aromatic carbon-only ring size 5
173      >= 5 saturated or aromatic nitrogen-containing ring size 5
174      >= 5 saturated or aromatic heteroatom-containing ring size 5
175      >= 5 unsaturated non-aromatic carbon-only ring size 5
176      >= 5 unsaturated non-aromatic nitrogen-containing ring size 5
177      >= 5 unsaturated non-aromatic heteroatom-containing ring size 5

178      >= 1 any ring size 6
179      >= 1 saturated or aromatic carbon-only ring size 6
180      >= 1 saturated or aromatic nitrogen-containing ring size 6
181      >= 1 saturated or aromatic heteroatom-containing ring size 6
182      >= 1 unsaturated non-aromatic carbon-only ring size 6
183      >= 1 unsaturated non-aromatic nitrogen-containing ring size 6
184      >= 1 unsaturated non-aromatic heteroatom-containing ring size 6
185      >= 2 any ring size 6
186      >= 2 saturated or aromatic carbon-only ring size 6
187      >= 2 saturated or aromatic nitrogen-containing ring size 6
188      >= 2 saturated or aromatic heteroatom-containing ring size 6
189      >= 2 unsaturated non-aromatic carbon-only ring size 6
190      >= 2 unsaturated non-aromatic nitrogen-containing ring size 6
191      >= 2 unsaturated non-aromatic heteroatom-containing ring size 6
192      >= 3 any ring size 6
193      >= 3 saturated or aromatic carbon-only ring size 6
194      >= 3 saturated or aromatic nitrogen-containing ring size 6
195      >= 3 saturated or aromatic heteroatom-containing ring size 6
196      >= 3 unsaturated non-aromatic carbon-only ring size 6
197      >= 3 unsaturated non-aromatic nitrogen-containing ring size 6
198      >= 3 unsaturated non-aromatic heteroatom-containing ring size 6
199      >= 4 any ring size 6
200      >= 4 saturated or aromatic carbon-only ring size 6
201      >= 4 saturated or aromatic nitrogen-containing ring size 6
202      >= 4 saturated or aromatic heteroatom-containing ring size 6
203      >= 4 unsaturated non-aromatic carbon-only ring size 6
204      >= 4 unsaturated non-aromatic nitrogen-containing ring size 6
205      >= 4 unsaturated non-aromatic heteroatom-containing ring size 6
206      >= 5 any ring size 6
207      >= 5 saturated or aromatic carbon-only ring size 6
208      >= 5 saturated or aromatic nitrogen-containing ring size 6
209      >= 5 saturated or aromatic heteroatom-containing ring size 6
210      >= 5 unsaturated non-aromatic carbon-only ring size 6
211      >= 5 unsaturated non-aromatic nitrogen-containing ring size 6
212      >= 5 unsaturated non-aromatic heteroatom-containing ring size 6

213      >= 1 any ring size 7
214      >= 1 saturated or aromatic carbon-only ring size 7
215      >= 1 saturated or aromatic nitrogen-containing ring size 7
216      >= 1 saturated or aromatic heteroatom-containing ring size 7
217      >= 1 unsaturated non-aromatic carbon-only ring size 7
218      >= 1 unsaturated non-aromatic nitrogen-containing ring size 7
219      >= 1 unsaturated non-aromatic heteroatom-containing ring size 7
220      >= 2 any ring size 7
221      >= 2 saturated or aromatic carbon-only ring size 7
222      >= 2 saturated or aromatic nitrogen-containing ring size 7
223      >= 2 saturated or aromatic heteroatom-containing ring size 7
224      >= 2 unsaturated non-aromatic carbon-only ring size 7
225      >= 2 unsaturated non-aromatic nitrogen-containing ring size 7
226      >= 2 unsaturated non-aromatic heteroatom-containing ring size 7

227      >= 1 any ring size 8
228      >= 1 saturated or aromatic carbon-only ring size 8
```

229	>= 1 saturated or aromatic nitrogen-containing ring size 8
230	>= 1 saturated or aromatic heteroatom-containing ring size 8
231	>= 1 unsaturated non-aromatic carbon-only ring size 8
232	>= 1 unsaturated non-aromatic nitrogen-containing ring size 8
233	>= 1 unsaturated non-aromatic heteroatom-containing ring size 8
234	>= 2 any ring size 8
235	>= 2 saturated or aromatic carbon-only ring size 8
236	>= 2 saturated or aromatic nitrogen-containing ring size 8
237	>= 2 saturated or aromatic heteroatom-containing ring size 8
238	>= 2 unsaturated non-aromatic carbon-only ring size 8
239	>= 2 unsaturated non-aromatic nitrogen-containing ring size 8
240	>= 2 unsaturated non-aromatic heteroatom-containing ring size 8
241	>= 1 any ring size 9
242	>= 1 saturated or aromatic carbon-only ring size 9
243	>= 1 saturated or aromatic nitrogen-containing ring size 9
244	>= 1 saturated or aromatic heteroatom-containing ring size 9
245	>= 1 unsaturated non-aromatic carbon-only ring size 9
246	>= 1 unsaturated non-aromatic nitrogen-containing ring size 9
247	>= 1 unsaturated non-aromatic heteroatom-containing ring size 9
248	>= 1 any ring size 10
249	>= 1 saturated or aromatic carbon-only ring size 10
250	>= 1 saturated or aromatic nitrogen-containing ring size 10
251	>= 1 saturated or aromatic heteroatom-containing ring size 10
252	>= 1 unsaturated non-aromatic carbon-only ring size 10
253	>= 1 unsaturated non-aromatic nitrogen-containing ring size 10
254	>= 1 unsaturated non-aromatic heteroatom-containing ring size 10
255	>= 1 aromatic ring
256	>= 1 hetero-aromatic ring
257	>= 2 aromatic rings
258	>= 2 hetero-aromatic rings
259	>= 3 aromatic rings
260	>= 3 hetero-aromatic rings
261	>= 4 aromatic rings
262	>= 4 hetero-aromatic rings

Section 3: Simple atom pairs - These bits test for the presence of patterns of bonded atom pairs, regardless of bond order or count.

Bit Position	Bit Substructure
263	Li-H
264	Li-Li
265	Li-B
266	Li-C
267	Li-O
268	Li-F
269	Li-P
270	Li-S
271	Li-Cl
272	B-H
273	B-B
274	B-C
275	B-N
276	B-O
277	B-F
278	B-Si
279	B-P
280	B-S
281	B-Cl
282	B-Br
283	C-H
284	C-C
285	C-N

286	C-O
287	C-F
288	C-Na
289	C-Mg
290	C-Al
291	C-Si
292	C-P
293	C-S
294	C-Cl
295	C-As
296	C-Se
297	C-Br
298	C-I
299	N-H
300	N-N
301	N-O
302	N-F
303	N-Si
304	N-P
305	N-S
306	N-Cl
307	N-Br
308	O-H
309	O-O
310	O-Mg
311	O-Na
312	O-Al
313	O-Si
314	O-P
315	O-K
316	F-P
317	F-S
318	Al-H
319	Al-Cl
320	Si-H
321	Si-Si
322	Si-Cl
323	P-H
324	P-P
325	As-H
326	As-As

Section 4: Simple atom nearest neighbors - These bits test for the presence of atom nearest neighbor patterns, regardless of bond order (denoted by "~") or count, but where bond aromaticity (denoted by ":") is significant.

Bit Position	Bit Substructure
327	C(~Br)(~C)
328	C(~Br)(~C)(~C)
329	C(~Br)(~H)
330	C(~Br)(:C)
331	C(~Br)(:N)
332	C(~C)(~C)
333	C(~C)(~C)(~C)
334	C(~C)(~C)(~C)(~C)
335	C(~C)(~C)(~C)(~H)
336	C(~C)(~C)(~C)(~N)
337	C(~C)(~C)(~C)(~O)
338	C(~C)(~C)(~H)(~N)
339	C(~C)(~C)(~H)(~O)
340	C(~C)(~C)(~N)
341	C(~C)(~C)(~O)
342	C(~C)(~Cl)
343	C(~C)(~Cl)(~H)
344	C(~C)(~H)

```
345      C(~C)(~H)(~N)
346      C(~C)(~H)(~O)
347      C(~C)(~H)(~O)(~O)
348      C(~C)(~H)(~P)
349      C(~C)(~H)(~S)
350      C(~C)(~I)
351      C(~C)(~N)
352      C(~C)(~O)
353      C(~C)(~S)
354      C(~C)(~Si)
355      C(~C)(:C)
356      C(~C)(:C)(:C)
357      C(~C)(:C)(:N)
358      C(~C)(:N)
359      C(~C)(:N)(:N)
360      C(~Cl)(~Cl)
361      C(~Cl)(~H)
362      C(~Cl)(:C)
363      C(~F)(~F)
364      C(~F)(:C)
365      C(~H)(~N)
366      C(~H)(~O)
367      C(~H)(~O)(~O)
368      C(~H)(~S)
369      C(~H)(~Si)
370      C(~H)(:C)
371      C(~H)(:C)(:C)
372      C(~H)(:C)(:N)
373      C(~H)(:N)
374      C(~H)(~H)(~H)
375      C(~N)(~N)
376      C(~N)(:C)
377      C(~N)(:C)(:C)
378      C(~N)(:C)(:N)
379      C(~N)(:N)
380      C(~O)(~O)
381      C(~O)(:C)
382      C(~O)(:C)(:C)
383      C(~S)(:C)
384      C(:C)(:C)
385      C(:C)(:C)(:C)
386      C(:C)(:C)(:N)
387      C(:C)(:N)
388      C(:C)(:N)(:N)
389      C(:N)(:N)
390      N(~C)(~C)
391      N(~C)(~C)(~C)
392      N(~C)(~C)(~H)
393      N(~C)(~H)
394      N(~C)(~H)(~N)
395      N(~C)(~O)
396      N(~C)(:C)
397      N(~C)(:C)(:C)
398      N(~H)(~N)
399      N(~H)(:C)
400      N(~H)(:C)(:C)
401      N(~O)(~O)
402      N(~O)(:O)
403      N(:C)(:C)
404      N(:C)(:C)(:C)
405      O(~C)(~C)
406      O(~C)(~H)
407      O(~C)(~P)
408      O(~H)(~S)
409      O(:C)(:C)
410      P(~C)(~C)
411      P(~O)(~O)
412      S(~C)(~C)
```

413	S(~C)(~H)
414	S(~C)(~O)
415	Si(~C)(~C)

Section 5: Detailed atom neighborhoods - These bits test for the presence of detailed atom neighborhood patterns, regardless of count, but where bond orders are specific, bond aromaticity matches both single and double bonds, and where "-", "=", and "#" matches a single bond, double bond, and triple bond order, respectively.

Bit Position	Bit Substructure
416	C=C
417	C#C
418	C=N
419	C#N
420	C=O
421	C=S
422	N=N
423	N=O
424	N=P
425	P=O
426	P=P
427	C(#C)(-C)
428	C(#C)(-H)
429	C(#N)(-C)
430	C(-C)(-C)(=C)
431	C(-C)(-C)(=N)
432	C(-C)(-C)(=O)
433	C(-C)(-Cl)(=O)
434	C(-C)(-H)(=C)
435	C(-C)(-H)(=N)
436	C(-C)(-H)(=O)
437	C(-C)(-N)(=C)
438	C(-C)(-N)(=N)
439	C(-C)(-N)(=O)
440	C(-C)(-O)(=O)
441	C(-C)(=C)
442	C(-C)(=N)
443	C(-C)(=O)
444	C(-Cl)(=O)
445	C(-H)(-N)(=C)
446	C(-H)(=C)
447	C(-H)(=N)
448	C(-H)(=O)
449	C(-N)(=C)
450	C(-N)(=N)
451	C(-N)(=O)
452	C(-O)(=O)
453	N(-C)(=C)
454	N(-C)(=O)
455	N(-O)(=O)
456	P(-O)(=O)
457	S(-C)(=O)
458	S(-O)(=O)
459	S(=O)(=O)

Section 6: Simple SMARTS patterns - These bits test for the presence of simple SMARTS patterns, regardless of count, but where bond orders are specific and bond aromaticity matches both single and double bonds.

Bit Position	Bit Substructure
460	C-C-C#C

461	O-C-C=N
462	O-C-C=O
463	N:C-S-[#1]
464	N-C-C=C
465	O=S-C-C
466	N#C-C=C
467	C=N-N-C
468	O=S-C-N
469	S-S-C:C
470	C:C-C=C
471	S:C:C:C
472	C:N:C-C
473	S-C:N:C
474	S:C:C:N
475	S-C=N-C
476	C-O-C=C
477	N-N-C:C
478	S-C=N-[#1]
479	S-C-S-C
480	C:S:C-C
481	O-S-C:C
482	C:N-C:C
483	N-S-C:C
484	N-C:N:C
485	N:C:C:N
486	N-C:N:N
487	N-C=N-C
488	N-C=N-[#1]
489	N-C-S-C
490	C-C-C=C
491	C-N:C-[#1]
492	N-C:O:C
493	O=C-C:C
494	O=C-C:N
495	C-N-C:C
496	N:N-C-[#1]
497	O-C:C:N
498	O-C=C-C
499	N-C:C:N
500	C-S-C:C
501	Cl-C:C-C
502	N-C=C-[#1]
503	Cl-C:C-[#1]
504	N:C:N-C
505	Cl-C:C-O
506	C-C:N:C
507	C-C-S-C
508	S=C-N-C
509	Br-C:C-C
510	[#1]-N-N-[#1]
511	S=C-N-[#1]
512	C-[As]-O-[#1]
513	S:C:C-[#1]
514	O-N-C-C
515	N-N-C-C
516	[#1]-C=C-[#1]
517	N-N-C-N
518	O=C-N-N
519	N=C-N-C
520	C=C-C:C
521	C:N-C-[#1]
522	C-N-N-[#1]
523	N:C:C-C
524	C-C=C-C
525	[As]-C:C-[#1]
526	Cl-C:C-Cl
527	C:C:N-[#1]
528	[#1]-N-C-[#1]

```
529     Cl-C-C-Cl
530     N:C-C:C
531     S-C:C-C
532     S-C:C-[#1]
533     S-C:C-N
534     S-C:C-O
535     O=C-C-C
536     O=C-C-N
537     O=C-C-O
538     N=C-C-C
539     N=C-C-[#1]
540     C-N-C-[#1]
541     O-C:C-C
542     O-C:C-[#1]
543     O-C:C-N
544     O-C:C-O
545     N-C:C-C
546     N-C:C-[#1]
547     N-C:C-N
548     O-C-C:C
549     N-C-C:C
550     Cl-C-C-C
551     Cl-C-C-O
552     C:C-C:C
553     O=C-C=C
554     Br-C-C-C
555     N=C-C=C
556     C=C-C-C
557     N:C-O-[#1]
558     O=N-C:C
559     O-C-N-[#1]
560     N-C-N-C
561     Cl-C-C=O
562     Br-C-C=O
563     O-C-O-C
564     C=C-C=C
565     C:C-O-C
566     O-C-C-N
567     O-C-C-O
568     N#C-C-C
569     N-C-C-N
570     C:C-C-C
571     [#1]-C-O-[#1]
572     N:C:N:C
573     O-C-C=C
574     O-C-C:C-C
575     O-C-C:C-O
576     N=C-C:C-[#1]
577     C:C-N-C:C
578     C-C:C-C:C
579     O=C-C-C-C
580     O=C-C-C-N
581     O=C-C-C-O
582     C-C-C-C-C
583     Cl-C:C-O-C
584     C:C-C=C-C
585     C-C:C-N-C
586     C-S-C-C-C
587     N-C:C-O-[#1]
588     O=C-C-C=O
589     C-C:C-O-C
590     C-C:C-O-[#1]
591     Cl-C-C-C-C
592     N-C-C-C-C
593     N-C-C-C-N
594     C-O-C-C=C
595     C:C-C-C-C
596     N=C-N-C-C
```

597 O=C-C-C:C
598 Cl-C:C:C-C
599 [#1]-C-C=C-[#1]
600 N-C:C:C-C
601 N-C:C:C-N
602 O=C-C-N-C
603 C-C:C:C-C
604 C-O-C-C:C
605 O=C-C-O-C
606 O-C:C-C-C
607 N-C-C-C:C
608 C-C-C-C:C
609 Cl-C-C-N-C
610 C-O-C-O-C
611 N-C-C-N-C
612 N-C-O-C-C
613 C-N-C-C-C
614 C-C-O-C-C
615 N-C-C-O-C
616 C:C:N:N:C
617 C-C-C-O-[#1]
618 C:C-C-C:C
619 O-C-C=C-C
620 C:C-O-C-C
621 N-C:C:C:N
622 O=C-O-C:C
623 O=C-C:C-C
624 O=C-C:C-N
625 O=C-C:C-O
626 C-O-C:C-C
627 O=[As]-C:C:C
628 C-N-C-C:C
629 S-C:C:C-N
630 O-C:C-O-C
631 O-C:C-O-[#1]
632 C-C-O-C:C
633 N-C-C:C-C
634 C-C-C:C-C
635 N-N-C-N-[#1]
636 C-N-C-N-C
637 O-C-C-C-C
638 O-C-C-C-N
639 O-C-C-C-O
640 C=C-C-C-C
641 O-C-C-C=C
642 O-C-C-C=O
643 [#1]-C-C-N-[#1]
644 C-C=N-N-C
645 O=C-N-C-C
646 O=C-N-C-[#1]
647 O=C-N-C-N
648 O=N-C:C-N
649 O=N-C:C-O
650 O=C-N-C=O
651 O-C:C:C-C
652 O-C:C:C-N
653 O-C:C:C-O
654 N-C-N-C-C
655 O-C-C-C:C
656 C-C-N-C-C
657 C-N-C:C-C
658 C-C-S-C-C
659 O-C-C-N-C
660 C-C=C-C-C
661 O-C-O-C-C
662 O-C-C-O-C
663 O-C-C-O-[#1]
664 C-C=C-C=C

665	N-C:C-C-C
666	C=C-C-O-C
667	C=C-C-O-[#1]
668	C-C:C-C-C
669	Cl-C:C-C=O
670	Br-C:C:C-C
671	O=C-C=C-C
672	O=C-C=C-[#1]
673	O=C-C=C-N
674	N-C-N-C:C
675	Br-C-C-C:C
676	N#C-C-C-C
677	C-C=C-C:C
678	C-C-C=C-C
679	C-C-C-C-C-C
680	O-C-C-C-C-C
681	O-C-C-C-C-O
682	O-C-C-C-C-N
683	N-C-C-C-C-C
684	O=C-C-C-C-C
685	O=C-C-C-C-N
686	O=C-C-C-C-O
687	O=C-C-C-C=O
688	C-C-C-C-C-C-C
689	O-C-C-C-C-C-C
690	O-C-C-C-C-C-O
691	O-C-C-C-C-C-N
692	O=C-C-C-C-C-C
693	O=C-C-C-C-C-O
694	O=C-C-C-C-C=O
695	O=C-C-C-C-C-N
696	C-C-C-C-C-C-C-C
697	C-C-C-C-C-C(C)-C
698	O-C-C-C-C-C-C-C
699	O-C-C-C-C-C(C)-C
700	O-C-C-C-C-C-O-C
701	O-C-C-C-C-C(O)-C
702	O-C-C-C-C-C-N-C
703	O-C-C-C-C-C(N)-C
704	O=C-C-C-C-C-C-C
705	O=C-C-C-C-C(O)-C
706	O=C-C-C-C-C(=O)-C
707	O=C-C-C-C-C(N)-C
708	C-C(C)-C-C
709	C-C(C)-C-C-C
710	C-C-C(C)-C-C
711	C-C(C)(C)-C-C
712	C-C(C)-C(C)-C

Section 7: Complex SMARTS patterns - These bits test for the presence of complex SMARTS patterns, regardless of count, but where bond orders and bond aromaticity are specific.

Bit Position	Bit Substructure
713	Cc1ccc(C)cc1
714	Cc1ccc(O)cc1
715	Cc1ccc(S)cc1
716	Cc1ccc(N)cc1
717	Cc1ccc(Cl)cc1
718	Cc1ccc(Br)cc1
719	Oc1ccc(O)cc1
720	Oc1ccc(S)cc1
721	Oc1ccc(N)cc1
722	Oc1ccc(Cl)cc1
723	Oc1ccc(Br)cc1
724	Sc1ccc(S)cc1

```
Sc1ccc(N)cc1
Sc1ccc(Cl)cc1
Sc1ccc(Br)cc1
Nc1ccc(N)cc1
Nc1ccc(Cl)cc1
Nc1ccc(Br)cc1
Clc1ccc(Cl)cc1
Clc1ccc(Br)cc1
Brc1ccc(Br)cc1
Cc1cc(C)ccc1
Cc1cc(O)ccc1
Cc1cc(S)ccc1
Cc1cc(N)ccc1
Cc1cc(Cl)ccc1
Cc1cc(Br)ccc1
Oc1cc(O)ccc1
Oc1cc(S)ccc1
Oc1cc(N)ccc1
Oc1cc(Cl)ccc1
Oc1cc(Br)ccc1
Sc1cc(S)ccc1
Sc1cc(N)ccc1
Sc1cc(Cl)ccc1
Sc1cc(Br)ccc1
Nc1cc(N)ccc1
Nc1cc(Cl)ccc1
Nc1cc(Br)ccc1
Clc1cc(Cl)ccc1
Clc1cc(Br)ccc1
Brc1cc(Br)ccc1
Cc1c(C)cccc1
Cc1c(O)cccc1
Cc1c(S)cccc1
Cc1c(N)cccc1
Cc1c(Cl)cccc1
Cc1c(Br)cccc1
Oc1c(O)cccc1
Oc1c(S)cccc1
Oc1c(N)cccc1
Oc1c(Cl)cccc1
Oc1c(Br)cccc1
Sc1c(S)cccc1
Sc1c(N)cccc1
Sc1c(Cl)cccc1
Sc1c(Br)cccc1
Nc1c(N)cccc1
Nc1c(Cl)cccc1
Nc1c(Br)cccc1
Clc1c(Cl)cccc1
Clc1c(Br)cccc1
Brc1c(Br)cccc1
CC1CCC(C)CC1
CC1CCC(O)CC1
CC1CCC(S)CC1
CC1CCC(N)CC1
CC1CCC(Cl)CC1
CC1CCC(Br)CC1
OC1CCC(O)CC1
OC1CCC(S)CC1
OC1CCC(N)CC1
OC1CCC(Cl)CC1
OC1CCC(Br)CC1
SC1CCC(S)CC1
SC1CCC(N)CC1
SC1CCC(Cl)CC1
SC1CCC(Br)CC1
NC1CCC(N)CC1
NC1CCC(Cl)CC1
```

793 NC1CCC(Br)CC1
794 ClC1CCC(Cl)CC1
795 ClC1CCC(Br)CC1
796 BrC1CCC(Br)CC1
797 CC1CC(C)CCC1
798 CC1CC(O)CCC1
799 CC1CC(S)CCC1
800 CC1CC(N)CCC1
801 CC1CC(Cl)CCC1
802 CC1CC(Br)CCC1
803 OC1CC(O)CCC1
804 OC1CC(S)CCC1
805 OC1CC(N)CCC1
806 OC1CC(Cl)CCC1
807 OC1CC(Br)CCC1
808 SC1CC(S)CCC1
809 SC1CC(N)CCC1
810 SC1CC(Cl)CCC1
811 SC1CC(Br)CCC1
812 NC1CC(N)CCC1
813 NC1CC(Cl)CCC1
814 NC1CC(Br)CCC1
815 ClC1CC(Cl)CCC1
816 ClC1CC(Br)CCC1
817 BrC1CC(Br)CCC1
818 CC1C(C)CCCC1
819 CC1C(O)CCCC1
820 CC1C(S)CCCC1
821 CC1C(N)CCCC1
822 CC1C(Cl)CCCC1
823 CC1C(Br)CCCC1
824 OC1C(O)CCCC1
825 OC1C(S)CCCC1
826 OC1C(N)CCCC1
827 OC1C(Cl)CCCC1
828 OC1C(Br)CCCC1
829 SC1C(S)CCCC1
830 SC1C(N)CCCC1
831 SC1C(Cl)CCCC1
832 SC1C(Br)CCCC1
833 NC1C(N)CCCC1
834 NC1C(Cl)CCCC1
835 NC1C(Br)CCCC1
836 ClC1C(Cl)CCCC1
837 ClC1C(Br)CCCC1
838 BrC1C(Br)CCCC1
839 CC1CC(C)CC1
840 CC1CC(O)CC1
841 CC1CC(S)CC1
842 CC1CC(N)CC1
843 CC1CC(Cl)CC1
844 CC1CC(Br)CC1
845 OC1CC(O)CC1
846 OC1CC(S)CC1
847 OC1CC(N)CC1
848 OC1CC(Cl)CC1
849 OC1CC(Br)CC1
850 SC1CC(S)CC1
851 SC1CC(N)CC1
852 SC1CC(Cl)CC1
853 SC1CC(Br)CC1
854 NC1CC(N)CC1
855 NC1CC(Cl)CC1
856 NC1CC(Br)CC1
857 ClC1CC(Cl)CC1
858 ClC1CC(Br)CC1
859 BrC1CC(Br)CC1
860 CC1C(C)CCC1

861	CC1C(O)CCC1
862	CC1C(S)CCC1
863	CC1C(N)CCC1
864	CC1C(Cl)CCC1
865	CC1C(Br)CCC1
866	OC1C(O)CCC1
867	OC1C(S)CCC1
868	OC1C(N)CCC1
869	OC1C(Cl)CCC1
870	OC1C(Br)CCC1
871	SC1C(S)CCC1
872	SC1C(N)CCC1
873	SC1C(Cl)CCC1
874	SC1C(Br)CCC1
875	NC1C(N)CCC1
876	NC1C(Cl)CC1
877	NC1C(Br)CCC1
878	ClC1C(Cl)CCC1
879	ClC1C(Br)CCC1
880	BrC1C(Br)CCC1

Decoding PubChem Fingerprints

PubChem fingerprints are currently 881 bits in length. Binary data is stored in one
byte increments. The fingerprint is, therefore, 111 bytes in length (888 bits), which includes padding of seven bits at the end to complete the last byte. A four-byte prefix, containing the bit length of the fingerprint (881 bits), increases the stored PubChem fingerprint size to 115 bytes (920 bits).

When PubChem fingerprints are encoded in base64 format, the base64-encoded fingerprints are 156 bytes in length. The last two bytes are padding so that the base64 length is divisible by four (156 bytes - 2 bytes = 154 bytes). Each base64 byte encodes six binary bits (154 bytes * 6 bits/byte = 924 bits). The last four bits are padding to complete the last base64 byte (924 bits - 4 bits = 920 bits). The resulting 920 binary bits (115 bytes) are described in the previous paragraph.

Document Version History

V1.3 - 2009May01 - Updated introduction to describe how to identify
the PubChem Substructure Fingerprint property in
a PubChem Compound record.
V1.2 - 2007Aug30 - Added section on decoding PubChem fingerprints.
V1.1 - 2007Aug06 - Corrected and expanded documentation of bits
with SMARTS patterns used.
V1.0 - 2005Dec02 - Initial release.