Machine Learning Optimization for AMES Mutagenicity Dataset

Isaac Yeoh [a]
[a] University of California, San Francisco, 94143 United States

Our group: Isaac, Miko, Xiao, and Jason, evaluated machine learning models against the AMES Mutagenicity dataset obtained from TDCommons.AI. The dataset consisted of 2 columns: SMILES and mutagenicity. we were tasked with creating models with a goal of obtaining one with an AUC score higher than 80%. We ultimately chose to make a Random Forest Classifier (RFC) and Support Vector Machine (SVM). Initially, without looking at the reference paper's data processing methods, we wanted to try creating our model using our own data processing method. Our method relied on RDKit's Descriptor Package, which split SMILES into machine readable features. However, our RFC yielded an AUC score of ~66%, which was not ideal. Hence, we switched data processing methods, settling on PaDELPy, which breaks SMILE strings down to CDK Fingerprint's 1024 features. This yielded 4 additional models: RFC with ~88% score, XGBoost with ~88% score, SVM with ~87% score, and Logistic Regression with ~83% score. We ultimately decided that our SVM was the best model due to RFC and XGBoost overfitting significantly.

Introduction

A mutagen is a chemical or physical agent that is capable of inducing changes in DNA, and is often linked to carcinogenicity (1). Some drugs that can cause said mutations are: 5-Bromouracil, o-Dianisidine, and 4-Chloroaniline. The AMES mutagenicity dataset is a collection of mutagenic and non-mutagenic compounds. It has 2 columns: SMILES and Mutagenicity. SMILES stand for "Simplified Molecular Input Line Entry System", and it is a standardized way to represent a molecule's structure using a simple string of characters that can be easily read by computers (2). On the other hand, the Mutagenicity column is denoted by 1 for mutagenic and 0 for non-mutagenic. This dataset was obtained from TDcommons.ai, an open-science initiative with AI/ML-ready datasets and AI/ML tasks for therapeutics, spanning the discovery and development of safe and effective medicines (3). The AMES mutagenicity dataset is widely used in computational toxicology and cheminformatics to predict chemical safety. It also supports the development of models to reduce reliance on animal testing. Its applications are broad, encompassing toxicity prediction and feature engineering in drug discovery.

Our dataset consists of 7,255 drugs' SMILES and mutagenicity, which is labelled Y. Figure 1 illustrates a snipet from the dataset:

	Drug_ID	Drug	Y
0	Drug 1	O=[N+]([O-])c1c2c(c3ccc4cccc5ccc1c3c45)CCCC2	1
1	Drug 2	O=c1c2ccccc2c(=O)c2c1ccc1c2[nH]c2c3c(=O)c4cccc...	0
2	Drug 3	[N-]=[N+]=CC(=O)NCC(=O)NN	1
3	Drug 4	[N-]=[N+]=C1C=NC(=O)NC1=O	1
4	Drug 6	CCCCN(CC(O)C1=CC(=[N+]=[N-])C(=O)C=C1)N=O	1
5	Drug 7	[N-]=[N+]=CC(=O)OCC(N)C(=O)O	1
6	Drug 9	CC(=O)OC1(C(C)=O)CCC2C3C=C(Cl)C4=CC(=O)OCC4(C)...	0
7	Drug 10	Nc1nc(N)nc(N)n1	0
8	Drug 11	Cc1ccc(N=Nc2c(O)ccc3ccccc23)c([N+](=O)[O-])c1	1
9	Drug 12	CC(C)CC(=O)Nc1snc2ccccc12	0

Figure 1: Snippet of AMES mutagenicity dataset

Through TDC's tox package, the split data was obtained. TDC has already pre-split the dataset into train, validation, and test sets for their users with a split percentage of 70%, 20%, and 10% respectively. The dataset distribution indicates that 55% of the compounds are classified as mutagenic while the remaining 45% are non-mutagenic.

The paper reference that the AMES mutagenicity refers to is "In silico prediction of chemical Ames mutagenicity" by Xu, Conying, et al. (4). In their paper, they described using 5 different machine learning. The five machine learning models employed include Support Vector Machines (SVM), decision trees, artificial neural networks (ANN), k-nearest neighbors (kNN), and naïve Bayes (NB). Our group opted to create a model independently, without referencing the paper's data processing methods, to try to create an unbiased model. From the 5 choices, we opted out of kNN due to briefly covering it in class, having learned kMeans instead. Similarly, an ANN model was not chosen since at the time of our discussion, we have not learned about neural networks yet. Moving on to NB, in class, we learned that all features are assumed to be independent, which results in a weakness in learning relationships between features. So NB was definitely not an appropriate model to consider given our data's circumstance.

Thus, we narrowed our options to SVM and a decision tree model. A SVM is a supervised machine learning algorithm that classifies data by finding an optimal line or hyperplane that maximizes the distance between each class in an N-dimensional space (9). A decision tree is a non-parametric supervised learning algorithm, and it has a hierarchical, tree structure, which consists of a root node, branches, internal nodes and leaf nodes, suitable for both classification and regression tasks (10).

However, we felt that a single decision tree was too simple. Therefore, we opted for an Random Forest Classifier (RFC) instead. The RFC is a commonly-used machine learning algorithm that combines the output of multiple decision trees to reach a single result (5). Theoretically, the more trees in a RFC, the higher its accuracy should be. Users are also able to extract important features from RFCs and rank them by how important the features are.

Our overall objective was to create a model that was capable of predicting mutagenicity fairly accurately, with a goal of a score greater than 80%.

RDKit's Descriptor Package

Data Extraction and Processing

Our group discovered that RDKit's Descriptor Package provided an excellent foundation for our project. With the descriptor package, we were able to extract 5 features which we found to be important: Morgan Fingerprint with radius 1 (FpDensityMorgan1), Morgan Fingerprint with radius 2 (FpDensityMorgan2), Morgan Fingerprint with radius 3 (FpDensityMorgan3), Exact Molecular Weight, and the average molecular weight of the molecule ignoring hydrogens (Heavy Atom Count). Morgan fingerprints enable mapping of certain structures of the molecule within certain radius of organic molecule bonds (6). The algorithm only considers the immediate neighbors of each atom in the radius when generating the fingerprint value. Figure 2 illustrates what the Morgan algorithm considers given different radius.

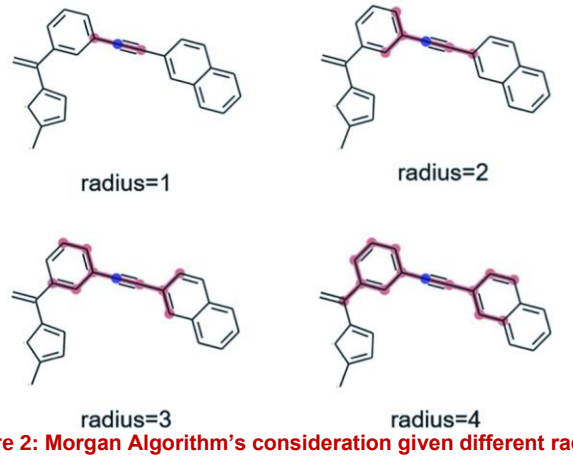

Figure 2: Morgan Algorithm's consideration given different radius values

There were more features that we were able to extract from RDKit's Descriptor Package. However, we decided that the other features were irrelevant or simply redundant for our model. The low number of features presented a challenge. Due to that, we decided to move on with the RFC instead of the SVM.

In addition to the feature extraction, our group validated the mutagenic column by checking that the values in the column only have 0s and 1s; filtering out all values that are not these numbers. We also validated the SMILES strings by passing them through RDKit's Chem package, and filtering out the invalid strings using an if statement.

RDKit's RFC Model

With the data processing done, the features were simply fit against the mutagenic outputs for each drug in the RFC. With 1000 decision trees used as the only adjusted hyperparameter, the RFC was created. Figure 3 notes the ROC AUC scores obtained by the model. Our train set's ROC AUC obtained a score of 0.984, validation with 0.655, and test with 0.667.

Our group decided to use ROC AUC because it compares the relation between the true positive rate (sensitivity) and the false positive rate (1 – specificity) across different thresholds. It is also one of the best metrics for binary classification, and is ideal for evaluating RFCs where class probabilities are used to make decisions.

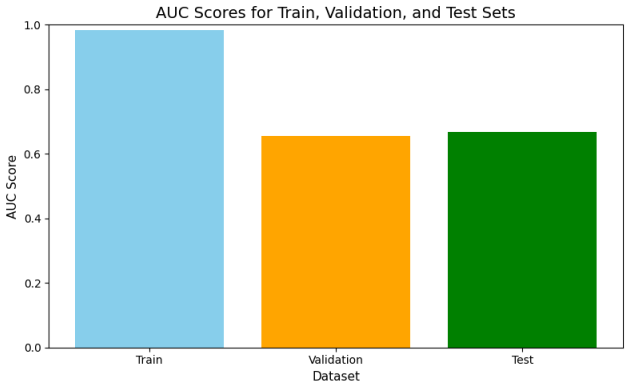

Figure 3: AUC scores for train, validation, and test sets on RDKit's RFC

On the same note, the F1 scoring metric was not chosen because it focuses solely on the positive class and combines precision and recall, ignoring the true negative rates. Precision and Recall were not chosen due to their limited utility in showing false negative impacts and false positive impacts respectively. Finally, even though our dataset is relatively balanced (55%-45%), accuracy can still obscure issues like disproportionate false positives or false negatives.

Due to the high train score as compared with the validation score, it is certain that our model is overfitting. The low test score also indicates that the features used to trian our model was most likely not impactful enough. Thus, our group had to revise our strategy to create a better model.

PaDELPy Package

Data Extraction and Processing

Failing to use RDKit, our group turned back to the paper by Conying, et al. (4) for data extraction hints. They processed their data using PaDELPy, which breaks SMILE strings down into machine readable features called fingerprints. The 5 fingerprints Conying, et al. extracted were: CDK fingerprints (FP), Estate fingerprints (Estate), MACCS keys (MACCS), PubChem fingerprints (PubChem), and Substructure fingerprints (SubFP). However, our group decided to just focus on one fingerprint: CDK, since the paper obtained its highest performing model using CDK.

PaDELPy's descriptor function accepts .smi files as inputs, so we had to convert our train, validation, and test splits from pandas' data frame files to .smi files. Among the features available in PaDELPy's descriptor function, we selected detect aromaticity, standardized nitro, standardized tautomers, removed salt, and most importantly, set fingerprints to true. Detect aromaticity enables detection of aromaticity (like benzene) in molecules; standardized nitro enables detection of aromaticity (like -NO2) in molecules; standardized tautomers standardize molecules that can interconvert by rearranging bonds and atoms; fingerprints specifies that molecular fingerprints be calculated in addition to descriptors.

Feature Interpretability

One major downside of PaDELPy is the interpretability of its features. Since PaDELPy breaks the SMILE strings down using a java bash script, it is not possible to know how it breaks the SMILE strings through reading the code. Figure 4 illustrates a sample PaDELPy CDK fingerprint breakdown of SMILE strings.

	A	B	C	D	E	F	G	H	I	J
1	Name	FP1	FP2	FP3	FP4	FP5	FP6	FP7	FP8	FP9
2	AUTOGEN_train_molecule_1	0	0	0	0	0	0	0	0	0
3	AUTOGEN_train_molecule_2	0	0	0	0	0	0	0	0	0
4	AUTOGEN_train_molecule_3	0	0	0	0	0	0	0	0	0
5	AUTOGEN_train_molecule_4	0	0	0	0	0	0	0	0	0
6	AUTOGEN_train_molecule_5	0	0	0	1	0	0	0	0	0
7	AUTOGEN_train_molecule_6	0	0	0	0	0	0	0	0	0
8	AUTOGEN_train_molecule_7	0	0	0	1	0	1	0	0	0
9	AUTOGEN_train_molecule_8	0	1	0	0	0	0	0	0	0
10	AUTOGEN_train_molecule_9	0	0	0	0	0	0	0	0	0
11	AUTOGEN_train_molecule_10	0	0	1	0	0	0	0	0	0
12	AUTOGEN_train_molecule_11	0	0	0	0	0	0	0	0	0
13	AUTOGEN_train_molecule_12	0	0	0	0	0	0	0	0	0
14	AUTOGEN_train_molecule_13	0	0	0	0	0	0	0	0	0
15	AUTOGEN_train_molecule_14	0	1	0	0	0	0	1	0	0
16	AUTOGEN_train_molecule_15	1	0	0	0	0	0	0	0	0
17	AUTOGEN_train_molecule_16	0	0	0	0	0	0	0	0	0
18	AUTOGEN_train_molecule_17	0	1	1	0	0	0	1	0	1
19	AUTOGEN_train_molecule_18	0	0	1	0	0	0	0	1	0
20	AUTOGEN_train_molecule_19	0	0	0	0	0	0	0	0	0
21	AUTOGEN_train_molecule_20	0	0	0	0	0	0	0	0	0
22	AUTOGEN_train_molecule_21	0	0	0	0	0	0	0	0	0
23	AUTOGEN_train_molecule_22	0	1	0	0	0	0	0	0	0
24	AUTOGEN_train_molecule_23	0	1	0	0	0	0	0	0	0

Figure 4: CDK Fingerprints of SMILE strings using PaDELPy

CDK fingerprints break the SMILE strings down into 1024 features. However, not all features are necessarily useful for the model's training. Our approach to this problem was filtering out features with low variance values.

	Feature1	Feature2	Feature3	Feature4	Feature5
0	1.0	0.0	5.2	0.0	2.3
1	1.0	0.0	5.4	0.0	2.5
2	1.0	0.0	5.6	0.0	2.4
3	1.0	0.0	5.8	0.0	2.6
Variance:	0	0	0.1	0	0.015

Figure 5: Calculating variance

Variance is a measure of dispersion, meaning it is a measure of how far a set of numbers is spread out from their average value (7). A sample calculation is provided in Figure 5. Our group decided to filter out values with variance lower than 0.1 for the best selection of features for our model, which narrowed our 1024 features down to 519 features.

Group Model Evaluations

With feature extraction out of the way, our group was finally able to build models. Trying our hand at the RFC again, we were able to obtain significantly better results this time as seen in Figure 6.

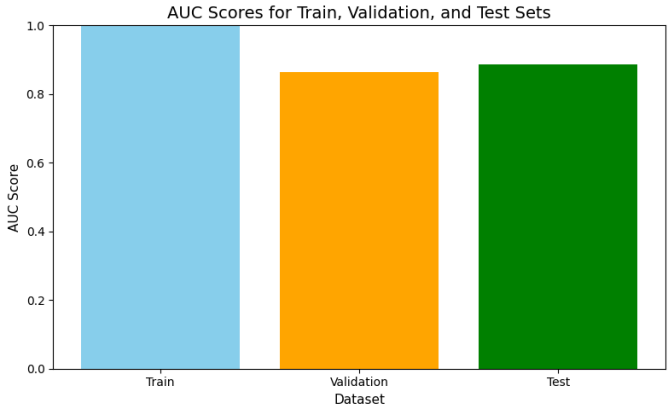

Figure 6: RFC ROC AUC scores using PaDELPy

With a train ROC AUC of 99.8%, Validation ROC AUC of 86.5%, and 88.6%, our model was once again overfitting. This time, it was expected since we used 3000 decision trees as the only hyperparameter to train the model. We tried implementing more hyperparameters, such as adjusting the max depth of the tree, the minimum leaf samples, and more, which managed to lower our train AUC to a reasonable level. However, this caused our test and validation ROC AUC to drop significantly. Hence, we decided not to adjust any more hyperparameters outside of the number of decision trees, and stuck with an overfitting RFC.

After observing the strong performance of the RFC using PaDELPy's descriptor breakdown of the SMILES strings, our group decided to proceed with the SVM. Given that there were over 500 features, we were confident that the SVM would effectively handle the high-dimensional feature space. Figure 7 shows how our SVM performed at different learning rates.

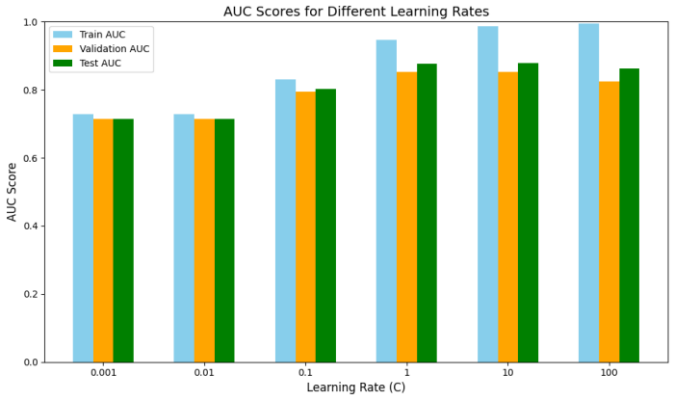

Figure 7: SVM with RBF kernel scores at different learning rates' scores

The SVM with different learning rates were all trained on the RBF kernel since it produced the best scores. As seen by figure 7, learning rates 1 and 10 seem to have the best test and validation results: learning rate 1 with a test AUC of 87.6% and validation AUC of 85.3%, while learning rate 10 with a test AUC of 87.7% and validation AUC of 85.2%. However, the training AUC in learning rate 10 is significantly higher than the training AUC in learning rate 1. Therefore, we decided that the SVM with learning rate 1 was still a better model despite having a slightly lower test AUC score.

Individual Model Evaluations

To determine whether the RFC is truly the best decision tree model, I tested an XGBoost tree on the feature-extracted data generated by PaDELPy. XGBoost, short for Extreme Gradient Boosting, is a scalable and distributed gradient-boosted decision tree (GBDT) library. It offers parallel tree boosting and is widely regarded as a top machine learning library for tasks such as regression, classification, and ranking (8).

While both random forest and GBDT rely on ensembles of decision trees, they differ in how these trees are constructed and combined. Random forest employs a technique called bagging, where full decision trees are trained in parallel using random bootstrap samples of the dataset, and the final prediction is the average of all tree predictions. In contrast, GBDT trains an ensemble of shallow trees iteratively, with each subsequent tree focusing on the residual errors of the previous model. The final prediction is a weighted sum of all tree predictions (8).

Random forest's bagging approach reduces variance and overfitting, while GBDT's boosting approach reduces bias and underfitting. Unlike traditional GBDT, XGBoost builds trees in parallel rather than sequentially, enhancing efficiency and scalability (8). Figure 8 shows the results obtained from fitting the ~520 features obtained from PaDELPy on a XGBoost Tree at different learning rates.

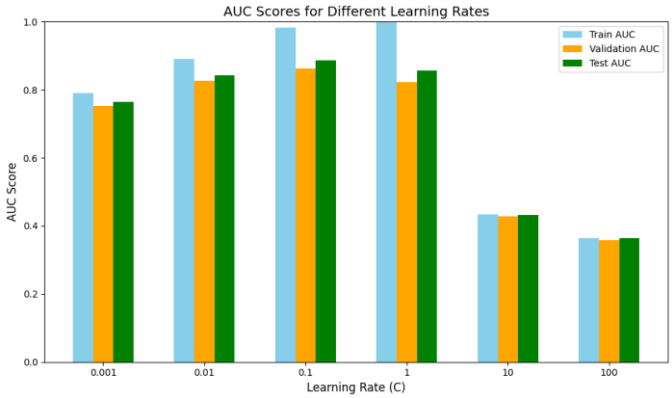

Figure 8: XGBoost at different learning rates' scores

Similar to the SVM, the XGBoost allows the users to adjust the learning rates to boost the model's performance and tackle overfitting. Unlike the RFC, I used 1000 tree estimators since the RFC heavily overfitted on 3000. However, as noticed by figure 8, I was still getting relatively high train AUC scores compared to vlidation AUC scores. Overall, the best learning rate obtained for this model is the one with learning rate 0.1 with train AUC of 98.3%, validation AUC of 86.3%, and test validation of 88.4%.

Another model that I was curious about was the logistic regression model. Our group disregarded it earlier since we believed that it was simple of a model to produce meaningful results. However, it is a great model to be used on binary classification data, which is what the AMES mutagenicity dataset is. Hence, I fit my features from PaDELPy onto a logistic regression model and obtained scores seen in Figure 9.

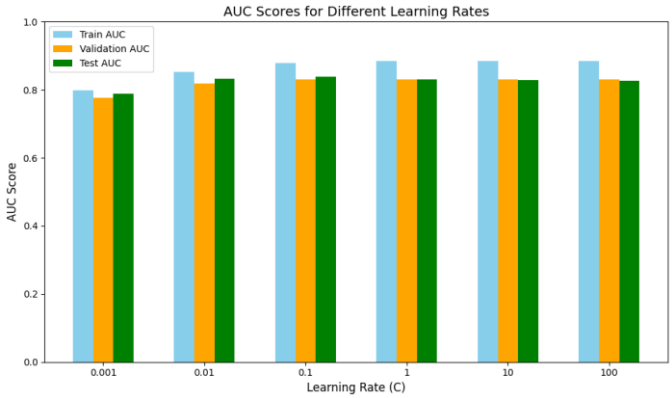

Figure 9: Logistic Regression with L2 penalty at different rates' scores

One important hyperparameter tuning which I did to the logistic regression model was including a L2 penalty. A L2 penalty, also known as ridge regression, adds a "squared magnitude" of coefficient as penalty term to the loss function (11). This basically prevents overfitting. As seen by the train AUCs across all learning rates in Figure 9, they are all relatively similar to the validation AUC, which means the model is not overfitting. However, the scores produced by the logistic regression model was not the best. With a learning rate of 0.1, the best logistic regression model had a train AUC of 88.0%, validation AUC of 83.3%, and test AUC of 83.6%.

Discussion

Model	Test ROC AUC Score	Train ROC AUC Score	Validation ROC AUC Score	Train - Val Score
SVM (PaDELpy)	87.6%	94.6%	85.4%	9.2%
Random Forest Classifier (PaDELpy)	88.6%	99.8%	86.5%	13.3%
XGBoost (PaDELpy)	88.4%	98.3%	86.3%	12%
Logistic Regression (PaDELpy)	83.6%	88.0%	83.3%	4.7%
Random Forest Classifier (RDKit)	66.7%	98.4%	65.5%	32.9%

Figure 10: Overall AUC Scores

Figure 10 displays the results obtained from all models, group and individual, ranked from best to worst. Our worst model in 5th place was the RFC trained using RDKit's descriptor feature extraction, having the lowest test and validation scores, along with an extremely high train AUC score. In 4th place, we have the logistic regression from PaDELPy, which is a significant increase compared to the RFC trained on RDKit. It is the model which had the least amount of overfitting, with only a 4.7% difference between the train and validation scores. In 3rd place, we have the XGBoost tree, with a test AUC score of 88.4%, but had a high train – validation score of 12%, pertaining to overfitting. The RFC trained on PaDELPy's feature extraction and filtering was placed 2nd. With a test AUC of 88.6% and validation AUC of 86.5%, it was our best model in terms of prediction. However, it is also one of the highest overfitting models in the PaDELPy group, with a train – validation score of 13.3%. Finally, in 1st place, our group's SVM had the best overall performance. Despite having a lower test and validation AUC than RFC and XGBoost, we decided that this model was the best simply because of its lower train –

validation score, indicating that it isn't overfitting as much compared to the RFC and XGBoost tree. Overall, the group's model did better than my individual models.

Model	Validation	External Validation
MACCS-kNN	0.824	0.956
PubChem-kNN	0.824	0.970
FP-SVM	0.844	0.903
MACCS-SVM	0.853	0.924
PubChem-SVM	0.853	0.949

Figure 11: Xu, Conying, et al.'s Scores

Comparing our results to the results obtained by Xu, Conying, et all. in Figure 11, our models were relatively decent, but not as impressive as theirs.

Evaluating our models, our biggest weakness is our overfitting issue. As explained earlier, hyperparameter tuning significantly affected our test and validation scores even though it brought our train AUC scores down. We can certainly improve them given more time on the project. One possible method is to combine the feature extractions from RDKit and PaDELPy together for a more robust model training process. After doing so, we can implement hyperparameter training such as limiting the maximum depth of the tree, to make the models less prone to overfitting while hopefully optaining a higher train, val, and test ROC AUC score. A more time inefficient method to improve our models would be to test hyperparameters individually, extracting those that perform better than our current model's scores. We can also test our model against other fingerprints such as EState, MACCS, and Pubchem instead of just CDK for PaDELPy.

References

1. Mutagen. Genome.gov. Published 2024. Accessed December 11, 2024. https://www.genome.gov/genetics-glossary/Mutagen
2. US. SMILES Tutorial | Research | US EPA. Epa.gov. Published 2024. Accessed December 11, 2024. https://archive.epa.gov/med/med_archive_03/web/html/smiles.html#:~:text=What%20is%20SMILES?,learn%20a%20handful%20of%20rules.
3. Papers with Code - tdcommons Dataset. Paperswithcode.com. Published 2022. Accessed December 11, 2024. https://paperswithcode.com/dataset/tdcommons#:~:text=tdcommons%20(Therapeutics%20Data%20Commons)&text=Discovery%20and%20Development-,Therapeutics%20Data%20Commons%20is%20an%20open%2Dscience%20initiative%20with%20AI,of%20safe%20and%20effective%20medicines.
4. Xu C, Cheng F, Chen L, et al. In silico Prediction of Chemical Ames Mutagenicity. Journal of Chemical Information and Modeling. 2012;52(11):2840-2847. doi: https://doi.org/10.1021/ci300400a
5. IBM. Random Forest. Ibm.com. Published October 20, 2021. Accessed December 12, 2024. https://www.ibm.com/topics/random-forest
6. Medin D. Data Science for drug discovery research -Morgan fingerprints in Python. Medium. Published November 2, 2022. Accessed December 12, 2024. https://darkomedin-datascience.medium.com/data-science-for-drug-discovery-research-morgan-fingerprints-using-alanine-and-testosterone-92a2c69dd765
7. Variance - Wikipedia. Wikipedia.org. Published 2016. Accessed December 12, 2024. https://en.wikipedia.org/wiki/Variance
8. What is XGBoost? NVIDIA Data Science Glossary. Published 2019. Accessed December 12, 2024. https://www.nvidia.com/en-us/glossary/xgboost/
9. IBM. Support Vector Machine. Ibm.com. Published December 12, 2023. Accessed December 12, 2024. https://www.ibm.com/topics/support-vector-machine
10. IBM. Decision tree. Ibm.com. Published November 2, 2021. Accessed December 12, 2024. https://www.ibm.com/topics/decision-trees
11. Nagpal A. L1 and L2 Regularization Methods - Towards Data Science. Medium. Published October 13, 2017. Accessed December 13, 2024. https://towardsdatascience.com/l1-and-l2-regularization-methods-ce25e7fc831c

GitHub Link

- https://github.com/Socks2109/CDD203_FP