1. Problem Introduction

The field of Nanotechnology has been experiencing substantial growth rates for a few years now. While nanotechnology is currently being applied to a broad range of problems across different fields such as Energy, Electronics, Medicine, etc. I wanted to get a glimpse of what nanotechnology problems are currently being studied and came across the problem of toxicity. Since nanoparticles are extremely small, especially those with sizes less than 100 nanometers. These nanoparticles can easily enter and lodge themselves in organs within the human body. Consequently, causing damage or even killing cells within the body also known as cytotoxicity. Thus, for this project I will be aiming to develop a few different classification Machine Learning models, assessing their performance through different metrics and optimizing to achieve the best possible results.

2. Data Introduction

For this project I will be working with the Nanoparticle Toxicity Dataset found on Kaggle. This dataset contains about 800 records which are composed of several features including: core size, hydrodynamic size, surface area, surface charge, electric charge, dosage, exposure time, number of oxygen atoms, and energy (possibly surface energy). The dataset also contains a class which takes values of Toxic and nonToxic. I have a few concerns about this dataset, one of them being that it does not contain any information regarding units of measurement. Another concern is that this dataset can raise possible misinterpretation, mainly due to the classification of Toxic or nonToxic. This classification can mean different things given a different context, for example a toxic nanoparticle may only be toxic to a specific cell type such as cancer cells in which case this classification would be desired.

Additionally, if the nanoparticle is toxic to benign cells within the body, then this is an unwanted result.

3. Data-Preprocessing

This dataset is relatively clean, as expected from a Kaggle dataset. I began by assessing the missing value count, which was 0 in this case. The only issue I ran into with this dataset was that many records were duplicated; after removing the duplicates only 1/3 of the original data was left. Additionally, this dataset contains one categorical feature which signifies the Chemical compound of the nanoparticle, I opted to use this feature for my models rather than simply removing it. Thus, I had to perform one-hot encoding to make them usable predictors in the models.

4. Visualization

As I suspected from preprocessing the data, dropping the duplicated records significantly changed the distribution of toxic and non-toxic nanoparticles. We can see in Figure-1 that non-toxic nanoparticles outnumber the toxic nanoparticles by a sizeable amount, this will certainly cause the model to perform better on classifying non-toxic nanoparticles.

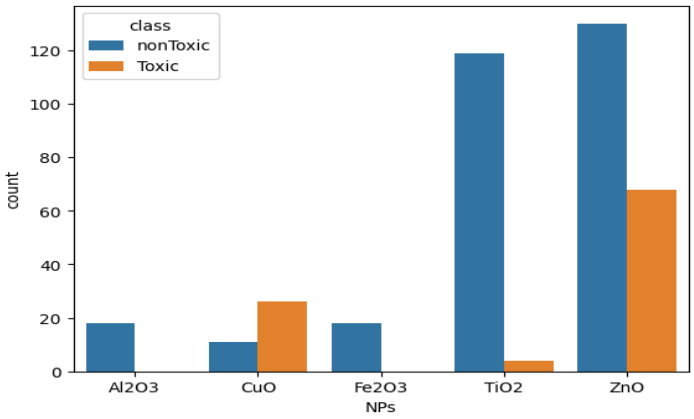
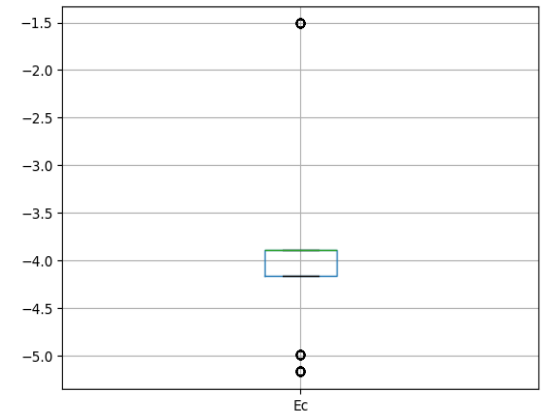
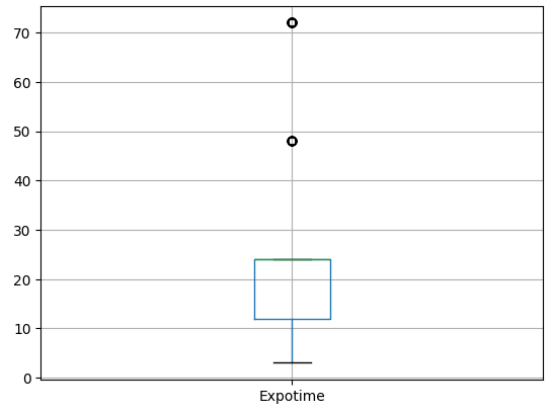
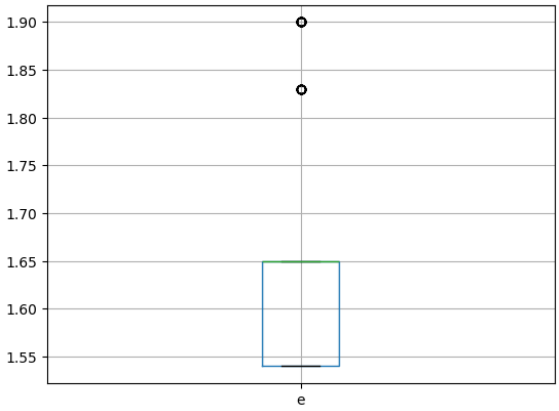


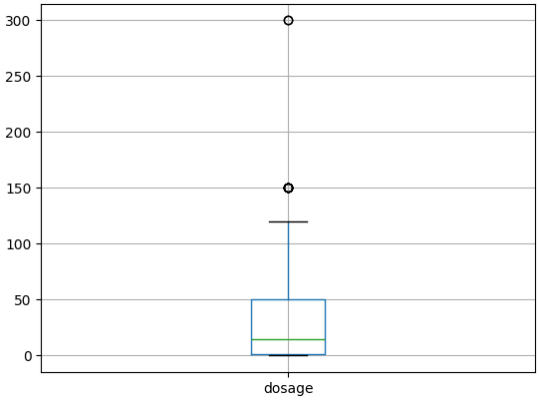
Figure 1: nanoparticle distribution across classes

I was also interested in viewing which features may have outliers, since the nanoparticle toxicity dataset is severely limited and highly favoring the non-toxic class, outliers could also introduce additional problems leading to highly inaccurate predictions. Consequently, I will be cross-validating several scaling transformations provided by Scikit-learn including RobustScaler which may work better with the outliers. The following boxplot figures show those features which have several outliers.









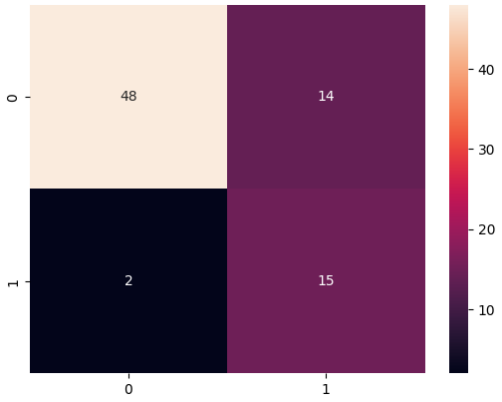
5. Modeling and Evaluation

5.1. Methods

For this project, I wanted to explore several classification models which require the use of many different hyper-parameters. To better streamline the process, I opted to use Scikit-learn Pipelines which allowed me to assemble a series of steps from which I can cross-validate all combinations by using Scikit-learn's GridSearchCV. Additionally, by using GridSearchCV I was able to determine which parameters provide the best test set performance. For cross-validation, I used several model-specific hyperparameters along with different types of scaling such as MinMaxScaler, StandardScaler and RobustScaler. Additionally, since the non-toxic class carried the most weight initially, I attempted to balance the classes by changing their weights during training.

5.2. Logistic Regression Classifier

The first model I assessed was a Logistic Regression, which as I expected would be very fast given its simplicity and the small dataset which is just over 350 samples. However, this model had the worst performance of the three I examined. As can be seen from its confusion matrix, it incorrectly classified a good percentage of the non-toxic class as toxic. And incorrectly classified two toxic nanoparticles as nontoxic which has a large cost compared to misclassifying non-toxic nanoparticles.

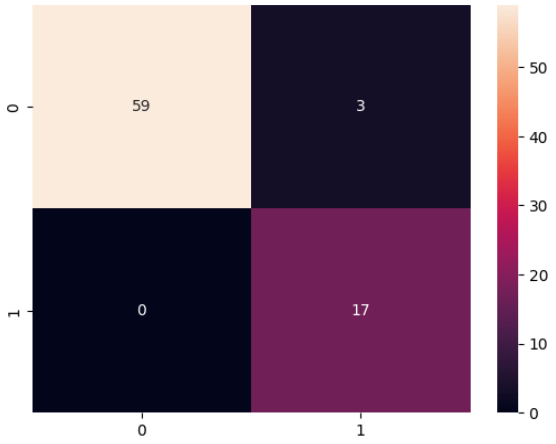


AUC: 0.828

	precision	recall	f1-score	support
0	0.96	0.77	0.86	62
1	0.52	0.88	0.65	17
accuracy			0.80	79
macro avg	0.74	0.83	0.75	79
weighted avg	0.86	0.80	0.81	79

5.3. Random Forest Classifier

The second model I tested was a Random Forest Classifier; I expected much better performance from this model although it came at the cost of much longer computation time. This model achieved the highest accuracy ranging from 96% to 98%. It did not misclassify any toxic nanoparticles which carry the highest cost (Type 2 error). It did misclassify a few non-toxic nanoparticles as toxic (Type 1 error) however; this is not as costly as making Type 2 errors.

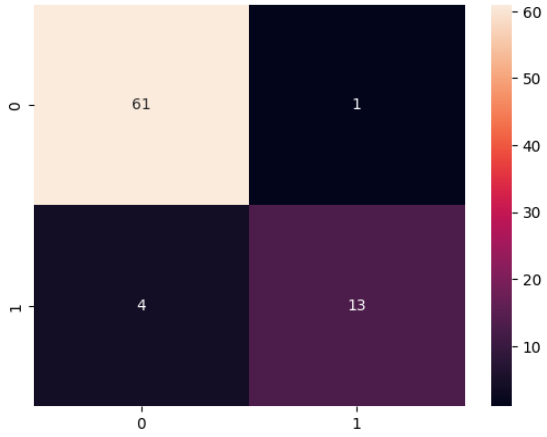


AUC: 0.976

	precision	recall	f1-score	support
0	1.00	0.95	0.98	62
1	0.85	1.00	0.92	17
accuracy			0.96	79
macro avg	0.93	0.98	0.95	79
weighted avg	0.97	0.96	0.96	79

Support Vector Classifier

The last model I evaluated was a Support Vector Classifier; I opted to use this model for its ability to separate high-dimensional data. Since the distribution of features in this dataset highly overlapped, it might perform better with different kernels than the previous models. While it did outperform the Logistic Regression, it did not outperform the Random Forest. Although, this model is much more computationally efficient than the Random Forest. However, while this model achieved an accuracy of 94% it did make a few Type 2 errors even after shifting the class balance towards the toxic class.



AUC: 0.874

	precision	recall	f1-score	support
0	0.94	0.98	0.96	62
1	0.93	0.76	0.84	17
accuracy			0.94	79
macro avg	0.93	0.87	0.90	79
weighted avg	0.94	0.94	0.93	79

6. Impact

The nanoparticle toxicity classification problem can certainly have different impacts, depending on the context. As I mentioned before, depending on the application of the nanoparticle, making a Type 2 (False Negative) error could have severe consequences since it could directly affect a life. Therefore, more specialized models should be developed based on the desired outcome. Lastly, I am confident that this dataset is not sufficient to accurately classify diverse nanoparticles as it only contains 5 different types of nanoparticles.

Sources

UCI Machine Learning. (2024, July 22). *Nanoparticle Toxicity Dataset*. Kaggle. https://www.kaggle.com/datasets/ucimachinelearning/nanoparticle-toxicity-dataset/data