University of the Philippines Manila
College of Arts and Sciences
Department of Physical Sciences and Mathematics

NaTTA: *In-silico* Classification of Nanotoxicity Using QSAR-Perturbation Based Model.

A special problem in partial fulfillment
of the requirements for the degree of
Bachelor of Science in Computer Science

Submitted by:

Heidi A. Puato

June 2023

Permission is given for the following people to have access to this SP:

Available to the general public	Yes
Available only after consultation with author/SP adviser	No
Available only to those bound by confidentiality agreement	No

ACCEPTANCE SHEET

The Special Problem entitled "NaTTA: *In-silico* Classification of Nanotoxicity Using QSAR-Perturbation Based Model." prepared and submitted by Heidi A. Puato in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

Perlita E. Gasmen, M.Sc. (*cand.*)
Adviser

EXAMINERS:

	Approved	**Disapproved**
1. Avegail D. Carpio, M.Sc.	_____	_____
2. Richard Bryann L. Chua, M.Sc.	_____	_____
3. Ma. Sheila A. Magboo, Ph.D. (*cand.*)	_____	_____
4. Vincent Peter C. Magboo, M.D.	_____	_____
5. Marbert John C. Marasigan, M.Sc. (*cand.*)	_____	_____
6. Geoffrey A. Solano, Ph.D.	_____	_____

Accepted and approved as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

Vio Jianu C. Mojica, M.Sc.
Unit Head
Mathematical and Computing Sciences Unit
Department of Physical Sciences and Mathematics

Marie Josephine M. De Luna, Ph.D.
Chair
Department of Physical Sciences and Mathematics

Maria Constancia O. Carrillo, Ph.D.
Dean
College of Arts and Sciences

Abstract

Application of nanoparticles (NPs) in many different fields brought several benefits, especially in biomedicine, physics, chemistry, and agriculture. However, nanoparticles can exhibit toxic effects due to their very high surface-to-volume ratio causing harm to biological systems and to their respective ecosystems. Nanotoxicity testing is an important phase to determine the potential risks that NPs may bring. On the other hand, the process to perform experimental assays often requires quite a lot of time and resources. An alternative way to perform nanotoxicity testing is through in-silico testing. In-silico methods are usually centered around the quantitative structure-activity relationship (QSAR) modeling, however in this work, we refined the predictive capacity of QSAR modeling by integrating the Perturbation Theory, which was a recent novel method of testing nanotoxicity. Using different machine learning algorithms, this work developed a QSAR-perturbation model to predict toxicity profiles of NPs under diverse experimental conditions. The models were developed from a dataset of 5,437 NP-NP pairs derived from applying perturbation theory to 260 unique NPs. In the results, XGBoost was the top performing model with 98.43% MCC value. This is comparable to previous results in existing literature. The QSAR-PT model was then employed in a web application as a final output.

Keywords: Nanoparticle, Nanotoxicity, QSAR model, Perturbation Theory

Contents

List of Figures

List of Tables

I. Introduction

A. Background of the Study

Even before its popularity and known modern applications, nanotechnology has been part of the world's history as early as the fourth century AD. Relics, such as the Lycurgus cup made by the roman civilizations, showed usage of materials containing nanoparticles that causes the cup to exhibit dichroism [1]. On the other hand, modern nanotechnology was formally introduced in 1959 by Richard Feynman. Since then, continuous studies had been conducted which made it possible for its applications in various discipline today [2].

Nanotechnology is defined as the understanding and practical application of materials that ranges from 1 to 100 nanometers called nanomaterials or nanoparticles [3]. Due to their minute structure, the properties of nanoparticles (nanoparticles) can have drastic alterations which enables nanostructured materials to have specific performance or have new properties [4]. In line with this, production of engineered nanomaterials (ENMs) for applications in biomedicine, agriculture, electronics, and even in energy saving, became extensive in hopes of providing better solutions and precise and customized treatments [5].

However, despite the immense benefits that ENMs provide, nanoparticles can exhibit toxic effects due to their very high surface-to-volume ratio [6]. Nanoparticles are extremely reactive. Moreover, they can penetrate cellular membranes and biological barriers, and therefore poses a risk to biological systems and to their respective ecosystems [6]. There are different ways to measure toxicity based on the information one wants to gain. To measure toxicity based on cell viability CC_{50} is used. TC_{50} is used to know how much concentration will cause toxicity. Meanwhile, to know the concentration of nanoparticles that will prevent root elongation of a plant IC_{50} is used, which is also known as the half maximal inhibitory concentration. These three

are just some examples on how toxicity is measures [7].

Ways to address the issue of nanotoxicity in the past were mainly composed of in vivo and in vitro techniques. This means that toxicity test were performed on cell cultures or in living organisms [8]. However, these methods are often expensive and time-consuming, especially when one needs to obtain permissions to conduct the tests [8]. To offer an alternative way, in-silico testing were introduced for cheaper and more simpler process of testing nanotoxicity.

In-silico testing are computer-aided verification to classify whether a certain NP is toxic or not. By implementing machine learning algorithms, predictive models that could classify the toxicological profile of an NP were develop. Through this method, cost can be minimized by reducing the need for conventional trial and error testing done in laboratories [9].

To perform in-silico testing, the methodologies used are usually centered around the quantitative structure-activity relationship (QSAR) modeling. QSAR modeling is a method to predict the biological activity or toxicity of substances with the application of mathematical statistics and knowledge of machine learning [10]. Essentially, the concept of QSAR is that toxic effects of a substance can be predicted from its relating molecular descriptors [10].

In relation to QSAR modeling, many past efforts have already tried implementing the method. However, most of these studies are based on classical concepts and resorted to linear analyses [7]. According to [11], majority of published literature applied the method only against a single biological system. Additionally, results from classical QSAR modeling were shown to be inadequate. Hence, predicting using nano-specific features are gaining momentum [11].

A very recent, promising way to refine the predictive capacity of QSAR modeling and to address its one-to-one limitation is the integration of perturbation theory in QSAR models [12]. In QSAR perturbation models, it introduces a case-case pairs

in which one case is used as a reference state, while the other is taken as the predicted output. The variables used to predict toxicity are called descriptors [7]. These descriptors are the differences in the dynamics with regards to the physicochemical properties and experimental conditions of the nanoparticles being classified [12]. Through this technique, a specific case can be predicted based from the differing experimental conditions a nanoparticle is subjected to.

In this project, a QSAR perturbation model was developed to facilitate an in-silico classification of toxicological profiles of nanoparticles. The study used a dataset from a previous research by [7]. The dataset was composed of 54,371 data points derived from 260 unique nanoparticles. Each entry in the dataset included atleast one of the five toxicity measures - IC_{50}, CC_{50}, EC_{50}, TC_{50}, and LC_{50}. However, this study aims to do a comprehensive examination on the nanoparticles that were measured using IC_{50}, which is a measure of concentration of nanoparticles that will inhibit 50% root elongation of plants. Using different machine learning algorithms, as well as a deep learning methodology, this project developed a QSAR perturbation model that addressed the limitation of classical QSAR modeling. In this way, risk assessment of nanoparticles will be more economical and safe for both the researcher and the environment.

B. Statement of the Problem

Current studies that support development of in-silico nanotoxicity classification use classic QSAR modeling, which has a major limitation of making the analysis linear. Therefore, most of the published literature were only focused on one-to-one approach. In this study, the researcher explored the capabilities of the novel QSAR-perturbation based models. To the best of the researcher's ability, there are only extremely limited available studies found that conducted classification of nanotoxicity using the said novel method. The limited existing studies only employed Latent Dirichlet Allocation

(LDA) [13] [14] [15] and Artificial Neural Network (ANN) [7] algorithms in the models they created. Thus, there are no known system yet that uses the discussed novel methodology while using other machine learning algorithms.

C. Objectives of the Study

The main objective of this study is to develop a QSAR-perturbation based model that could facilitate in-silico classification of nanotoxicity. Additionally, the final output of this project is an application that allows its user to classify an NP as toxic or non-toxic. Specifically, the study performed the following:

1. Use only the data that uses IC_{50} as measure of toxicity in the new state from the dataset taken from [7].
2. Address the class imbalance using SMOTE.
3. Perform data normalization.
4. Split the dataset into 80% training set and 20% testing set.
5. Perform classification using Support Vector Machine (SVM), Classification and Regression Tree (CART), Extreme Gradient Boosting (XGBoost), and Artificial Neural Network (ANN).
6. Perform feature importance using SHAP.
7. Integrate the top performing resulting models in an application that can do the following actions:

   The system:

   (a) Compute for the perturbation values based from the input data of the user and the conditions set for the reference nanoparticle.

(b) Perform classification (non-toxic or toxic).

The user:

(a) Enter needed information (molar volume, size, electronegativity, and spectral moments based on different measurements) of the sample nanoparticle.

(b) Enter the experimental conditions (type of NP, toxicity measure, endpoint, shape, condition when size was taken, assay time, and coating agent) of the reference nanoparticle.

(c) View the resulting toxicity classification.

(d) View the performance metrics of the classification models (accuracy, precision, sensitivity, specificity, area under the curve (AUC), and Matthews Correlation Coefficient (MCC).

D. Significance of the Project

Traditional trial and error testing of nanotoxicity are often expensive, time-consuming, and demands a great deal of effort to perform. In order to alleviate this situation, in-silico testing is an alternative way to perform nanotoxicology testing. By applying machine learning algorithms, toxicity assessments of nanoparticles can be done more economical and user-friendly.

In another perspective, the output of this project may be used by future scientists in their training in nanochemical testing. Additionally, the project may provide help in the training of future professionals in using intelligent machines.

Academically, this paper will contribute in the growing knowledge of nanotechnology, as well as insight on the capabilities of the novel QSAR-pertubration based model.

E. Scope and Limitations

1. The study only used a part of the dataset taken from a previous study by [7].
2. The part taken from the dataset are the data points that uses toxicity measurement IC_{50} in the new state; wherein a value of greater than or equal 245.68 is classified as non-toxic, meanwhile lower than the given value is considered toxic.
3. The study only used perturbation values as descriptors for classification.
4. The study only perform binary classification (toxic or non-toxic).
5. The study only used existing available python package/library tools for model building and system integration.

F. Assumptions

The following points are the assumptions regarding the system application that will be developed for the model created.

1. The nanoparticle of interest uses IC_{50} as the measurement of toxicity.
2. The combination of inputs in the reference nanoparticle exists in the database of the application.
3. The input data are not invalid inputs.

II. Review of Related Literature

Nanotechnology is considered as one of the most promising technologies of the 21st century [3]. Since the formulation of its idea in 1959, the continued research and steady investigations had led this scientific discipline to its popularity and modern applications today [16]. Nanotechnology is defined as the application of the manipulation of materials at a nanoscale [3]. Nanomaterials are the products of nanotechnology. It is mainly composed of particles in the range of 1-100 nanometers. Due to their very small structure, scientists were able to engineer these nanoparticles to have specific purpose and to exhibit new properties. Hence, it found its way in several different applications in the industry [4].

At present, engineered nanomaterials can be encountered in many aspects in the daily life. In food and agriculture, [17] showed the current applications of ENMs. In food they are used as food additives, flavor enhancers, preservatives, and even as toxin detector. On the other hand, in agriculture ENMs are used as fertilizers and pesticides. Nanotechnology also has huge contributions in biomedicine, especially during the outbreak of the COVID-19 pandemic. Using nanotechnology, drug delivery were enhanced to navigate biological microenvironments, as well as to transcend biological barriers in order to deliver therapeutics to specific cells and tissues in the body [18]. Moderna and Pfizer-BioNTech Covid-19 vaccines are some of the popular examples of this application at present [18]. There are many other applications of the technology in other scientific disciplines, hence production of ENMs became extensive [5].

Together with the productions and manipulations of new nanoparticles, serious concerns regarding the applications of these particles have also risen in the last two decades [8]. Because of their small structures, these nanoparticles can easily pass through cell membranes and boundaries [6]. Moreover, they have very high surface-to-volume ratio making them extremely reactive. Hence, these particles can exhibit toxic effects to its intended endpoint [6]. In a study by [19], nanotoxicity can affect

its environment in three ways: (1) it can directly affect species, (2) it can cause transformation in biological entities and activities upon contact with other pollutants, and (3) it can provoke the non-living environment to undergo structural changes. Therefore, it is necessary to perform nanotoxicity tests and risk assessments of these particles before their massive production and release in the market.

Toxicity tests and risk assessments are usually done in laboratories using traditional in-vivo and in-vitro means [20]. In these methods, testing are either done using cell cultures in glass containers (in-vitro) or using live samples such as animals, plants, and whole cells (in-vivo). Evaluation through these methods can be expensive and time-consuming. In order to perform such tests, one needs to consider the amount of trials they needed to obtain substantial information. Furthermore, if the experiment will require live samples the researchers would need to obtain permissions from governing bodies and could also raise potential ethical issues [10]. As a result, alternative ways of testing were sought and one of them is in-silico testing.

In-silico nanotoxocity testing is the result of using computational power in the classification of the toxicological profiles of nanoparticles. Equipped with abundant amount of data from previous traditional nanotoxicological experiments, these data can now be utilized for creation of in-silico based testings. In this way, the expensive and meticulous process of toxicity and risk assessment will be reduced [9]. There are several known methods in performing in-silico testing, but the most popular technique used by majority of published literatures were quantitative structure-activity relationship (QSAR) modeling [11].

The concept of QSAR modeling is that physiological activity of particles can be inferred from their associated chemical compositions [10]. In this context, toxic effects of nanoparticles can be modelled as a function of molecular descriptors that a nanoparticles has. Molecular descriptors are the physicochemical properties of a molecule, this include their molar volume, electronegativity, polarizability, size, spec-

tral moments, etc..

An example of a study that employed QSAR modeling was conducted by [21]. The study used metallic nanomaterials in order to predict its ecotoxicological effects on *Daphnia magna.* Using six different supervised machine learning algorithms, the researchers built predictive models that utilized a dataset split in 60:40 ratio. Results showed that the top performing algorithm was the Random Forest with an accuracy of 87% , specificity of 94%, and an AUC value of 96%.

In an another study by [22], they performed in-vitro and in-silico toxicity testing of metal oxide nanoparticles to *Escherichia coli.* For the in-silico phase of their testing, the researchers used Multiple Linear Regression (MLR). Their study found that model 2 from seven models that they have created produced the best results when pharmacological approach was considered. This meant that metallic oxide nanoparticles induce cytotoxicity in *E. coli* as a result of an 'individual action' when in mixture rather than as an 'additive effect' or 'synergistic effect'.

An in-silico approach was also applied in a toxicity classification study of superparamagnetic iron oxide nanoparticles (SPIONs) in stem-cell monitoring [23]. In the study, the researchers used Auto-ML to perform nano-QSAR modeling. Nano-QSAR modeling is similar to the classic QSAR with the exception of using nanofeatures as variables to base the prediction. In the evaluation of the developed model, its performance indicated a value of 91% accuracy, 93% precision, 91% recall, and 91% f1-score. Further analysis showed that the attributes that heavily influenced the outcome of the study were the physicochemical properties such as the size and magnetic core of the nanoparticles. There are many more published literature that used QSAR modeling, some of them used metal oxides nanoparticles and predicted their toxicity using Monte Carlo optimization [24], and partial least squares regression [25] [26].

Despite the robust results of present NP risk assessments, classic QSAR modeling has a major limitation of only producing one-to-one results. There is a diverse number

of bioindicators that nanoparticles may have the chance to interact with. Synergy between these entities results to innumerable possibilities of biological behaviors [12]. Hence, a challenge to cover these unaccounted activities gave rise to a recent novel method of testing nanotoxicity, which is the integration of perturbation theory to QSAR.

Perturbation theory is a method used to improve an approximation by calculating errors or deviations from an initial solution [12]. In line with this, its integration in the classic QSAR model will allow the in-silico nanotoxicity testing to have multiple input conditions and produce predictive models for a variety of response targets [27].

In QSAR-Perturbation model or PT-QSAR model , the structure of the paradigm is replaced with case-case pairing from a one-to-one biological effect and a reference chemical pairing [12]. This means that in the novel method, the pair consist of one chemical used as a reference, and a second chemical used to calculate the molecular descriptors and eventually be classified [7].

To the researcher's knowledge, the very first published PT-QSAR model for nanotoxicity testing was by [13] in 2014. In the study, the researchers used the model to predict cytotoxicity in nanoparticles under diverse experimental conditions. Their dataset consist of 41 nanoparticles and generating the case-case pairing, the resulting final dataset has 1681 pairings. Then, it was randomly split into 75:25 ratio for training and testing. To develop the predictive model, linear discriminant analysis (LDA) was used together with a forward step-wise procedure for selecting variables . Through this method, they were able to come up with a model equation that exhibited the highest statistical significance with as few descriptors as possible. Performance metrics showed that their equation 4 achieved the most promising results, with an accuracy of 93.58% and an AUC value of 0.983. As an extension of the study, [14] also used the same PT-QSAR model to test for ecotoxicity and cytotoxicity of uncoated and coated nanoparticles.

In a similar study by [7], the researchers also used the same PT-QSAR model in an attempt to create a unified predicitve model for nanotoxicity in nanoparticles. Using a currated dataset from different previous assessments, the researchers used artificial neural network as the ML algorithm to develop their model. As a results, the study was able to obtain an overall accuracy of 98.5% and an AUC value of 0.999.

In a more recent study by [15], they also implemented a PT-QSAR model to predict genotoxicity of metal oxide nanoparticles. In their study, they used 78 unique nanoparticles to generate 6084 NP-NP pairs. Similar to [13] and [14], the ML algorithm used was also LDA and as a result they were able to have an accuracy of 97.81%.

To the best of the researcher's ability, only these reviewed articles were found to have used the novel method in developing an in-silico classification model for nanotoxicity assessment. Hence, the full capabilities of the said method is still under discovery and further investigation. Given with the concepts behind nanotechnology, the serious concerns that comes with it, and the methodologies to address the issue of toxic effects, this study would like to expound more on the performance of applying computational power in the toxicity and risk assessment of nanoparticles, most specifically in using the novel method PT-QSAR modeling.

III. Theoretical Framework

A. Nanotoxicity

Nanotoxicity is the harmful effects that nanoparticles can have on their intended endpoint. Since nanoparticles are very small, it can be manipulated to have unique physicochemical properties in order to be utilized for a specific purpose. However, their small size makes them highly reactive which in turn can cause damage as a result of toxicity. Thus, toxic and risk assessments are a must in order to come-up with safe-by-design nanomaterials before its production and release. Nanotoxicity can be measured depending on the information one want to obtain.

B. In-silico Nanotoxicological Testing

In-silico Nanotoxicological testing is a process of evaluating nanoparticles for their toxic effects by utilizing computational power to perform simulations. In this course, the process of toxic and risk assessment of nanoparticles are made simpler to perform. To carry out this method, using qualitative structure-activity relationship (QSAR) model is the most common approach. However, assessments done using this model produces one-to-one results. Classic QSAR model outputs are usually only against one biological target, and is limited to just one experimental condition. Hence, information garnered from using this method lacks assessment powers if one is to consider the diverse possible experimental conditions and biological entity that an NP can be tested against.

C. QSAR-Perturbation Model

QSAR-Perturbation model or PT-QSAR model is the resulting technique of integrating the perturbation theory in classic QSAR modeling. Using this model, the limitation of classic QSAR was addressed. By generating NP-NP pairs, the classification

of the nanoparticles are extended to consider the different experimental conditions it is under and the several bio-target they are tested against. In [12], they provided a protocol that can be followed in performing this novel method.

$$\Delta \mathrm{DD}_i(c_j) = \mathrm{DD}_i(c_j)_{\mathrm{new}} - \mathrm{DD}_i(c_j)_{\mathrm{ref}}$$

$$\Delta \mathrm{DG}\mu_k(\mathrm{PP}) = G\mu_k(\mathrm{PP})_{\mathrm{new}} - G\mu_k(\mathrm{PP})_{\mathrm{ref}}$$

Figure 1: Equation for getting perturbation values.

To simplify the protocol, the main features used in this model are perturbation values or simply the differences of the properties of each NP-NP pairs. Figure 1 are the equations used to calculate the perturbations values between each pair of nanoparticles.

On the other hand, figure 2 represents the QSAR part of the model, which depicts that the classification of the target variable is a function of the classification of the reference NP, and their perturbation values.

$$\mathrm{Tox}_i(c_j)_{\mathrm{new}} = f\left[\mathrm{Tox}_i(c_j)_{\mathrm{ref}}, \Delta \mathrm{DD}_i(c_j), \Delta \mathrm{DG}\mu_k(\mathrm{PP})\right]$$

Figure 2: QSAR equation.

D. Data Pre-processing

D..1 Synthetic Minority Oversampling Technique (SMOTE)

SMOTE is an oversampling technique used to address class imbalance in the dataset. Using SMOTE, the minority class in the dataset were increased by generating synthetic samples. The data generated by using this technique are not duplicates of the original dataset, but slightly different ones; hence the term 'synthetic' [28]. SMOTE works by drawing a random sample from the minority class and using these to create

a slightly different data points. Then, these synthetic data points are added to the dataset to make it balanced [29].

E. Machine Learning Algorithms

E..1 Support Vector Machine

Support Vector Machine (SVM) is a machine learning algorithm that is mainly used for classification, regression, and outlier detection [30]. The goal of SVM is to find a hyperplane that will segregate the support vectors in order to produce the most optimal classification in an N-dimensional space (N-number of features) [31]. The hyperplane can be multiple lines/boundaries depending on the number of features used in the dataset. Support vectors are the data points that are closest to the hyperplane. The position of these point will determine the location of the hyperplane.

E..2 Classification and Regression Tree

Classification and Regression Tree (CART) is a supervised machine learning algorithm typically used to predict a target value from learned decision rules inferred from the data [32]. Essentially, it is a decision tree, wherein nodes are split into sub-nodes based on a threshold value. However, its difference with DT is that CART uses the best homogeneity value as the basis of splitting rather than a threshold value of an attribute [33]. To obtain the homogeneity value, CART uses Gini Index, wherein it calculates the impurity for all possible splits. The split with the lowest Gini index or impurity is deemed as the best split [34].

E..3 Extreme Gradient Boosting

Extreme Gradient Boosting (XGBoost) is an open-source library that provides efficient implementation of gradient boosting [35]. It is a tree-based algorithm that can be used for classification and also regression problems [36]. In XGBoost, trees are

built in parallel, unlike in most gradient boosting algorithms where its is built sequentially [37]. Generally, XGBoost is an algorithm that build models by combining the predictive power of multiple weaker ones in order to output a highly accurate prediction and improve the robustness of the models.

E..4 Artificial Neural Network

Artificial Neural Network (ANN) is a deep learning algorithm, which is a subset of machine learning. ANN is a computational model that tries to mimic how the human brain works, hence human intervention are minimized [38]. ANN is composed of three node layers that are interconnected with each other. The input layer, which consist the input neurons, accepts the data and directly pass it onto the hidden layer. The hidden layer are the inner layers that is responsible for receiving the data and will perform the information gathering by weighing it according to the ANN's internal system. Lastly, the output layer is where the desired predictions are obtained [39]. By going through these layers, predictions are made as the weighted sum of the neurons in the neural network.

F. Performance Metrics

F..1 Matthews Correlation Coefficient (MCC)

MCC is a correlation coefficient between the predicted values and the true values. It is usually used to summarize the confusion matrix from a resulting classification model. In order to have a good classification model, the resulting MCC value must be close to 1. This will also be the main metric that will be used in evaluating and comparing the models to determine the best performing model.

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

F..2 Area Under the Curve (AUC)

AUC is a metric used to evaluate the performance of the model across possible classifications. With AUC, it can show the probability of the model to rank a random positive example higher than a random negative example.

F..3 Accuracy

Accuracy is the ratio of correct predictions to the total number of input samples used in the classifier.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

F..4 Precision

Precision is the ratio of correct true positive predictions to the total number of correct predictions made by the classifier.

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

F..5 Sensitivity / Recall

Sensitivity is the true positive rate or the proportion of true positive predictions that are actually predicted correctly.

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

F..6 Specificity

Specificity is the proportion of true false predictions that are actually predicted correctly.

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

G. Feature Importance

G..1 SHapley Additive exPlanations (SHAP)

SHAP is a tool mainly used to determine the impact of a feature to the resulting value of the target variable. It uses a model-agnostic approach, thus it is applicable to use in any type of model in measuring the importance of each features used. In SHAP, the dataset can be considered as the team and each feature is a player. Each of the players have contribution to the team, and the resulting average absolute value of their contribution can be considered as their importance to the team. Hence, SHAP is a tool mainly used to add global or/and local explainability to a machine learning model.

IV. Design and Implementation

A. Dataset

The dataset for this project was taken from [7]. It is composed of 260 unique nanoparticles and was paired with each other to generate the NP-NP pairs. Only about 80% of the resulting pairs were used by the original study. In this project, it only consider the NP-NP pair that used (IC_{50}) as its measure of toxicity in the new state. Hence, the final dataset will be composed of 5,437 NP-NP pairs. The dataset has 10 descriptors that were used as features for the creation of the models. Each descriptor, except for the dummy classifier, are perturbation values computed from the NP-NP pairings. Table 1 shows the description of each descriptors.

B. Model Implementation

To achieve the goal of this project, there are three main phases that constitutes the main processes done in this study.

The first phase of the project is pre-processing. An exploratory data analysis was conducted on the dataset to get to know the data and gain insights on the possible correlations of the variables with each other. The dataset also has a class imbalance that could result to low model performance. In order to address this issue, SMOTE was applied. Additionally, normalization of values was also done to ensure that no feature intrinsically influence the prediction due to difference in range. Finally, the dataset was split into 80% training set and 20% testing set.

The second phase is the model building. In here, four ML algorithms was used namely, SVM, CART, XGBoost, and ANN. All developed models were evaluated using the performance metrics discussed in the previous chapter. Additionally, feature importance using SHAP was also employed to determine the descriptor that has the greatest impact on the classification.

	Descriptors	Concept
1	$DDV(m_e)$	The change of the molar volume between the nanoparticles used, being dependent on the measures of the toxic effects.
2	$DDL(m_e)$	Variation of size between the nanoparticles, being dependent on the measures of toxic effects.
3	$DD\mu_1(ATO)b_t$	Difference of the spectral moment of order 1 (weighted by atomic weight) between nanoparticles used, being dependent on the bio-target.
4	$DD\mu_3(POL)n_s$	Change of the spectral moment of order 3 (weighted by polarizability) between nanoparticles used, being dependent on the shape of nanoparticles.
5	$DDE(d_m)$	Variation of the electronegativity between the nanoparticles used, being dependent on the experimental condition the size of the nanoparticles were measured.
6	$DD\mu_3(VAN)t_a$	Difference of the spectral moment of order 3 (weighted by atomic van der Waals radius) between nanoparticles, being dependent on exposure times.
7	$DD\mu_2(ATO)t_a$	Difference of the spectral moment of order 2 (weighted by atomic weight) between nanoparticles, being dependent on exposure times.
8	$DG\mu_2(HYD)s_c$	Difference of general spectral moment of order 2 (weighted by hydrophobicity).
9	$DG\mu_5(PSA)s_c$	Difference of general spectral moment of order 5 (weighted by polar surface area).
10	$Tox_i(c_j)_r$	Dummy classifier that describes the toxic effect of the reference NP.

Table 1: The descriptors used in the PT-QSAR model.

The last phase of this project was the application development which was be the final output of the project. The application is a web-based app, and an I/O system that enable users to enter the needed properties of their NP which then classified as toxic or non-toxic.

Figure 3 shows the general flow of the processes done to come up with the final output for this project.

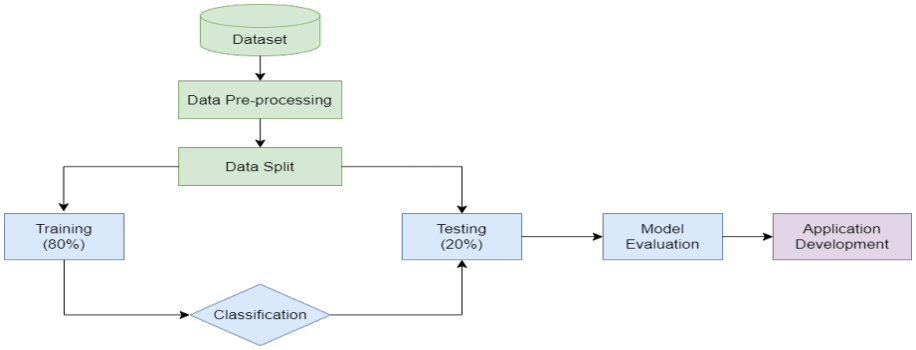


Figure 3: Flowchart of the model implementation.

C. Use Cases

The resulting application is a system tailored for scientists/researchers whose expertise lies in nanotechnology. They are the targeted users for the output of this study since they are the ones who has access to the data needed to use the application. Moreover, they also has the abilities to interpret the resulting prediction to generate useful information.

The use-case diagram in figure 4 shows the following actions that the user can do to interact with the application.

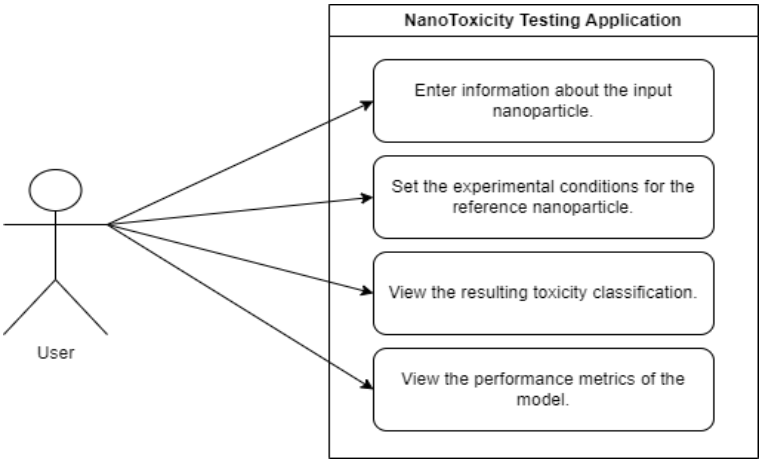


Figure 4: Use-case diagram of the system.

D. System Architecture

The summary of the input and output of the system is shown in the context diagram in figure 5.

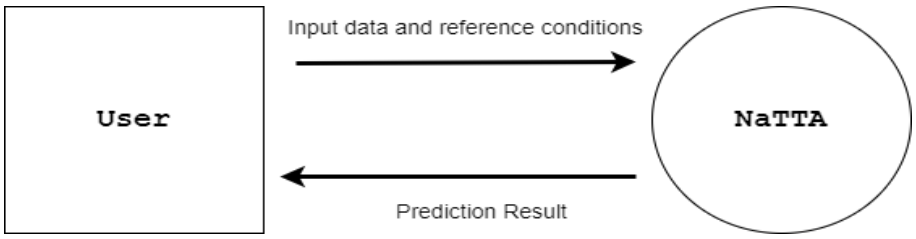


Figure 5: I/O context diagram.

In the process of the application development, the following are the frameworks and modules used.

D..1 Django

Django is an open-source web framework that is mainly used for easy development of web applications. It is written in high-level Python language and promotes rapid development and pragmatic design of applications.

D..2 Scikit-Learn

Scikit-learn is a Python module which is available for free. It is mainly used for data analysis and machine learning. The module is simple to use and has efficient tools which can help users whose domain of expertise is outside of data science. In this project, this python module was used to develop the models for XGBoost, CART, and SVM.

D..3 TensorFlow

TensorFlow is another open-source platform mainly used for machine learning. This module has a rich system that can manage all aspects of machine learning. In par-

ticular, this project used "tf.keras", which is a variant of TensorFlow, in developing the model for ANN.

E. Technical Architecture

The system is built using the following components:

- Visual Studio Code as IDE
- Python 3.10.11
- Django 4.2.1
- Scikit-learn 1.2.2
- TensorFlow 2.12.0

V. Results

A. Date Pre-processing

As part of the standard process of performing machine learning classification, the dataset taken from the study of [7] was subjected to data pre-processing. To start the pre-process the dataset was checked for possible correlations between each feature variables. This was done in order to explore the dataset and gain an insight to what does the data look like and have an idea on the possible effects it will have on the resulting models. From figure 7, it can be seen that most of the variables have strong positive correlations with one another. The variables which has the strongest positive correlation were the third variable and the seventh variable with a 0.77 Pearson r value. Meanwhile, there are also some variables that exhibit strong negative correlation. The seventh variable and the tenth variable has a correlation value of -0.66 Pearson r.

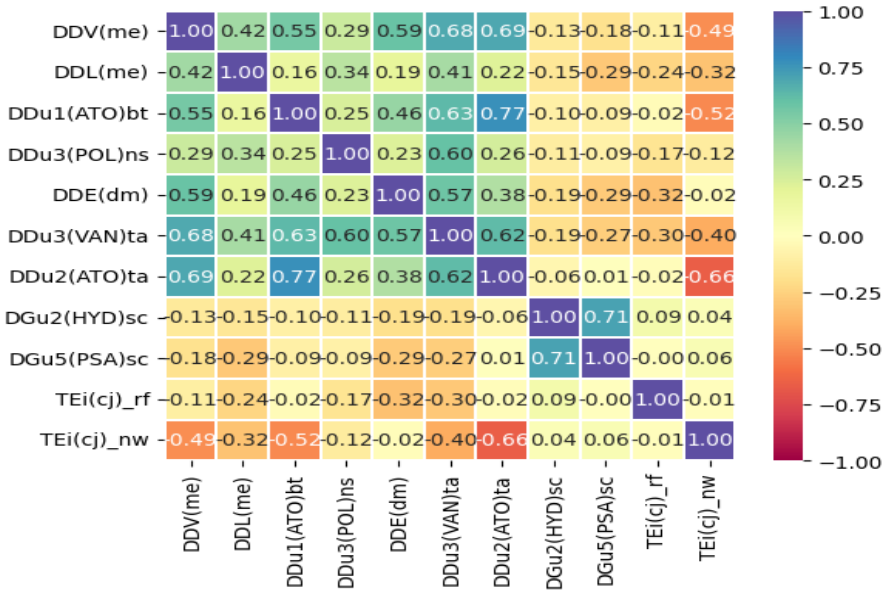


Figure 6: Correlation coefficient heatmap.

After checking for correlations, the data was partitioned into an 80-20 split. The 80% of the data was used for training the models, while the 20% was used for testing

the resulting trained model. The final dataset used in the study was composed of 5,437 observations, wherein 4,781 were non-toxic while the remaining 656 were toxic. Given these information, it is highly evident that there is a great imbalance between the two classes. In order to address the uneven distribution of classes, SMOTE was applied to even the number of toxic and non-toxic data points in the training set. Table 2 summarizes the split and the distribution of classes in the dataset.

	Non-toxic	Toxic
Train Set	3,840	509
SMOTE-enhanced train set	3,840	3,840
Test Set	941	147

Table 2: Train and Test Set Class Distributions

B. Model Building and Evaluation

After the pre-processing techniques were performed, the study proceeded in its second phase. The classifiers used in the study were XGBoost, SVM, CART, and ANN. Each classifiers were trained twice. In the first instance of training, the classifiers were trained using the train set without SMOTE, while the second round of training used the SMOTE-enhanced train set. After the models were built, the resulting performance metrics of each models were compared. The models that used the imbalanced train set were compared to those that used the SMOTE-enhanced set. Moreover, each models were also compared to one another. These comparisons were done in order to determine the effect of SMOTE on the predictive capabilities of the models, as well as to identify which model achieved the highest performance and thus the best performing model. To make the comparison, the performance of the models were evaluated using the different performance metrics mentioned in the previous chapters. The main metric that will be used to base the best performing model is the Matthew's Correlation Coefficient (MCC). The MCC score tells how much the model is capable in distinguishing between classes; however it will only generate a high quality score

if a model was able to correctly predict a high percentage of negative and positive instances of data.

B..1 Model Performance without SMOTE

For the first round of training the models, the train set without SMOTE was used. Table 3, shows the summary of the performance metrics achieved by each classifiers. From the data presented in the table, XGBoost was the best performing model that attained an AUC value of 98.19% and an MCC value of 97.23%. Then, it was followed by SVM with an AUC and MCC value of 98.60% and 96.12%, respectively. Although, the SVM model has a higher AUC score than XGBoost, the difference was very little compared to their MCC scores. Therefore, XGBoost was rendered the best model.

Metrics	SVM	XGBoost	CART	ANN
MCC	96.12%	97.23%	94.44%	91.98%
AUC	98.60%	98.19%	96.67%	94.63%
Accuracy	99.08%	99.36%	98.71%	98.16%
Precision	99.68%	99.47%	99.05%	98.42%
Recall	99.26%	99.79%	99.47%	99.47%
Specificity	97.96%	96.60%	93.88%	89.80%

Table 3: Summary of performance metrics without using SMOTE.

B..2 Model Performance with SMOTE

For the second round of training, the models were trained using the SMOTE-enhanced train set. From table 4, XGBoost remains the top performing model out of the four classifiers. It boasted a 99.21% and 98.43% AUC and MCC value, respectively. Additionally, looking at the other performance metrics it also achieved the highest value out of all the other models.

Comparing the models obtained from the first round of training and the models trained with SMOTE, the resulting predictive models from the training with SMOTE

Metrics	SVM	XGBoost	CART	ANN
MCC	94.36%	98.43%	97.26%	94.49%
AUC	98.63%	99.21%	98.77%	97.25%
Accuracy	98.62%	99.63%	99.36%	98.71%
Precision	99.78%	99.79%	99.68%	99.26%
Recall	98.62%	99.79%	99.57%	99.26%
Specificity	98.64%	98.64%	97.96%	95.24%

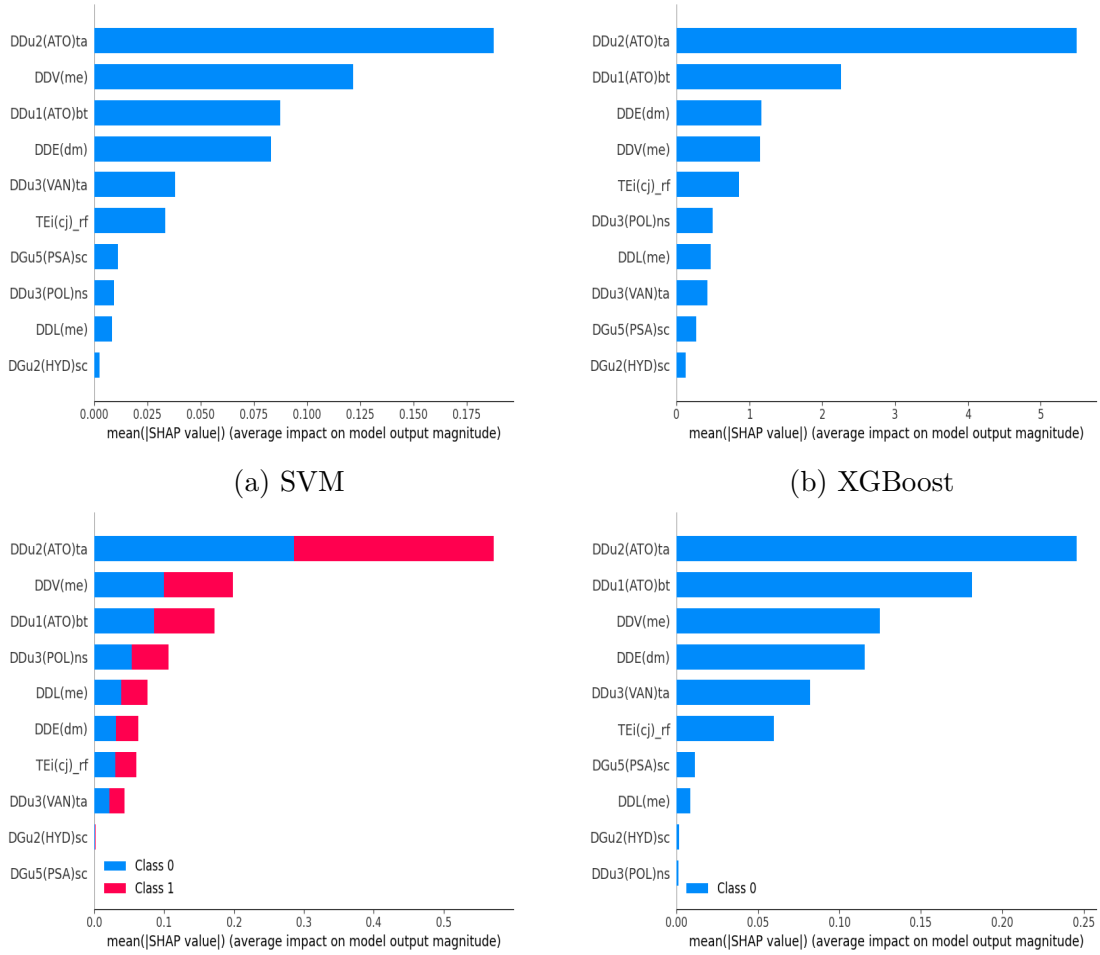
Table 4: Summary of performance metrics with SMOTE.

produced better results. Although, in some models the value of their MCC scores were higher without SMOTE, the other performance metrics of the models with SMOTE implies otherwise. Among all the models that was developed, XGBoost with SMOTE was considered to be the best performing model which garnered the highest MCC scores.

C. Feature Importance

Aside from the evaluation of the models, determining the impact of each features in the resulting decision of each model was also part of the second phase. This process was done through feature importance. In the study, impact of each features was measured by employing SHAP. Through SHAP, the study was able to determine the global importance of each feature and check which one has the most impact in the resulting value of the target variable.

Figure 8 shows the resulting feature importance derived from performing SHAP. The most significant feature for all four classifiers was $DD\mu_2(ATO)t_a$, the change in spectral moment of order 2 weighted by atomic mass in reference to the assay time. Meanwhile, the second important feature differ from some of the models. In SVM and CART, the second decisive variable was $DDV(m_e)$, while for XGBoost and ANN it was $DD\mu_1(ATO)b_t$. However, even though there was difference in the order of the top important features it was not that different from each other. In fact, it mostly differ only by two to three levels. Nonetheless, through SHAP it was determined that



(a) SVM

(b) XGBoost

(c) CART

(d) ANN

Figure 7: Feature importance using SHAP

the most decisive and important feature across the four models were the same.

D. Web Application

After the model building and evaluation, the last phase of the study was application development.

D..1 Home Page

The home page is the first page or the landing page that the user of the application will encounter when they navigated to the web application. The home page contains

the title of the application and a brief introduction on its background, as well as its purpose. At the bottom of the introduction, a button that says "Let's Test" can be clicked to navigate to the tester.

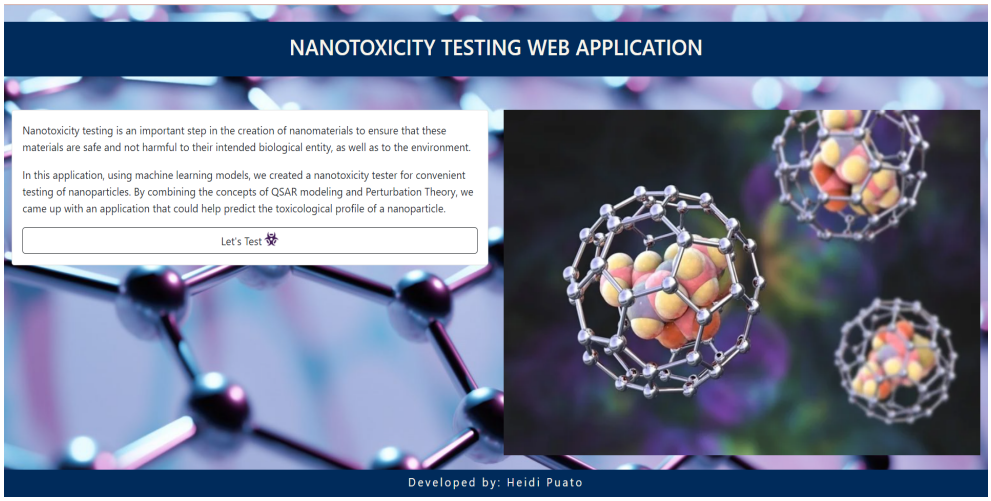


Figure 8: Home Page.

D..2 Test Page

When the user of the application clicked the button from the home page, they will be directed to the test page. The test page contains the forms where the user will enter the information of their nanoparticle. The left box on the page contains the fields to be filled out by the user for the information about the nanoparticle they want to classify. In each label of the field, a small question mark icon can be clicked to enable a popover that shows a short description of what the fields are. On right side of the page contains the box that houses the fields that prompt the users for the conditions they want their reference particle to be subjected to. Once the user had filled out all the fields, the gray button can be clicked to finally get a prediction. On the top part of the page is a navbar that can be used to navigate to the other pages of the application. The "Home" leads to the home page, "Test" leads to the test page, and "About" leads to about page.

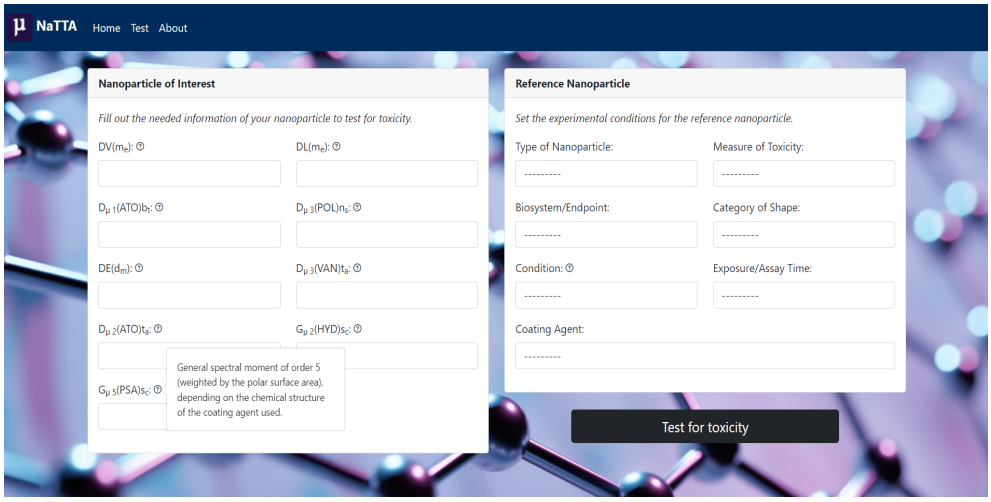


Figure 9: Test Page.

D..3 Result Page

Once the user had clicked the gray button from the test page, they will be redirected to the result page once the application completed the classification in the backend. The result page consist also of two boxes. The left box is the classification results from the XGBoost model, while the right box is the classification results from the CART model. Both boxes contains the same set of information. The resulting classifications were shown first to the user. The word "non-toxic" in green capital letters were written if the resulting prediction was non-toxic, meanwhile "toxic" in red capital letters were shown for toxic predictions. After the resulting classification, the performance metrics of the model were also shown to the user, as well as the image of the ROC curve. This will provide the user information on how well the models were trained.

D..4 About Page

In order for the users to gain more information about the purpose and the concepts behind the application, the users may navigate to the about page which contains these information. The users can go to the about page from the navbar. In the about page, the users can read the explanations of the concepts behind the application that

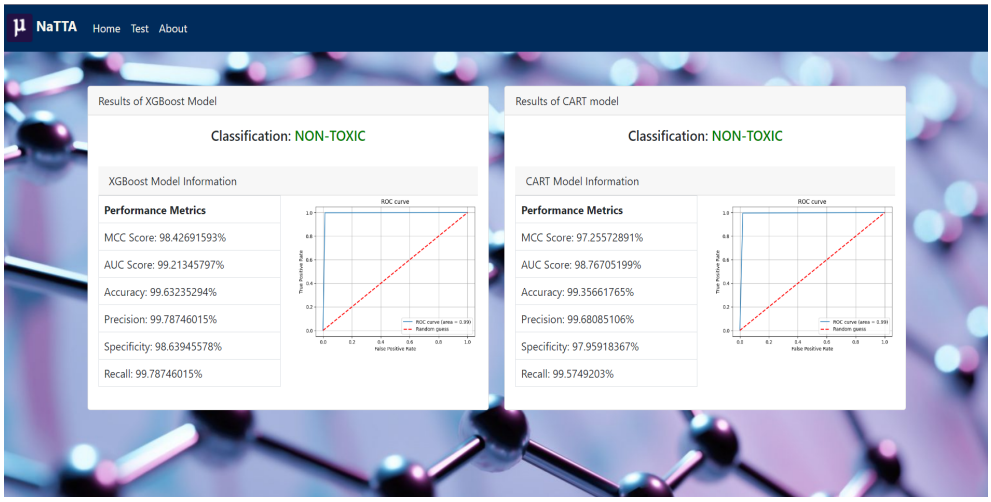


Figure 10: Result Page

may help them utilize the application in a manner suitable for them.

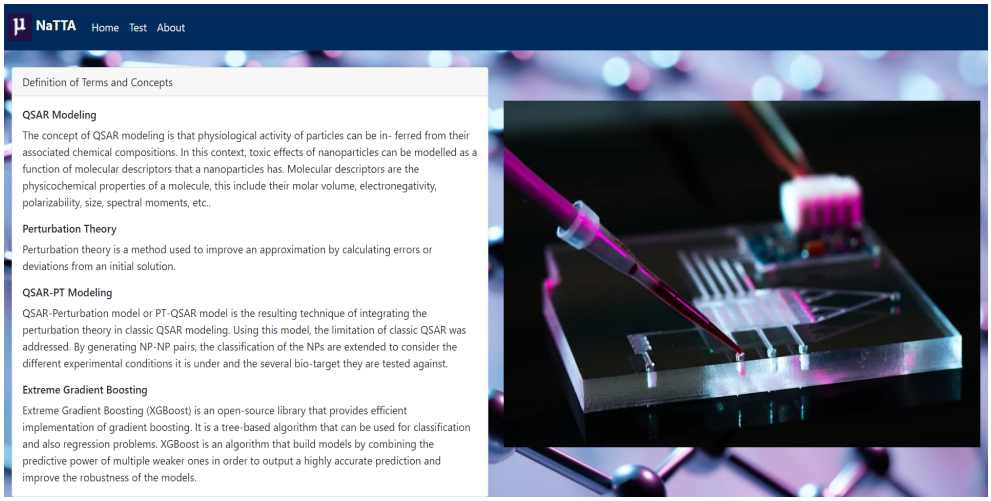


Figure 11: About Page

VI. Discussions

The aim of this study was to create a predictive model that will classify nanoparticles as either toxic or non-toxic. After the models were built, the top performing model/s were integrated to a web application. The web application developed was named NaTTA. NaTTA is a simple application that aims to predict the toxicological profile of a nanoparticle based from the perturbation values computed from the input of the user.

During the model development and its evaluation, the resulting models that used SMOTE produced better models than the ones without. Although some models achieved a higher AUC and MCC score without SMOTE, considering the other performance metrics for each model, those using SMOTE has better overall metrics. Through this comparison, the effect of SMOTE on the predictive capability of the model is observed. Models trained using imbalanced dataset can have a tendency to become bias towards the majority class. One way to work around this is to evaluate the models using AUC score. The models trained with the imbalanced dataset in this study achieved relatively high scores. However, comparing these to the AUC scores achieved by the ones that used SMOTE, there is a significant increase in the performance of each model, especially the ANN model. Additionally, this study demands attention to the percentage of classification of negative data. Comparing the specificity scores achieved by the two set of models, SMOTE boosted the performance of the models to predict negative data points correctly. Considering the increase in performances of the models, it confirms that SMOTE has an overall positive effect on the performances of the models.

For the model that performed the best in classification, both XGBoost models from the two rounds of training achieved the highest performance in their respective batch. However, comparing the two XGBoost models, the one that used SMOTE outperform the other, and thus the best model. Comparing the resulting values of its metrics

to previous studies [7][13][14], the XGBoost with SMOTE displayed a comparable performance. Furthermore, it actually exceeded the models by the previous studies. On the other hand, it is important to note that the scope of their studies are more encompassing unlike the limited and more focused scope of this one.

In the feature importance of the models used in the study, the results showed that the most important feature from each of the classifiers used was $DD\mu_2(ATO)t_a$. All four types of models got the same results. One possibility as to why it has the most impact in the value of the target variable can be attributed to its strong correlation to the target variable. From the EDA performed prior to the model development, $DD\mu_2(ATO)t_a$ exhibit a strong negative correlation with the target variable. Another thing that can be observed from the EDA analysis is that the other variables that also displayed strong correlations also placed high rankings in the feature importance. But note that their correlations with $DD\mu_2(ATO)t_a$ is much stronger than the target variable. Nonetheless, these correlations can be considered the reasons why these features indirectly became the subsequent most important features after the first one.

Comparing the resulting feature importance to the study of [7], their results converge with the findings of this study. In their paper, they determined the most significant factors affecting NP toxicity by calculating sensitivities of each feature. A high sensitivity score correlates to a high degree of impact on the target variable. $DD\mu_2(ATO)t_a$ was also the penultimate descriptor in terms of importance in their study. This descriptor accounts for the decrease in the physichochemical properties in molecular regions of two bonds or less. Additionally, this descriptor shows that toxicity of nanoparticles are time-dependent as it considers the assay times the nanoparticles were tested.

Examining the subsequent important features after $DD\mu_2(ATO)t_a$, in different orders $DD\mu_1(ATO)b_t$, $DDV(m_e)$, and $DDE(d_m)$ were the other descriptors that placed high rankings. These results are also consistent with the study of [7]. As

mentioned in the previous section of the study, nanoparticles are highly reactive due to their minute size. They can penetrate the barrier or the protective membranes of their endpoint. However, if their molar volume and atomic weight increases, their size also increases. Consequently, this decreases their chance to pass through the membranes. Therefore, descriptors that has relation to the size of the nanoparticle are expected to have high impact on its toxicity.

On the other hand, $DDE(d_m)$ is the descriptor that characterizes the difference in electronegativity. Electronegativity is the tendency to attract electrons in a molecule. In the study of [14], their results showed that increase in electronegativity causes toxicity to diminish. In line with this, the same descriptor in this study exhibits a parallel behavior and is calculated to be one of the most decisive feature of the models.

As the final output of the study, two models were used in the development of the application. The models used were the XGBoost and CART models that used SMOTE. These two models were used because they performed significantly better than the remaining other two models in terms of their MCC scores.

VII. Conclusions

Nanoparticles (NPs) are materials ranging from 1-100 nanometers. Due to their minute structure, scientists were able to engineer these materials to have specific purpose and exhibit new properties. Application of NPs found its way in different aspects of our life and has brought several benefits, especially in medicine, food, and agriculture. However, the major concern over these materials also stem from its very small structure. NPs are extremely reactive materials because of their high surface-to-volume ratio. Hence, it can exhibit toxic effects to its intended endpoint.

Nanotoxicity testing became an important phase to determine the potential risks NPs may bring. Initially, conventional experimental methods were used in testing, but these methods were expensive and time-consuming. Thus, in-silico methods became the alternative. In-silico nanotoxicity tests are usually centered around (QSAR) modeling, but in this work, Perturbation theory was integrated in an attempt to refine the predictive capability of the QSAR modeling.

The study used four different classifiers to develop the predictive models namely - SVM, XGBoost, CART, and ANN. As the first step of model building, the imbalance in the dataset was first addressed using SMOTE in order to eliminate possible bias towards the majority class. Two rounds of training were conducted in order to determine the effect of SMOTE on the models. As expected, there was a decrease in some of the metrics used due to the application of SMOTE, however it improved the AUC scores and some of the other metrics of the models. The final results showed that XGBoost with SMOTE displayed the best performance out of all the models.

For feature importance, the models were evaluated using SHAP to determine the impact of each features in the final decision of the models. The final results showed that $DD\mu_2(ATO)t_a$ was the most important feature across all four types of models. This was to be expected since in the EDA, the same feature exhibited the strongest correlation out of all descriptors to the target variable. However, it is notable that

$DD\mu_2(ATO)t_a$ is also strongly correlated to other variables and thus the succeeding important features according to SHAP were also the same descriptors.

For the output of the project, the web application was entitled NaTTA which was short for NanoToxicity Testing Application. It was developed using the Python language and Django framework. The application is easy to use and has a simple and intuitive design, which enables user to conveniently test the toxicological profile of their nanoparticle.

VIII. Recommendations

The following are some of the recommended ways to improve the findings of this study.

- Try other methods of handling the imbalanced dataset, or try a different dataset that has a more balanced distribution of classes.
- Use a dataset with a larger number of observations.
- Develop classification models using a different set of machine learning algorithms.
- Use a different tool for feature importance such as LIME, permutation importance, and Leave One Feature Out (LOFO).
- Perform a focused study on the other toxicity measures aside from $IC_{.}$

On the other hand, the web application can also be improved in the following ways:

- Incorporate an explainable AI (XAI) in order give insights to the user how the model in the application came up with the classification.
- Make the application accessible to mobile phones and other low powered devices for user's convenience.

IX. Bibliography

[1] S. Bayda, M. Adeel, T. Tuccinardi, M. Cordani, and F. Rizzolio, "The history of nanoscience and nanotechnology: From chemical–physical applications to nanomedicine," *Molecules*, vol. 25, 2020.

[2] K. Azzaoui, M. Barboucha, B. Hammouti, and R. Touzani, "Nanotechnology:history and various applications, a mini review," *EHEI Journal of Science and Technology*, vol. 02, 2022.

[3] L. Leon, E. Chung, and C. Rinaldi, *Nanoparticles for Biomedical Applications Fundamental Concepts: Biological Interactions and Clinical Applications*. Elsevier, 2020.

[4] M. Nasrollahzadeh, S. M. Sajadi, M. Sajjadi, and Z. Issaabadi, "An introduction to nanotechnology," *Interface Science and Technology*, vol. 28, 2019.

[5] M. Nasrollahzadeh, S. M. Sajadi, M. Sajjadi, and Z. Issaabadi, "Applications of nanotechnology in daily life," *Inteface Science and Technology*, vol. 28, 2019.

[6] A. V. Singh, P. Laux, A. Luch, C. Sudrik, S. Wiehr, A.-M. Wild, G. Santomauro, J. Bill, and M. Sitti, "Review of emerging concepts in nanotoxicology: opportunities and challenges for safer nanomaterial design," *Toxicology Mechanisms and Methods*, vol. 29, 2019.

[7] R. Concu, V. V. Kleandrova, A. Speck-Planche, and M. N. D. S. Cordeiro, "Probing the toxicity of nanoparticles: a unified in silico machine learning model based on perturbation theory," *Nanotoxicology*, vol. 11, 2017.

[8] A. Zeilinska, B. Costa, M. V. Ferreira, D. Miguéis, J. M. S. Louros, A. Durazzo, M. Lucarini, P. Eder, M. V. Chaud, M. Morsink, N. Willemen, P. Severino, A. Santini, and E. B. Souto, "Nanotoxicology and nanosafety: Safety-by-design

and testing at a glance," *International Journal of Environmental Research and Public Health*, vol. 17, 2020.

[9] N. R. Stillman, M. Kovacevic, I. Balaz, and S. Hauert, "In silico modelling of cancer nanomedicine, across scales and transport barriers," *npj Computational Materials*, vol. 6, 2020.

[10] H.-J. Huang, Y.-H. Lee, Y.-H. Hsu, C.-T. Liao, Y.-F. Lin, and H.-W. Chiu, "Current strategies in assessment of nanotoxicity: Alternatives to in vivo animal testing," *International Journal of Molecular Sciences*, vol. 22, 2021.

[11] I. Furxhi, F. Murphy, M. Mullins, A. Arvanitis, and C. A. Poland, "Nanotoxicology data for in silico tools: a literature review," *Nanotoxicology*, vol. 14, 2020.

[12] K. Roy, *Ecotoxicological QSARs*. Humana New York, NY, 2021.

[13] F. Luan, V. V. Kleandrova, H. González-Díaz, J. M. Ruso, A. Melo, A. Speck-Planche, and M. N. D. S. Cordeiro, "Computer-aided nanotoxicology: assessing cytotoxicity of nanoparticles under diverse experimental conditions by using a novel qstr-perturbation approach," *Nanoscale*, vol. 6, 2014.

[14] V. V. Kleandrova, F. Luan, H. Gonzalez-Díaz, J. M. Ruso, A. Speck-Planche, and M. N. D. S. Cordeiro, "Computational tool for risk assessment of nanomaterials: Novel qstr-perturbation model for simultaneous prediction of ecotoxicity and cytotoxicity of uncoated and coated nanoparticles under multiple experimental conditions," *Environmental Science and Technology*, vol. 48, 2014.

[15] A. K. Halder, A. Melo, and M. N. D. Cordeiro, "A unified in silico model based on perturbation theory for assessing the genotoxicity of metal oxide nanoparticles," *Chemosphere*, vol. 244, 2020.

[16] M. Napagoda, D. Jayathunga, and S. Witharana, *Introduction to Nanotechnology*. Singapore: Springer Nature Singapore, 2023.

[17] X. He, H. Deng, and H. min Hwang, "The current application of nanotechnology in food and agriculture," *Journal of Food and Drug Analysis*, vol. 27, 2019.

[18] A. C. Anselmo and S. Mitragotri, "Nanoparticles in the clinic: An update post covid-19 vaccines," *Bioengineering Translational Medicine*, vol. 6, 2021.

[19] A. Sarkar, D. Sarkar, and K. Poddar, *Nanotoxicity: Sources and Effects on Environment*. Cham: Springer International Publishing, 2019.

[20] C. Domingues, A. Santos, C. Alvarez-Lorenzo, A. Concheiro, I. Jarak, F. Veiga, I. Barbosa, M. Dourado, and A. Figueiras, "Where is nano today and where is it headed? a review of nanomedicine and the dilemma of nanotoxicology," *ACS Nano*, vol. 16, 2022.

[21] S. Balraadjsing, W. J.G.M.Peijnenburg, and M. G.Vijvera, "Exploring the potential of in silico machine learning tools for the prediction of acute daphnia magna nanotoxicity," *Chemosphere*, vol. 307, 2022.

[22] S. Kar, K. Pathakoti, D. Leszczynska, P. B. Tchounwou, and J. Leszczynski, "In vitro and in silico study of mixtures cytotoxicity of metal oxide nanoparticles to escherichia coli: a mechanistic approach," *Nanotoxicology*, vol. 16, 2022.

[23] M. I. Kotzabasaki, I. Sotiropoulos, , and H. Sarimveis, "Qsar modeling of the toxicity classification of superparamagnetic iron oxide nanoparticles (spions) in stem-cell monitoring applications: an integrated study from data curation to model development," *The Royal Society of Chemistry*, vol. 10, 2020.

[24] J. Cao, Y. Pan, Y. Jiang, R. Qi, B. Yuan, Z. Jia, J. Jiang, and Q. Wang, "Computer-aided nanotoxicology: Risk assessment of metal oxide nanoparticles via nano-qsar," *Green Chemistry*, vol. 22, 2020.

[25] J. Roya and K. Roy, "Risk assessment and data gap filling of toxicity of metal oxide nanoparticles (meox nps) used in nanomedicines: a mechanistic qsar approach," *Environmental Science: Nano*, vol. 9, 2022.

[26] T. Wanga, D. P. Russo, D. Bitounis, P. Demokritou, X. Jia, H. Huang, and H. Zhu, "Integrating structure annotation and machine learning approaches to develop graphene toxicity models," *Carbon*, vol. 204, 2023.

[27] B. Ortega-Tenezaca, V. Quevedo-Tumailli, H. Bediaga, J. Collados, S. Arrasate, G. M. adn Cristian R Munteanu, M. N. D. S. Cordeiro, and H. González-Díaz, "Ptml multi-label algorithms: Models, software, and applications," *Current Topic in Medicinal Chemsitry*, vol. 20, 2020.

[28] J. Korstanje, "Smote." https://towardsdatascience.com/smote-fdce2f605729#:~:text=SMOTE%20is%20a%20machine%20learning,with%20this%20type%20of%20data., 2021.

[29] S. Satpathy, "Overcoming class imbalance using smote techniques." https://www.analyticsvidhya.com/blog/2020/10/overcoming-class-imbalance-using-smote-techniques/, 2021.

[30] "Support vector machines." https://scikit-learn.org/stable/modules/svm.html.

[31] R. Gandhi, "Support vector machine — introduction to machine learning algorithms." https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca4 2018.

[32] "Decision trees." https://scikit-learn.org/stable/modules/tree.html.

[33] B. Dutta, "A classification and regression tree (cart) algorithm." https://www.analyticssteps.com/blogs/classification-and-regression-tree-cart-algorithm, 2021.

[34] S. Dobilas, "Cart: Classification and regression trees for clean but powerful models." https://towardsdatascience.com/cart-classification-and-regression-trees-for-clean-but-powerful-models, 2021.

[35] J. Brownlee, "Extreme gradient boosting (xgboost) ensemble in python." https://machinelearningmastery.com/extreme-gradient-boosting-ensemble-in-python/, 2020.

[36] S. Dobilas, "Xgboost: Extreme gradient boosting — how to improve on regular gradient boosting?." https://towardsdatascience.com/xgboost-extreme-gradient-boosting-how-to-improve-on-regular-gradient-boosting
2021.

[37] Nvidia, "Xgboost." https://www.nvidia.com/en-us/glossary/data-science/xgboost/.

[38] IBM, "What are neural networks?." https://www.ibm.com/topics/neural-networks.

[39] Techopedia, "Artificial neural network (ann)." https://www.techopedia.com/definition/5967/artificial-neural-network-ann, 2022.

X. Appendix

A. Source Code

Listing 1: Model Training (nano_models.py)

```
#Packages

import seaborn as sns #for matrix
import pandas as pd # for data manipulation
import matplotlib.pyplot as plt #for
    graph visualization
from sklearn.model_selection import
    train_test_split # will be used for data split
from sklearn.preprocessing import LabelEncoder
    # for preprocessing
from imblearn.over_sampling import SMOTE
    #for imbalance
from xgboost import XGBClassifier,
    plot_importance #for training
from sklearn.model_selection import GridSearchCV
    #for hyperparameter tuning
import joblib # for saving algorithm and
    preprocessing objects
# !pip install shap
import shap #for feature importance

from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import matthews_corrcoef
from sklearn.metrics import confusion_matrix

#load data
from google.colab import files
import io

uploaded = files.upload()
dataset = pd.read_csv(io.BytesIO(uploaded
    ['IC50.csv'])) # the name of the file
    uploaded must be the same in this line
    of code
#dataset = pd.read_csv('IC50.csv')
print(dataset)

#pre-process

dataset.info()
dataset.isna().sum()
correlation = dataset.corr()

# creating a colormap
colormap = sns.color_palette("Spectral",
    as_cmap=True)
sns.heatmap(correlation,annot=True,fmt='.2f',
    cmap=colormap,linewidths=0.1, vmax=1.0,
    vmin=-1.0, square=True)

#normalize data
for column in dataset.columns:
    dataset[column] = (dataset[column] - dataset
    [column].min()) / (dataset[column].max() -
    dataset[column].min())
print(dataset)

y = dataset['TEi(cj)_nw']
X = dataset.drop(['TEi(cj)_nw'], axis=1)

# split data into train and test sets
seed = 7
test_size = 0.20
X_train, X_test, y_train, y_test =
    train_test_split(X, y, test_size=test_size,
    random_state=seed)

#SMOTE
oversample = SMOTE(random_state=seed)
X_train, y_train = oversample.fit_resample
    (X_train, y_train)

"""#XGBOOST"""

#XGBoost
model = XGBClassifier()
model.fit(X_train, y_train)

# make predictions for test data
y_pred = model.predict(X_test)

# Evaluate predictions
accuracy = accuracy_score(y_test,y_pred)*100
precision = precision_score(y_test,y_pred)*100
f1 = f1_score(y_test,y_pred)*100
recall = recall_score(y_test,y_pred)*100
confusion_mat = confusion_matrix(y_test,y_pred)

fpr1, tpr1, thresh1 = roc_curve(y_test, y_pred,
    pos_label=1)
random_probs = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random_probs,
    pos_label=1)
auc_score = roc_auc_score(y_test, y_pred)

mcc = matthews_corrcoef(y_test, y_pred)

x_pred = model.predict(X_train)
conf = confusion_matrix(y_train,x_pred)
print(conf)

# Printing the Results
print("Accuracy:",accuracy)
print("Precision:",precision)
print("F1-score:",f1)
print("Recall/Sensitivity:",recall)
print("MCC:",mcc)
print("Confusion Matrix")
print(confusion_mat)

#Confusion Matrix
sns.heatmap(confusion_mat, annot=True, fmt='g',
    cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# AUC score and plot of ROC curve
print("AUC Score:",auc_score)
plt.plot(fpr1, tpr1, label='ROC curve (area =
    %.2f)' %auc_score)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2,
    color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()

#feature importance using shap
shap.initjs()
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test, plot_type="bar")

#save model
joblib.dump(model, 'xg_model.pkl')

"""#SVM"""

#SVM
from sklearn.svm import SVC #for training
classifier = SVC(kernel = 'rbf', random_state = seed)
classifier.fit(X_train, y_train)

# make predictions for test data
y_pred = classifier.predict(X_test)
```

```
# Evaluate predictions
accuracy = accuracy_score(y_test,y_pred)*100
precision = precision_score(y_test,y_pred)*100
f1 = f1_score(y_test,y_pred)*100
recall = recall_score(y_test,y_pred)*100
confusion_mat = confusion_matrix(y_test,y_pred)

fpr1, tpr1, thresh1 = roc_curve(y_test, y_pred,
    pos_label=1)
random_probs = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random_probs,
    pos_label=1)
auc_score = roc_auc_score(y_test, y_pred)

mcc = matthews_corrcoef(y_test, y_pred)

# Printing the Results
print("Accuracy:",accuracy)
print("Precision:",precision)
print("F1-score:",f1)
print("Recall/Sensitivity:",recall)
print("MCC:",mcc)
print("Confusion Matrix")
print(confusion_mat)

#Confusion Matrix
sns.heatmap(confusion_mat, annot=True, fmt='g',
    cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# AUC score and plot of ROC curve
print("AUC Score:",auc_score)
plt.plot(fpr1, tpr1, label='ROC curve (area =
    %.2f)' %auc_score)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2,
    color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()

#feature importance using shap
shap.initjs()
explainer = shap.KernelExplainer(
    classifier.predict, shap.sample(X_train,3))
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test,
    plot_type="bar")

#save model
joblib.dump(classifier, 'svm_model.pkl')

"""#CART"""

#CART
from sklearn.tree import DecisionTreeClassifier
    #for training
cart = DecisionTreeClassifier(criterion='gini',
    splitter='best')
cart.fit(X_train, y_train)

# make predictions for test data
y_pred = cart.predict(X_test)

# Evaluate predictions
accuracy = accuracy_score(y_test,y_pred)*100
precision = precision_score(y_test,y_pred)*100
f1 = f1_score(y_test,y_pred)*100
recall = recall_score(y_test,y_pred)*100
confusion_mat = confusion_matrix(y_test,y_pred)

fpr1, tpr1, thresh1 = roc_curve(y_test, y_pred,
    pos_label=1)
random_probs = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random_probs,
    pos_label=1)
auc_score = roc_auc_score(y_test, y_pred)

mcc = matthews_corrcoef(y_test, y_pred)

# Printing the Results
print("Accuracy:",accuracy)
print("Precision:",precision)
print("F1-score:",f1)
print("Recall/Sensitivity:",recall)
print("MCC:",mcc)
print("Confusion Matrix")
print(confusion_mat)

#Confusion Matrix
sns.heatmap(confusion_mat, annot=True, fmt='g',
    cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# AUC score and plot of ROC curve
print("AUC Score:",auc_score)
plt.plot(fpr1, tpr1, label='ROC curve (area =
    %.2f)' %auc_score)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2,
    color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()

#feature importance using shap
shap.initjs()
explainer = shap.TreeExplainer(cart, X_train)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)

# Feature importance
importance = cart.feature_importances_
for i,v in enumerate(importance):
 print('Feature: %0d, Score: %.5f' % (i,v))
plt.bar([x for x in range(len(importance))], importance)
plt.show()

#save model
joblib.dump(cart, 'cart_model.pkl')

"""#ANN-Keras"""
import keras
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

#Build the model
ann = tf.keras.Sequential([
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
    # 1 because its a binary classification
])

#Compile the model
ann.compile(
    loss = tf.keras.losses.binary_crossentropy,
    optimizer = tf.keras.optimizers.Adam(
        learning_rate = 0.02),
    metrics = [
        tf.keras.metrics.BinaryAccuracy(name='accuracy'),
        tf.keras.metrics.Precision(name='precision'),
        tf.keras.metrics.Recall(name='recall')
    ]
)

ann.fit(X_train, y_train, batch_size = 10, epochs = 100)

y_pred = ann.predict(X_test)
y_pred = (y_pred > 0.5)

# Evaluate predictions
accuracy = accuracy_score(y_test,y_pred)*100
precision = precision_score(y_test,y_pred)*100
f1 = f1_score(y_test,y_pred)*100
recall = recall_score(y_test,y_pred)*100
confusion_mat = confusion_matrix(y_test,y_pred)

fpr1, tpr1, thresh1 = roc_curve(y_test, y_pred,
    pos_label=1)
random_probs = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random_probs,
    pos_label=1)
auc_score = roc_auc_score(y_test, y_pred)

mcc = matthews_corrcoef(y_test, y_pred)

# Printing the Results
print("Accuracy for ANN is:",accuracy)
print("Precision for ANN is:",precision)
print("F1-score for ANN is:",f1)
print("Recall for ANN is:",recall)
print("MCC:",mcc)
```

```
print("Confusion Matrix")
print(confusion_mat)

#Confusion Matrix
sns.heatmap(confusion_mat, annot=True, fmt='g',
    cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# AUC score and plot of ROC curve
print("AUC Score:",auc_score)
plt.plot(fpr1, tpr1, label='ROC curve (area =
    %.2f)' %auc_score)
plt.plot([0, 1], [0, 1], linestyle='--', lw=2,
    color='r', label='Random guess')
plt.title('ROC curve')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.grid()
plt.legend()
plt.show()

#feature importance using shap
shap.initjs()
explainer = shap.KernelExplainer(ann.predict,
    shap.sample(X_train,7))
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test, plot_type="bar")

#save model
joblib.dump(ann, 'ann_model.pkl')
```

Listing 2: models.py

```
from django.db import models

# Create your models here.

class Features(models.Model):
    val1 = models.FloatField(max_length=30)
    val2 = models.FloatField(max_length=30)
    val3 = models.FloatField(max_length=30)
    val4 = models.FloatField(max_length=30)
    val5 = models.FloatField(max_length=30)
    val6 = models.FloatField(max_length=30)
    val7 = models.FloatField(max_length=30)
    val8 = models.FloatField(max_length=30)
    val9 = models.FloatField(max_length=30)
    val10 = models.FloatField(max_length=30)

NP_TYPE = (
    ('AG', 'Ag'),
    ('NIFE', 'NiFe2O4'),
    ('AL', 'Al2O3'),
    ('BI', 'Bi2O3'),
    ('CO', 'CoO'),
    ('CR', 'Cr2O3'),
    ('FE', 'Fe2O3'),
    ('IN', 'In2O3'),
    ('LA', 'La2O3'),
    ('NO', 'NiO'),
    ('SB', 'Sb2O3'),
    ('SN', 'SnO2'),
    ('TI', 'TiO2'),
    ('V', 'V2O3'),
    ('W', 'WO3'),
    ('Y', 'Y2O3'),
    ('ZN', 'ZnO'),
    ('ZR', 'ZrO2'),
    ('PT', 'Pt')
)

TOX_MEASURE = (
    ('CC', 'CC50'),
    ('LC', 'LC50'),
    ('EC', 'EC50')
)

ENDPOINT = (
    ('RAW', 'RAW 264.7 (M)'),
    ('A', 'A549 (H)'),
    ('HACAT', 'HaCaT (H)'),
    ('DR', 'Danio rerio (embryos)'),
    ('PS', 'Pseudokirchneriella subcapitata')
)

SHAPE = (
    ('SPHERE', 'spherical'),
    ('IRREG', 'irregular'),
    ('POLY', 'polyhedral'),
    ('N/A', 'n/a')
)

CONDITION = (
    ('WATER', 'H2O'),
    ('DRY', 'Dry'),
    ('DULBECCO', 'DMEM')
)

ASSAY_TIME = (
    ('FOUR', '4'),
    ('TWOFOUR', '24'),
    ('FOUREIGHT', '48'),
    ('SEVENTWO', '72'),
    ('NINESIX', '96'),
    ('ONETWENTY', '120'),
    ('ONESIXTYEIGHT', '168')
)

COAT = (
    ('PDAD', 'PDADMAC'),
    ('U', 'UC'),
    ('OLE', 'oleate')
)

class Reference(models.Model):
    np = models.CharField(choices=NP_TYPE, max_length=15,
        null=False)
    me = models.CharField(choices=TOX_MEASURE, max_length=15,
        null=False)
    bt = models.CharField(choices=ENDPOINT, max_length=15,
        null=False)
    ns = models.CharField(choices=SHAPE, max_length=15,
        null=False)
    dm = models.CharField(choices=CONDITION, max_length=15,
        null=False)
    ta = models.CharField(choices=ASSAY_TIME, max_length=15,
        null=False)
    sc = models.CharField(choices=COAT, max_length=15,
        null=False)
    f1 = models.FloatField(max_length=100, null=False)
    f2 = models.FloatField(max_length=100, null=False)
    f3 = models.FloatField(max_length=100, null=False)
    f4 = models.FloatField(max_length=100, null=False)
    f5 = models.FloatField(max_length=100, null=False)
    f6 = models.FloatField(max_length=100, null=False)
    f7 = models.FloatField(max_length=100, null=False)
    f8 = models.FloatField(max_length=100, null=False)
    f9 = models.FloatField(max_length=100, null=False)
    f10 = models.FloatField(max_length=100, null=False)
```

Listing 3: forms.py

```
from django import forms
from .models import *
from django.db import models
from django.db.models import fields
from django.forms import NumberInput
from django.forms import widgets, ModelForm

class FeatureForm(forms.ModelForm):
    class Meta:
        model = Features
        fields = "__all__"
        widgets = {
            'val1': NumberInput(attrs={'class':
                'form-control'}),
            'val2': NumberInput(attrs={'class':
                'form-control'}),
            'val3': NumberInput(attrs={'class':
                'form-control'}),
            'val4': NumberInput(attrs={'class':
                'form-control'}),
            'val5': NumberInput(attrs={'class':
                'form-control'}),
```

```
            'val6': NumberInput(attrs={'class':
                'form-control'}),
            'val7': NumberInput(attrs={'class':
                'form-control'}),
            'val8': NumberInput(attrs={'class':
                'form-control'}),
            'val9': NumberInput(attrs={'class':
                'form-control'}),
            'val10': NumberInput(attrs={'class':
                'form-control'}),
        }
```

Listing 4: views.py

```
from django.shortcuts import render
from .models import *
from .forms import *
import joblib
import numpy as np
from tensorflow.keras.models import load_model


# Create your views here.
def home(request):
    return render(request,"nanotest/home.html")

def test(request):
    form = FeatureForm

    data = {"form":form}
    return render(request,"nanotest/test.html", data)

def about(request):
    return render(request,"nanotest/about.html")

def predict(request):
#    cls=joblib.load('xg_model.pkl')
   cls= load_model('ann_keras_model.h5')
   lis=[]

   val01 = request.POST.get('val1')
   val1 = float(val01)
   val02 = request.POST.get('val2')
   val2 = float(val02)
   val03 = request.POST.get('val3')
   val3 = float(val03)
   val04 = request.POST.get('val4')
   val4 = float(val04)
   val05 = request.POST.get('val5')
   val5 = float(val05)
   val06 = request.POST.get('val6')
   val6 = float(val06)
   val07 = request.POST.get('val7')
   val7 = float(val07)
   val08 = request.POST.get('val8')
   val8 = float(val08)
   val09 = request.POST.get('val9')
   val9 = float(val09)
   val010 = request.POST.get('val10')
   val10 = float(val010)

   #normalize input
   val1 = (val1 - (-9.599)) / (10.666 - (-9.599))
   val2 = (val2 - (-103.384)) / (71.316 - (-103.384))
   val3 = (val3 - (-421.105)) / (394.078 - (-421.105))
   val4 = (val4 - (-301.826)) / (67.751 - (-301.826))
   val5 = (val5 - (-1.345)) / (1.197 - (-1.345))
   val6 = (val6 - (-114.686)) / (87.55 - (-114.686))
   val7 = (val7 - (-57605.281)) / (38894.4 - (-57605.281))
   val8 = (val8 - (-1127.534)) / (35.027 - (-1127.534))
   val9 = (val9 - (-355915832.7)) / (28233130.49 -
    (-355915832.7))
   val10 = (val10 - (-1)) / (1 - (-1))

   lis.append(val1)
   lis.append(val2)
   lis.append(val3)
   lis.append(val4)
   lis.append(val5)
   lis.append(val6)
   lis.append(val7)
   lis.append(val8)
   lis.append(val9)
   lis.append(val10)
   print(lis)

   data_array = np.asarray(lis)
   arr= data_array.reshape(1,10)
   print(arr)

   ans = cls.predict(arr)
   print(ans)
   ans = (ans > 0.5)
   print(ans)

   finalans=''
   if(ans==1):
      finalans='Your nanoparticle is NON-TOXIC'
   elif(ans==0):
      finalans = 'Your nanoparticle is TOXIC'
   print(finalans)
   return render(request, "nanotest/result.html",{'ans':
    finalans,"val1":val01, "val2":val02,"val3":val03, "val4":
        val04,"val5":val05,"val6":val06,"val7":val07, "val8":
        val08,"val9":val09, "val10":val010,})
```

XI. Acknowledgment

After many years of learning, finally my journey as a student is coming to an end. For almost two decades, the school had been a second home to me, it is where I learned a lot of life lessons, gained extraordinary experiences, and unforgettable memories. As I close this chapter of my life, first I would like to express my gratitude to everyone who supported me and gave me the strength to reach this important milestone.

To my parents, Dad and Mama, thank you for everything. Finally we have made it! To my sister, who have my back whenever I am in a pinch, and to my older brothers who gave me advices when I needed it, thank you very much.

Next, to my friends who helped me survive being a student. Thank you for accepting me. To my best friend, Nicole who kept me in line during high school, and to Gwen and Lady who are my lifesavers in college. You guys played a big part in my life and has a huge space in my heart.

To all my professors during my stay in UPM, thank you for imparting your knowledge to us. I am forever grateful for all the lessons and the considerations I am given. To Ma'am Perl, thank you for being a great mentor during the formulation of this SP. Moreover, thank you for being a kind and approachable adviser, as well as for the moral support all throughout this project . I also extend my gratitude to Sir John and Jahaziel, who were my collaborators for this SP. Through this collaboration, I gained new experiences and was able to meet a new friend.

Lastly, everything that I was able to achieve was because of the help of Almighty God. Lord thank you for giving me the strength that I need to surpass all the hardships and obstacles that hindered my way. I also thank you for letting me meet the right people who completed this chapter of my life. Thank you for answering my prayers, as well as for being the mightiest rock that kept me safe all throughout my journey.