Nil Munté Guerrero

STRUCTURAL NANOFINGERPRINT GENERATION

MASTER THESIS

Supervised by Dr. Francesc Serratosa i Casanelles

University Master's Degree in Computer Security Engineering and Artificial Intelligence

UNIVERSITAT ROVIRA I VIRGILI

Tarragona

2023

ACKNOWLEDGMENTS

The completion of this Master Thesis would have not been possible without the help and collaboration with the group of people which I will comment below.

Firstly, I would like to thank Francesc Serratosa, the supervisor of this Master Thesis. He trusted me in the realization of this project, he has been a reference and he has guided me throughout the execution of this work. Thanks to his advice I have been able to complete it and, above all, to introduce myself deeper into the research world.

Secondly, I would like to thank to the DEIM, ETSE and URV, as well as to all the professors of the MESIIA, for their proximity, help and knowledge learnt during the course and development of this Master Thesis.

Finally, I would also like to thank Núria and my family for giving me encouragement and moral support at all times.

ABSTRACT

(English) This Master Thesis is focused on the study and generation of *Structural NanoFingerprints*, representations of metal-oxide nanoparticles based on the local information of their three-dimensional structure. Firstly, it is described their structure. Then, the algorithm implemented in C-language to correctly generate them is described, and tested using a correctness checking algorithm, as well as the study of the computational time of the code and its parts to determine which are the most consuming sections. Finally, the artificial generation of *Structural NanoFingerprints* is described using Linear and Non-Linear (Multiple) Regression with MATLAB, in which different models are presented.

(Catalan) Aquesta Tesi de Màster es centra en l'estudi i generació d'*Structural NanoFingerprints*, representacions de nanopartícules d'òxid metàl·lic basades en la informació local de la seva estructura tridimensional. En primer lloc, es descriu la seva estructura. A continuació, es descriu l'algorisme implementat en llenguatge C per generar-les correctament, i és testejat mitjançant un algorisme de comprovació de la seva correcció, així com l'estudi del temps computacional del codi i les seves parts per determinar quines són les seccions que més consumeixen. Finalment, es descriu la generació artificial d'*Structural NanoFingerprints*

mitjançant la Regressió (Múltiple) Lineal i No Lineal amb MATLAB, amb la que es presenten diversos models.

(Spanish) Esta Tesis de Máster se centra en el estudio y generación de Structural NanoFingerprints, representaciones de nanopartículas de óxido metálico basadas en la información local de su estructura tridimensional. En primer lugar, se describe su estructura. A continuación, se describe el algoritmo implementado en lenguaje C para generarlas correctamente, y se testea mediante un algoritmo de comprobación de su corrección, así como el estudio del tiempo computacional del código y sus partes para determinar cuáles son las secciones que más consumen. Finalmente, se describe la generación artificial d'Structural NanoFingerprints mediante la Regresión (Múltiple) Lineal y No Lineal con MATLAB, en la que se presentan diferentes modelos.

KEYWORDS

Nanoparticle. Particles with dimensions in the nanometre (1 nm = 10^{-9} m) range, with size of maximum 100 nm (1000 Å). The nanoparticles referred on this Master Thesis are metal-oxide nanoparticles, composed of Oxygen and metal atoms. During the Master Thesis, nanoparticles are also referred as nanocompounds.

Structural NanoFingerprint. Representations of nanocompounds based on the local information of the three-dimensional structure, which allows the characterisation of the local structure through the atoms' connectivity, maintaining the information of the three-dimensional structure but increasing the management ability.

XYZ file format. Datafile which specifies molecular geometries using a Cartesian coordinate system. They are used in this Master Thesis to describe the 3D-structure of a metal-oxide nanoparticle.

Subgraph search. Algorithm designed to count the number of appearances of a subgraph into a graph without repetitions.

Local Structure. Small set of close atoms and their bonds in the nanocompound.

Regression model. Statistical tool used to examine the relationship between a dependent variable (output) and one or more independent variables (predictors). It estimates this relationship by fitting a mathematical equation to the data and providing a set of coefficients used for prediction.

INDEX

CHAPTER I
INTRODUCTION

In the introduction section of this Master Thesis, it will be presented its context, aim, and the structure of the document.

1.1. Context of the Master Thesis

The Master Thesis has been done in collaboration with Dr. Francsesc Serratosa i Casanelles, principal researcher of the Research Group *ASCLEPIUS: Smart Technology for Smart Healthcare*, at *Universitat Rovira i Virgili* (URV).

Structural NanoFingerprints are representations of nanocompounds based on the local information of the three-dimensional structure. They were presented in *NanoCommons* congress, 2022, in Cyprus [1]. Structural NanoFingerprints are represented in form of a vector.

A *Structural NanoFingerprint* is a description of the local relations between atoms of the nanoparticle (i.e., their coordination within successive layers of neighbours), which allows the characterisation of the local structure through the atoms' connectivity, maintaining the information of the three-dimensional structure but increasing the management ability.

Structural NanoFingerprints can be defined as a fingerprint for metal oxide nanoparticles based on three steps: first, extracting the shell of the nanoparticle. Second, representing the shell as an attributed graph. Finally, extracting the appearance of some subgraphs. Therefore, *Structural NanoFingerprints* can be also named as *Subgraph NanoFingerprints*.

1.2. Aim of the Master Thesis

The aim of this Master Thesis is threefold.

One the one hand, the first objective is to study the *NanoFingerprints* structure and code in C language an algorithm to generate this structure given a three-dimensional structure of the nanocompound.

On the other hand, the second objective is to develop a Multiple Regression model to generate correct *NanoFingerprints*. Since the generation of these structures through a three-dimensional information (the first aim of this master thesis) is computationally very expensive, the generation of *NanoFingerprints* through a regression will be much less computationally

expensive. Different regression models will be presented (Linear, Non-Linear), depending on some cases that will be introduced.

Finally, the third objective is to evaluate and compare both methods and show the validity and usefulness of the regression models to generate *NanoFingerprints*.

1.3. Structure of the document

This Master Thesis is divided into seven chapters. The structure in which it will be developed is the following one.

Chapter I. Introduction. It is defined the context, aim, and structure of the document.

Chapter II. Structural NanoFingerprints. It is deeply explained the *Structural NanoFingerprint* structure, including its input parameters, sections, and output format. At the end of the chapter some examples of the *Structural NanoFingerprint* generation of TiO_2 with different sizes are presented.

Chapter III. The algorithm to generate NanoFingerprints. The algorithm to correctly generate *Structural NanoFingerprints* is presented and deeply explained, as well as a previous approach that was misleading.

Chapter IV. NanoFingerprint generation analysis. An introduction to the used metal-oxide nanoparticles and an analysis of the algorithm is done by presenting the results of some metal-oxide nanoparticles *NanoFingerprints* generation, a correctness checking algorithm and the study of the computational time spent by different sizes of metal-oxide nanoparticles.

Chapter V. Artificial generation of NanoFingerprints. This Chapter is focused on the artificial generation of *Structural NanoFingerprints*. On the first place, it is presented the motivation of artificially generating them and why a regression model is suitable. Then, the model is presented, including the explanation of the pre-processing and postprocessing phases, as well as the techniques and error metrics applied. Next, the four models generated are presented and studied, with results, and finally, some conclusions are taken, comparing the methods on this Chapter.

Chapter VI. Conclusion and future research lines. The aim of this last Chapter is to summarize the work and contribution of this Master Thesis, extract conclusions and propose improvements for future research lines.

Annex. The link to the GitHub and explanation of the codes and generated *NanoFingerprints* used during the Master Thesis are provided.

1.4. References

[1] F. Serratosa, S. Álvarez, L. Escorihuela and M. Calatayud. Subgraph NanoFingerprint for modelling metal oxide nanoparticles based on connected atoms exploration. NanoWeek & NanoCommons. Final Conference 2022, Cyprus 2022.

CHAPTER II
STRUCTURAL NANOFINGERPRINTS

On this Chapter it will be deeply explained the *Structural NanoFingerprint* structure, including its input parameters, sections, and output format. At the end of the chapter some examples of the *Structural NanoFingerprint* generation of TiO_2 with different sizes will be presented.

2.1. State of the art

Binary metal oxide nanoparticles are chemical compounds composed of two types of atoms, Oxygen, and any metal, for instance, O, Al, Cu, Fe, Ti or Zn. The most common metal oxide nanoparticles are Al_2O_3, CuO, Fe_2O_3, TiO_2, and ZnO. The analysis and prediction of some of their properties, such as toxicity, solubility, reactivity, or electronic structure, is crucial for better understanding these compounds and their use in the industry [1].

Nanoparticles can be represented as attributed graphs. An attributed graph is a mathematical model of an object composed of two types of representations: nodes and edges. Nodes are individual components in the object, and edges are relations between these components. Attributed graphs have been applied in machine learning for a long time in applications ranging from character recognition to social network exploring, among others, and more precisely, in Ligand-Based Virtual Screening [2][3].

The management of the whole structure is tedious in computational chemistry due to the management of large amount of data, as the number of atoms increases exponentially with the size. This is the reason why some methods discard the core of the nanocompound and only keep the three-dimensional structure of the shell of the nanocompound. It has been demonstrated that the reactivity or toxicity of the nanoparticle mostly depends on the shell of the nanoparticle. This is the reason why some methods have been presented to deduce the thickness of the shell of the nanoparticle with the aim of discerning between core and shell and putting more effort on understanding the properties of the shell [4].

In our case, atoms in the nanoparticle are represented by nodes and chemical bounds by edges. Moreover, nodes and edges can be attached with some attributes, which are the main information of these local parts of the objects and their relations. The node attributes are the type of atom (O, Al, Cu, Fe, Ti or Zn), and the edge attribute, despite of not being used, might be the type of bond between atoms or some relations between the nodes it connects.

The representation of all-atom three-dimensional structure of the nanocompound would be ideal, as it could account explicitly for structural effects. However, as it has been said, the use of the whole structure is tedious due to the huge data management and the structural complexity that accompanies the surface of the nanoparticle. For this reason, modelling size-realistic nanomaterials to analyse some of these properties is a current challenge in computational and theoretical chemistry.

Developing appropriate tools that allow the quantitative analysis of the structure, as well as the selection of regions of interest such as core-shell, is a crucial step to enable efficient analysis and processing of model nanostructures.

On the following sub-sections, it is presented the XYZ file format, the most-used format to represent metal oxide nanoparticles based on its 3D-structure. From it, and additional parameters, *Structural NanoFinerprints* are presented, a computationally less expensive and vector-based form to represent metal oxide nanoparticles.

2.2. The XYZ file format

XYZ datafiles specify molecular geometries using a Cartesian coordinate system. This simple, stripped-down, ASCII-readable format was intended to serve as a transition format for the XMol series of applications.

The XYZ format supports multi-step datasets [5][6]. Each step is represented by a two-line header, followed by one line for each atom.

- The first line of a step's header is the *number of atoms* in that step. This integer may be preceded by whitespace; anything on the line after the integer is ignored.

- The second line of the header leaves room for a *descriptive string*. This line may be blank, or it may contain some information pertinent to that particular step, but it must exist, and it must be just one line long.

- After the first two lines, the rest of lines describe, on each one, a single atom. Each line of text must contain at least four fields of information, separated by whitespace: the *atom's type* (a short string of alphanumeric characters, for instance, O, Al, Cu, Fe, Ti or Zn), the *x-position*, the *y-position* and the *z-position*.

Optionally, extra fields on each line after the first two ones may be used to specify a *charge for the atom*, and/or a *vector associated with the atom*. If an input line contains five or eight

fields, the fifth field is interpreted as the atom's charge; otherwise, a charge of zero is assumed. If an input line contains seven or eight fields, the last three fields are interpreted as the components of a vector. These components should be specified in angstroms[1].

Note that the XYZ format does not contain connectivity information. This intentional omission allows for greater flexibility: to create an XYZ file, it is not needed to know where a molecule's bonds are, just where its atoms are. Connectivity information is generated automatically for XYZ files as they are read into XMol-related applications. Briefly, if the distance between two atoms is less than the sum of their covalent radius, they are considered bonded.

Figure 2.1 contains an anatase nanoparticle TiO_2 of size 0.4 nm (4 Å) in specified in XYZ format. During the development of this Master Thesis some nanoparticles will be presented with name, for instance, TiO2_002. Firstly, it is found the metal atom (on this case, Titanium) and then, the diameter in angstroms (the diameter is half the size). Note that size is rounded, since the real size of the nanoparticle is 3.84 Å.

```
7
02 Ti
 0   -1.8650   -0.0000   -0.4685
 0    0.0000   -1.8650    0.4684
 0    0.0000    0.0000   -1.8740
Ti    0.0000    0.0000    0.0000
 0    0.0000    0.0000    1.8740
 0    0.0000    1.8650    0.4684
 0    1.8650    0.0000   -0.4685
```

Figure 2.1. TiO_2 of size 0.4 nm specified in XYZ format.

There are several sources to read and represent molecular structures in XYZ format. For instance, *Jmol*[2], an open-source viewer of molecular structures. In addition, there are programming platforms, such as MATLAB, which incorporate functions to read XYZ files (for instance, the *molviewer* function).

Metal-Oxide nanoparticles can be created by using *NanoCrystal*[3], a Web-Based Crystallographic Tool for the Construction of Nanoparticles Based on Their Crystal Habit. The generation of the three-dimensional structure of a nanocompound is usually carried out in two steps. In the first step, a crystalline structure is generated by replication of an initial core. And in a second step, this structure is slightly modified by an optimisation process based on moving

[1] An angstrom is equal to 0.1 nm.
[2] https://jmol.sourceforge.net
[3] https://nanocrystal.vi-seem.eu/CrystalTool/index.php

the atoms such that the forces on each atom, and the total energy are minimized. Clearly, the second step is computationally much more demanding than the first one.

The explanation on how the nanoparticles of different sizes were obtained is out of this Master Thesis scope, but the on the Reference Guide [7] it is explained its usage.

2.3. NanoFingerprint generation

2.3.1. Input parameters

Three parameters are needed to generate *NanoFingerprints* (Figure 2.2):

- *Shell thickness*. A positive real number that defines the external radius of the *NanoFingerprint* such that the atoms in the defined volume are considered to be inside the shell and thus, the atoms that influence on the generation of the *NanoFingerprint*. It is measured in angstroms (Å).

- *Maximum bonds* (*MAX*). A natural number that defines the maximum number of bonds per atom that are considered to generate the *NanoFingerprint*. This parameter is needed to generate a fixed representation of the *Structural NanoFingerprint*, as described in 2.3.3. Structural NanoFingerprint definition. A larger number makes a larger NanoFingerprint and a larger chance of having more null values in it.

- *3D structure*. A file in XYZ format that contains the 3D structural information of the *NanoFingerprint*. Note that it could be considered to introduce an "emptied" 3D structure, this is, an XYZ file that only contains the atoms in the shell. In this case, the XYZ is smaller and the generation of the *NanoFingerprint* is faster.

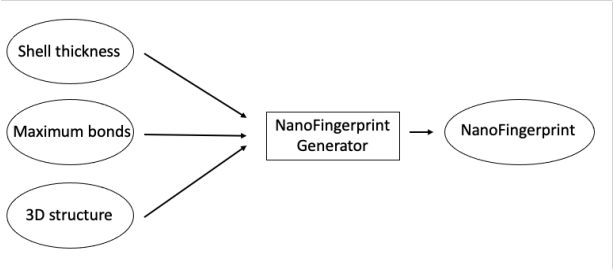


Figure 2.2. Main scheme to generate a *NanoFingerprint*.

The thickness input parameter has to be thoroughly chosen. As it has been said on 2.1. State of the art, some properties of the nanoparticle (such as reactivity or toxicity) mostly depend on the shell of the nanoparticle. Atoms and bonds in the core of the nanocompound have little or any influence on its reactivity or toxicity. To correctly obtain the shell, the thickness is key.

A tool to calculate the shell-depth is the webserver *NanoGen shell-depth calculator*[4] [8][9]. From an input nanoparticle in XYZ format, it first calculates the coordination number for each atom[5] and then uses these values with *Kneedle algorithm* [10] to find the optimal shell-depth.

2.3.2. Basic definitions

A *local structure* is a small set of close atoms and their bonds in the nanocompound. On Table 2.1 are presented some local structures that are used to define, in an organised manner, the *Structural NanoFingerprint*.

Local structure	Description
$O(x)$	A local structure composed of an Oxygen atom that has a covalent bond to other x Oxygen or metal atoms, independently.
$M(x)$	A local structure composed of a metal atom that have a covalent bond to other x Oxygen or metal atoms, independently.
$O(x,y)$	A local structure composed of a central Oxygen atom connected to x Oxygen and y metal atoms.
$M(x,y)$	A local structure composed of a central metal atom connected to x Oxygen and y metal atoms.
$O(x,y)-O(x',y')$	A local structure composed of an $O(x,y)$ and an $O(x',y')$ whose central Oxygen atoms are connected by a bond.
$M(x,y)-M(x',y')$	A local structure composed of a $M(x,y)$ and a M(x',y') whose central metal atoms are connected by a bond.
$O(x,y)-M(x',y')$	A local structure composed of an $O(x,y)$ and a M(x',y') whose central atoms are connected by a bond.

Table 2.1. Local Structure definitions.

[4] https://nanogen.me/shell-depth
[5] The coordination number of an atom refers to the total number of atoms, ions, or molecules bonded to the atom in question.

Note that the x and y atoms could be connected to other atoms.

2.3.3. Structural NanoFingerprint definition

A *Structural NanoFingerprint* is a vector of numbers that counts the number of appearances of local structures in the shell of the nanocompound. All values are positive integer numbers (zero included) except for the third element that is a positive real number, as it will be explained below.

A *Structural NanoFingerprint* is split up in four main sections: *Global information* (section 1), *atomic information* (section 2), *Linking information* (section 3) and *Structural information* (section 4). Note that the number of atoms and bonds involved in each local structure increases in each section, that is, larger local structures are considered.

Table 2.2 details the length of each section, three of them depending on the input parameter *Maximum Bonds* (*MAX*).

Section	Length
Section 1	6
Section 2	$2 \cdot MAX$
Section 3	$2 \cdot (MAX + 1)^2$
Section 4	$3 \cdot (MAX + 1)^4$

Table 2.2. Length of each *Structural NanoFingerprint* section.

Therefore, the total length of a *Structural NanoFingerprint* is:

$$length_{NanoFingerprint} = 6 + 2 \cdot MAX + 2 \cdot (MAX + 1)^2 + 3 \cdot (MAX + 1)^4$$

The reasons of each section length will be detailed on each section.

a) Section 1. Global information

The first section accounts for global information of the structure and the two parameters used to generate the *Structural NanoFingerprint*. It is composed of 6 values; therefore, its length is 6.

1. Shell thickness (Å).

2. Maximum number of bonds per atom (*MAX*).

3. Size (Å).

4. Atomic number of the metal.

5. Number of Oxygen atoms in the shell.

6. Number of metal atoms in the shell.

The shell thickness and maximum number of bonds are also the first and second algorithm input parameters, respectively, as detailed in 2.3.1. Input parameters.

Only the most external atoms that are in the volume defined between the maximum radius and the maximum radius minus the shell thickness are considered. In the case that all atoms of the compound are wanted to be considered, the shell thickness has to be set as a value larger or equal than half of the size of the nanocompound. The size is the third value in the *Structural NanoFingerprint* and it is the maximum distance between two external atoms of the nanocompound. If the nanocompound is spherical, then it represents the diameter.

Atoms with larger number of bonds than *MAX* are not considered. It is supposed they do not exist; thus, it has to be imposed a value such that any (or few) atoms are discarded.

Finally, regarding to the atomic number of the metal, its apparition reason is because *Structural NanoFingerprints* are thought to embed the structure of metal-oxide nanocompounds with only one type of metal.

b) Section 2. Atomic information

This section includes the information of the local structures $O(x)$ and $M(x)$. It is composed of $2 \cdot MAX$ values. The first MAX ones are referred to the Oxygen atoms $O(x)$, with x ranging from $x = 1$ to $x = MAX$, and the other ones to the metal atoms $M(x)$, with x ranging from $x = 1$ to $x = MAX$.

This section is described as follows:

1. Number of $O(1)$.

...

MAX. Number of $O(MAX)$.

$MAX + 1$. Number of $M(1)$.

...

$2 \cdot MAX$. Number of $M(MAX)$.

c) Section 3. Linking information

This section includes the information of the local structures $O(x, y)$ and $M(x, y)$. It is composed of $2 \cdot (MAX + 1)^2$ values.

The first $(MAX + 1)^2$ values are referred to the $O(x, y)$ local structures, with x and y ranging from $x = 0$ and $y = 0$ to $x = MAX$ and y= MAX. The second $(MAX + 1)^2$ values are referred to the $M(x, y)$ local structures, with the same range of x and y.

This section is described as follows:

1. Number of $O(0,0)$.

...

$MAX + 1$. Number of $O(0, MAX)$.

$MAX + 2$. Number of $O(1,0)$.

...

$(MAX + 1)^2$. Number of $O(MAX, MAX)$.

$(MAX + 1)^2 + 1$. Number of $M(0,0)$.

...

$(MAX + 1)^2 + MAX + 2$. Number of $M(0, MAX)$.

$(MAX + 1)^2 + MAX + 3$. Number of $M(1,0)$.

...

$2 \cdot (MAX + 1)^2$. Number of $M(MAX, MAX)$.

d) Section 4. Structural information

This section includes the information of the local structures $O(x,y) - O(x',y')$, $M(x,y) - M(x',y')$ and $O(x,y) - M(x',y')$. It is composed of $3 \cdot (MAX+1)^4$ values.

The first $(MAX+1)^4$ values are referred to the $O(x,y) - O(x',y')$ local structures, with x, x', y and y' ranging from $x = 0, x' = 0, y = 0$ and $y' = 0$ to $x = MAX, x' = MAX, y = MAX$ and $y' = MAX$. The second $(MAX+1)^4$ values are referred to the $M(x,y) - M(x',y')$ local structures, with the same range of x, x', y and y'. The third $(MAX+1)^4$ values are referred to the $O(x,y) - M(x',y')$ local structures, with the same range of x, x', y and y' as well.

This last section is described as follows:

1. Number of $O(0,0) - O(0,0)$.

...

$MAX+1$. Number of $O(0,0) - O(0,MAX)$.

$MAX+2$. Number of $O(0,0) - O(1,0)$.

...

$(MAX+1)^2$. Number of $O(0,0) - O(MAX,MAX)$.

$(MAX+1)^2 + 1$. Number of $O(0,1) - O(0,0)$.

...

$(MAX+1)^3$. Number of $O(0,MAX) - O(MAX,MAX)$.

$(MAX+1)^3 + 1$. Number of $O(1,0) - O(0,0)$.

...

$(MAX+1)^4$. Number of $O(MAX,MAX) - O(MAX,MAX)$.

$(MAX+1)^4 + 1$. Number of $M(0,0) - M(0,0)$.

...

$2 \cdot (MAX + 1)^4$. Number of $M(MAX, MAX) - M(MAX, MAX)$.

$2 \cdot (MAX + 1)^4 + 1$. Number of $O(0,0) - M(0,0)$.

...

$3 \cdot (MAX + 1)^4$. Number of $O(MAX, MAX) - M(MAX, MAX)$.

For the local structures $M(x, y) - M(x', y')$ and $O(x, y) - M(x', y')$, the index counting order was not as detailed as for $O(x, y) - O(x', y')$, but they follow the same dynamic.

2.3.4. Structural NanoFingerprint example

On this section some examples of correct *Structural NanoFingerprint* generations will be presented. On Figure 2.3 it is shown the 3D structure of the nanoparticle TiO_2 with size 0.6 nm (TiO2_003). The image have been generated by the MATLAB function *molviewer*. The rest of nanoparticles 3D representations on this work have been represented using this function.

The Oxygen atoms are represented as red spheres and the Titanium atoms in green. As it can be seen, there are two local structures marked. On green, there is the local structure $M(2,1)$. The central atom (node with id 0) is a Titanium, circled in green as well. It has two bonds with two Oxygen atoms (bonds formed by the node ids 0-4 and 0-1) and one bond with a Titanium atom (bond formed by the node ids 0-5). On yellow, there is the local structure $O(0,3)$. Its central atom (node with id 6) is an Oxygen, circled in yellow. It has three bonds with three Titanium atoms (bonds formed by the node ids 6-8, 6-5 and 6-3) and any bond with Oxygen atoms.

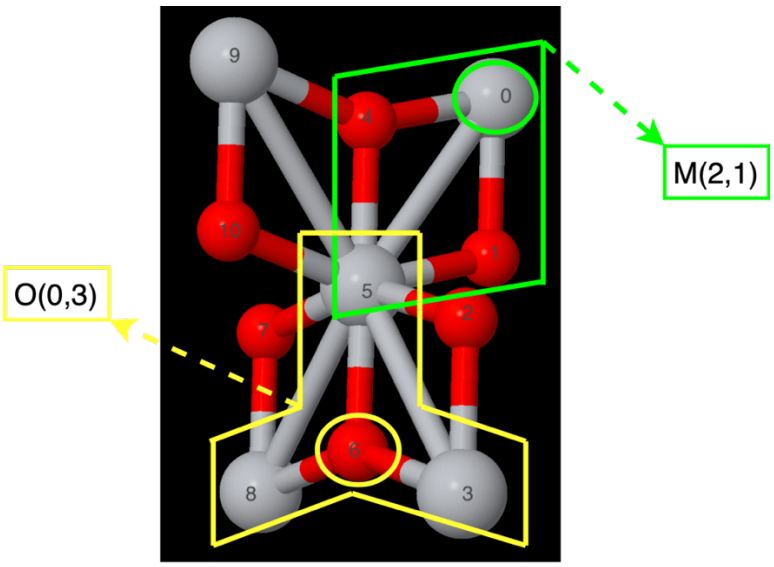


Figure 2.3. 3D structure of TiO_2 of size 0.6 nm (6 Å).

On Table 2.3 it is presented the extracted *NanoFingerprint* from the same previous three-dimensional structure, TiO_2 of size 0.6 nm, and it is compared to its 3D nanoparticle structure and XYZ format. It is a verbose Structural *NanoFingerprint*, indicating what each line is and just with the non-zero lines printed. The output formats of the *NanoFingerprints* generated are detailed on 3.3. Final approach. In addition, note that from section 2 on it is shown the index (position) of the local structure.

The coordination regarding to the number of first neighbours bonded to a different atom than itself is defined as $O(0,y)$ or $M(x,0)$: there are 4 two-fold coordinated O sites (marked as $O(0,2)$) and 2 three-fold coordinated O sites (marked as $O(0,3)$). Since there is no coordination for Ti, the local environment is not captured by the combination of pairs of O and Ti sites.

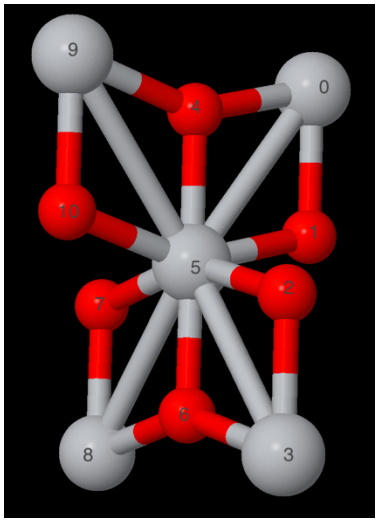
3D Structure	NanoFingerprint	XYZ format
	Shell: 100 MaxBonds: 10 Size: 5.988513 Atomic: 22 O: 6 M: 5 8-> O[2]: 4 9-> O[3]: 2 19-> M[3]: 4 26-> M[10]: 1 29-> O[0,2]: 4 30-> O[0,3]: 2 171-> M[2,1]: 4 218-> M[6,4]: 1 17764-> M[2,1]_M[6,4]: 4 29817-> O[0,2]_M[2,1]: 4 29864-> O[0,2]_M[6,4]: 4 29938-> O[0,3]_M[2,1]: 4 29985-> O[0,3]_M[6,4]: 2	11 02 Ti Ti -1.8650 -0.0000 -2.3425 O -1.8650 0.0000 -0.4685 O 0.0000 -1.8650 0.4684 Ti 0.0000 -1.8650 2.3425 O 0.0000 0.0000 -1.8740 Ti 0.0000 0.0000 0.0000 O 0.0000 0.0000 1.8740 O 0.0000 1.8650 0.4684 Ti 0.0000 1.8650 2.3425 Ti 1.8650 0.0000 -2.3424 O 1.8650 0.0000 -0.4685

Table 2.3. 3D structure of the nanoparticle of TiO_2 with size 0.6 nm (left), its Structural NanoFingerprint generation (center) and the nanoparticle described in XYZ format (right).

Note that the maximum number of bonds selected (second element of the *NanoFingerprint*), $MAX = 10$, is the same as the maximum number of bonds of the nanoparticle, corresponding to atom 5 (Titanium atom connected to 6 Oxygen atoms and 4 Titanium atoms), and Local Structure $M(6,4)$.

Regarding the size, for small structures the shell includes the whole nanoparticle whatever the thickness is, such as the previous one. As the particle starts to grow, the shell thickness parameter starts to make an effect on the considered atoms. For example, on Figure 2.4 it is shown the 3D structure of the nanoparticle TiO_2 with size 1.4 nm, this is, 14 Å (TiO2_007). On the left, the whole nanocompound was considered whereas on the right, a shell thickness

of 4 Å was imposed. From 123 total atoms, the shell is composed of 116 atoms (the core has 7 atoms).

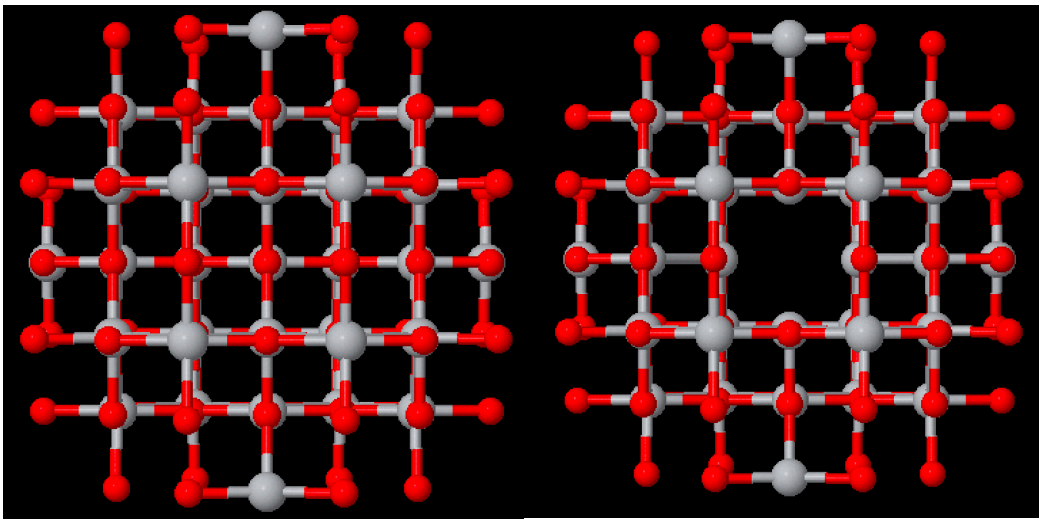


Figure 2.4. Comparison of the nanoparticle TiO_2 of size 1.4 nm with core (left), and without (right).

On Table 2.4 it is presented the extracted *NanoFingerprint* from the nanocompund TiO_2 with size 0.4 nm (in *verbose* format as the one on Table 2.3), and it is compared to its 3D nanoparticle structure and XYZ format (TiO2_002).

On this case, regarding to coordination, there are 6 singly coordinated O sites (marked as $O(0,1)$) and 1 six-fold coordinated Ti site (marked as $M(6,0)$). Moreover, the local environment is also captured by the combination of pairs of O and Ti sites: there are 6 singly coordinated O sites ($O(0,1)$) connected to a six-fold Ti site.

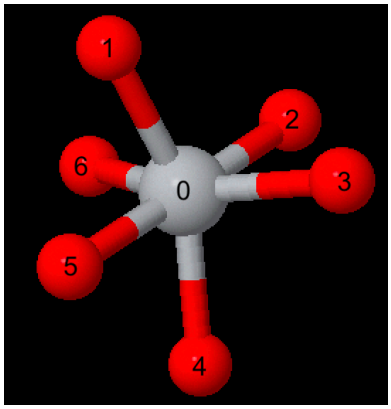
3D Structure	NanoFingerprint	XYZ format
	Shell: 100 MaxBonds: 6 Size: 3.845876 Atomic: 22 O: 6 M: 1 7-> O[1]: 6 18-> M[6]: 1 20-> O[0,1]: 6 110-> M[6,0]: 1 5011-> O[0,1]_M[6,0]: 6	7 O2Ti O -1.8650 -0.0000 -0.4685 O 0.0000 -1.8650 0.4684 O 0.0000 0.0000 -1.8740 Ti 0.0000 0.0000 0.0000 O 0.0000 0.0000 1.8740 O 0.0000 1.8650 0.4684 O 1.8650 0.0000 -0.4685

Table 2.4. 3D structure of the nanoparticle of TiO_2 with size 0.4 nm (left), its *Structural NanoFingerprint* generation (center) and the nanoparticle described in XYZ format (right).

Note that the maximum number of bonds selected (second element of the *NanoFingerprint*), $MAX = 6$, is also the maximum number of bonds of the nanoparticle, corresponding to node 0 (a Titanium atom connected to 6 Oxygen atoms), and Local Structure $M(6,0)$.

In the case of the shell of this nanoparticle, as well as the one on Table 2.3 (TiO_2 of size 0.6 nm), due to its size it always describes the whole nanoparticle regardless the thickness.

2.4. References

[1] Y. Çetin, B. Martorell and F. Serratosa, New descriptors in toxicology prediction of nanomaterials: Using quasi-ab initio MD simulations for the estimation of aqueous ZnO and TiO2 surface structure parameters, Nanotox2021, 2021.

[2] C. Garcia-Hernandez, A. Fernández and F Serratosa, Learning the Edit Costs of the Graph Edit Distance Applied to Ligand-Based Virtual Screening, Current Topics in Medicinal Chemistry, 20, pp: 1-11, 2020.

[3] C. Garcia-Hernandez, A. Fernandez and F. Serratosa, Ligand-Based Virtual Screening Using Graph Edit Distance as Molecular Similarity Measure, Journal of Chemical Information and Modelling, 59 (4), pp: 1410-1421, 2019.

[4] Tämm, K.; Sikk, L.; Burk, J.; Rallo, R.; Pokhrel, S.; Mädler, L.; Scott-Fordsmand, J. J.; Burk, P.; Tamm, T. Parametrization of nanoparticles: development of full-particle nanodescriptors. Nanoscale 2016, 8, 16243–16250.

[5] Herraez, A. (2021, November 28). File formats/Formats/XYZ. Jmol Wiki. Web link: https://wiki.jmol.org/index.php/File_formats/Formats/XYZ (Accessed: 24/03/2023).

[6] Jesus M. Castagnetto M. (Mon, 21 Oct 1996). CCL: SUMMARY: Some Molecular File Format Descriptions. Web link (Accessed: 24/03/2023): http://www.ccl.net/chemistry/resources/messages/1996/10/21.005-dir/index.html

[7] Chatzigoulas A., Karathanou K., Dellis D., Cournia Z. NanoCrystal: A web-based crystallographic tool for the construction of nanoparticles based on their crystal habit. J Chem Inf Model. 2018; 58(12):2380-2386, 10.1021/acs.jcim.8b00269.

[8] Burk, J.; Sikk, L.; Burk, P.; Manshian, B.; Soenen, S. J.; Scott-Fordsmand, J. J.; Tamm, T.; Tämm, K. Fe-doped ZnO nanoparticle toxicity: assessment by a new generation of nanodescriptors. Nanoscale, 2018, 10, 21985- 21993. DOI: 10.1039/C8NR05220D

[9] Tämm, K.; Sikk, L.; Burk, J.; Rallo, R.; Pokhrel, S.; Mädler, L.; Scott-Fordsmand, J.J.; Burk, P.; Tamm, T. Parametrization of nanoparticles: development of full-particle nanodescriptors. Nanoscale, 2016, 8, 16243-16250. DOI: 10.1039/c6nr04376c

[10] Satopaa, V., Albrecht, J., Irwin, D., & Raghavan, B. (2011, June). Finding a" kneedle" in a haystack: Detecting knee points in system behavior. In 2011 31st international conference on distributed computing systems workshops (pp. 166-171). IEEE.

CHAPTER III
THE ALGORITHM TO GENERATE NANOFINGERPRINTS

On this chapter the algorithm to correctly generate *Structural NanoFingerprints* is presented and deeply explained.

Firstly, an actual *NanoFingerprint* generator is introduced. Due to its shortcomings, there is the motivation to design a new algorithm. Then, two algorithms designed to find a subgraph into a graph and a Local Structure into a graph are presented, using the VF3 library, a graph matching algorithm [1][2]. These algorithms have been designed to help in the *NanoFingerprint* generation using the VF3 library; however, making this first approach is discarded due to a problematic found on its main concept and explained why (misleading approach).

Finally, the second and definitive approach to generate *NanoFingerprints* is presented.

3.1. State of the art

The history of *Structural NanoFingerprints* is very recent as it was introduced on section 1.1. Context of the Master Thesis, they were presented in *NanoCommons* congress, 2022, in Cyprus.

Therefore, only the public webserver called ATENA[6] generates *Structural NanoFingerprints* given the needed input parameters. Through it, the scientific community has seen its validity and usefulness. This public webserver has also other tools based on the 3D structure of the nanocompounds, such as toxicity prediction.

Nevertheless, the webserver *Structural NanoFingerprint* generator was implemented in *Neo4j* and there is the problem of the low scalability of the algorithm when structures are large, being computationally very expensive. In addition, it has been reported to have some problems and there is no public code available so it can be easily translated to other languages, such as C.

For these reasons, there is the motivation to implement an algorithm to generate Structural *NanoFingerprints* in a powerful, fast, and widely used programming language, such as C.

[6] https://atena.urv.cat/model/

3.2. First approach. The VF3 library

The first approach to implement a *Structural NanoFingerprint* generator was tried using the VF3 library (*vf3lib*), a graph matching algorithm, specialized in solving graph isomorphism and graph-subgraph isomorphism, developed by the MIVIA Research Group from the University of Salerno (UNISA) [1][2]. The *vf3lib* is publicly available at GitHub[7].

On this section, firstly, it will be explained the concept of subgraph isomorphism. Then, the VF3 library and its format files. Using it, a C project to effectively count the number of apparitions of a subgraph into a graph has been made. This program is evolved to a new version to count the number of apparitions of a *Local Structure* into a graph. Both algorithms are detailed. However, it is not further evolved to a S*tructural NanoFingerprint* generation algorithm, since the approach is misleading. Closing the section, this problem will be furtherly explained on 3.2.6. Examination of the results.

3.2.1. Subgraph isomorphism

In theoretical computer science, the subgraph isomorphism problem is a computational task in which two graphs G and H are given as input, and one must determine whether G contains a subgraph that is isomorphic to H. It is NP-complete [3].

An isomorphism of graphs G and H is a bijection between the vertex sets of G and H:

$$f: V(G) \rightarrow V(H)$$

Such that any two vertices u and v of G are adjacent in G if and only if $f(u)$ and $f(v)$ are adjacent in H [4]. This kind of bijection is commonly described as *edge-preserving bijection*, in accordance with the general notion of isomorphism being a structure-preserving bijection. If an isomorphism exists between two graphs, then the graphs are called isomorphic and denoted as $G \approx H$.

Graphs, in our case, are understood to be undirected non-labelled non-weighted graphs.

On Table 3.1 two graphs that are isomorphic are shown, as well as the isomorphisms between them, despite their different looking drawings.

[7] https://github.com/MiviaLab/vf3lib

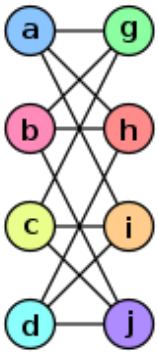
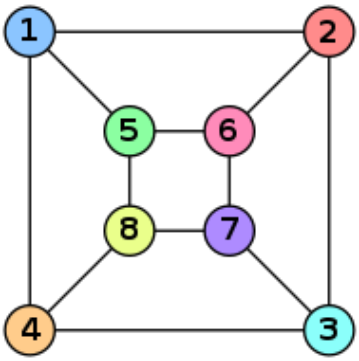
Graph G	Graph H	An isomorphism between G and H
		$f(a) = 1 \quad f(g) = 5$ $f(b) = 6 \quad f(h) = 2$ $f(c) = 8 \quad f(i) = 4$ $f(d) = 3 \quad f(j) = 7$

Table 3.1. Isomorphic graphs and the isomorphisms between them [4].

3.2.2. The VF3 library

The *vf3lib* is a software library containing all the currently published versions of VF3. In particular, it is able to solve both testing and listing problems, being able to determine not only if the isomorphism exists (testing) but also where and how many times a pattern graph is present inside the target graph (listing). On this work, the listing problem solving is the one of interest and the one that has been used.

VF3 has a low memory requirement (linear with respect to the graphs size) and its efficiency with respect to state-of-the-art algorithms is evident, especially when dealing with very huge and dense graphs. It is confirmed to be the fastest known algorithm available in the literature to solve graph and subgraph isomorphism on large and dense graphs.

The library is written in C++11 and contains the official implementations of VF2-Plus (previous version to VF3), improving its ability to deal with larger and denser graphs, VF3, VF3L and VF3P realized by the authors.

The *vf3lib* provides a *Makefile* that produces three different executables:

- VF3 (*vf3*). The algorithm with all the heuristics.
- VF3L (*vf3l*). A lightweight version, where the look-ahead is deactivated. This version fits for sparse or small graphs.
- VF3P (*vf3p*). A parallel version of VF3L, designed to effectively speed-up the algorithm on multicore architectures. It must be used when the problem is really hard.

To generate the executables, from the Terminal and inside the *vf3lib* main folder, where the *main.cpp* program is located, the instruction *make conversion* has to be executed. Then, the previous mentioned executables are successfully created inside the bin folder.

In the case of this project, only the *vf3* executable has been used. The input needed parameters of the program, in order, are:

1. *Subgraph*, in VF file format.
2. *Graph*, in VF file format.

The graph and subgraph must be specified in a specific format (VF) that is explained in the subsection 3.2.3. The VF file format. For instance, note that the all the lines contain the same node ids, but matched in a different order. For this reason, it is needed to count just the non-repeated number of solutions, regardless the node ordering. On this case, all the lines would count as just one apparition.

To execute the program from terminal, such as *vf3*, given a graph (*graph.grf*) and a subgraph (*subgraph.grf*) located inside the same folder, the following command line is needed to be executed:

./vf3 subgraph.grf graph.grf

The standard output provided by the algorithm is, in order:

1. Number of solutions found.
2. Time to find the first solution.
3. Time to find all the solutions.

This output, however, is not enough for the case of our implementation since it contains repeated solutions (in different order).

In addition, some additional parameters can be added to the command line. In the case of this approach, to thoroughly examine the solutions found by the algorithm, it was specified the parameter -s right after the executable and an output text file at the end:

./bin/vf3 -s subgraph.grf graph.grf out.txt

To obtain an output text file with all the solutions found. On Figure 3.1 it is shown a part of the output file of the *vf3* program given a graph and subgraph in VF file format. Each line contains a solution in which every pair of nodes ids are separated by colons (:). The first node from each pair indicates the node from the graph, while the second node indicates the node from the subgraph. Inside each pair, the node ids are separated by comma (,). There are as many pairs of nodes as subgraph containing nodes. In this way, each line contains a solution in which the subgraph is present inside the graph, indicating the matching nodes. As it was explained before, it is a listing problem solution.

```
Solution Found
2,0:9,1:11,2:12,3:13,4:14,5:24,6:
2,0:9,1:11,2:12,3:13,4:24,5:14,6:
2,0:9,1:11,2:12,3:14,4:13,5:24,6:
2,0:9,1:11,2:12,3:14,4:24,5:13,6:
2,0:9,1:11,2:12,3:24,4:13,5:14,6:
2,0:9,1:11,2:12,3:24,4:14,5:13,6:
2,0:9,1:13,2:12,3:11,4:14,5:24,6:
2,0:9,1:13,2:12,3:11,4:24,5:14,6:
2,0:9,1:13,2:12,3:14,4:11,5:24,6:
2,0:9,1:13,2:12,3:14,4:24,5:11,6:
```

Figure 3.1. Part of the output file of the *vf3* program.

For instance, note that the all the lines contain the same node ids, but matched in a different order. For this reason, it is needed to count just the non-repeated number of solutions, regardless the node ordering. On this case, all the lines would count as just one apparition.

3.2.3. The VF file format

It is the standard input graph and subgraph file format for *vf3lib*. The file can be specified in both .txt and .grf extension. The VF format is structured in the following way:

- On the first line, there is the number of nodes (N) of the graph.
- On the subsequent N lines, there are the node ids, one per line, followed by the node attribute. The node ids must be in the range from 0 to N-1. On the case of this application, the node attributes are the atomic numbers of the node atom.
- Next, for each node, there is a line containing the number of edges (E) coming out of the node, followed by E lines, one for each edge, containing the ids of the edge ends and the edge attribute (optional).

Additionally, there can be blank lines, and lines starting with # (comments), that are ignored. On Table 3.2 it is presented a part of the nanocompound TiO_2 of size 0.4 nm (TiO2_002) in VF file format.

```
# Number of nodes
7
# Node ids and attributes
0 8
1 8
2 8
3 22
4 8
5 8
6 8
# Edges coming out of node 0
1
0 3
# Edges coming out of node 1
1
1 3
...
```

Table 3.2. Part of the nanocompound TiO_2 of size 0.4 nm in VF file format.

3.2.4. The program to address the subgraph search

This program was designed to count the number of appearances of a subgraph into a graph, both in XYZ format, without repetitions, and in addition generate an output graph in XYZ format containing a modification of the input graph just containing the subgraph nodes (*output.xyz*). The program was coded in C language from scratch, using the *vf3* program, and its name was *substructuresearch_main.c*.

The input parameters of this program are, in order:

- *Graph*, in XYZ format.
- *Subgraph*, in XYZ. It can also be in VF format.
- Name of the *output graph*, in XYZ format.
- *Thickness* of the shell, measured in angstroms (Å).

Then, the programs outputs:

- The correctly generated *output graph*, in XYZ format.
- The *number of appearances* of the subgraph into the graph and the *number of coincident nodes* in both graph and subgraph on the Terminal.
- If there is any error, it specifies which one.

To execute the program from terminal, if the executable is located in the same folder as the input files, the following command line is needed to be executed:

./substructuresearch_main graph.grf subgraph.grf out_file.xyz thickness

The main steps of the algorithm are described hereafter. In parentheses, the name of the functions in charge of the step is detailed.

1. Given the graph (in XYZ format) and thickness (in Å), a reduced graph just containing the nodes of the shell is generated (*graphToThickness*).
2. The graph generated on the previous step (just containing the shell) is converted to VF format (*xyzToGraph*).
3. Given the input subgraph, if it is in XYZ format, it is converted to VF format (*xyzToGraph*). Otherwise, this step is skipped.
4. From the subgraph in VF format, it is obtained the number of atoms of the subgraph (*readNumAtomsGraph*).
5. With the graph in VF format generated on step 2 and the subgraph in VF format as inputs, it is called the *vf3* program (*vf3Execution*).
6. From the output file of the *vf3* program and the number of atoms of the subgraph, the number of appearances of the subgraph into the graph is obtained (*countAppearances*).
7. From the output file of the *vf3* program and the graph generated on step 1 (just containing the shell), the output graph in XYZ format containing a modification of the input graph just containing the subgraph nodes is generated (*newGraph*). In addition, the number of coincident nodes in both graph and subgraph, this is, the number of atoms of the new graph, is obtained.

On Figure 3.2 it is shown the main scheme of the subgraph search algorithm.

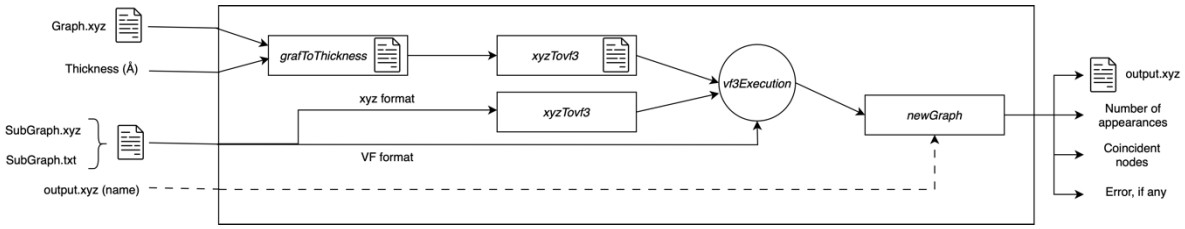


Figure 3.2. Main scheme of the subgraph search algorithm.

On Table 3.3 a brief description, the input and output parameters of the mentioned functions, and some additional ones, are described.

Function	Description	Input	Output
graphToThickness	Reduces the graph by just getting the nodes belonging to the shell, defined by the thickness (in nm).	Path of the graph file (char[100]), thickness (int), path name of the output graph file (char[100])	Negative number if there is any error, 0 otherwise.
xyzToGraph	Converts a graph to VF format (.txt). In addition, counts the number of atoms of the graph.	Path of the graph file (char[100]), path of the output graph file (char[100]), number of atoms of the graph (int, by reference)	Negative number if there is any error, 0 otherwise.
readNumAtomsGraph	Counts the number of atoms of the graph.	Path of the graph file (char[100]), number of atoms of the graph (int, by reference)	-1 if there was an error opening the read file, 0 otherwise.
vf3Execution	Executes the executable *vf3*, which returns its results on a text file	Path of the graph file (char[100]), path of the subgraph file (char[100]), path of the output file (char[100])	Negative number if there is any error, 0 otherwise.

countAppearances	Counts the number of appearances of a subgraph into a graph	vf3 generated path file (char[100]), number of atoms of the subgraph, number of appearances (int, by reference)	Negative number if there is any error, 0 otherwise.
newGraph	Transforms a graph into another one just containing the subgraph nodes	vf3 generated path file (char[100]), path of the graph file with just the nodes of the shell (char[100]), path name of the output graph file (char[100]), number of coincident nodes (int, by reference)	Negative number if there is any error, 0 otherwise.
calculateDistance	Calculates the distance between two atoms to know if there is a bond. Used in the functions *graphToThickness* and *xyzToGraph*.	Coordinates x, y, z of the two atoms (nodes in the graph) (float)	Distance between the two atoms (nodes)
distAtom	Returns the distance threshold between two atoms. Used in the function *xyzToGraph.*	Atomic number of the two atoms (nodes of the graph) (int)	Distance threshold between the two atoms (nodes)
toAtomic	Returns the atomic number of an atom (node). Used in the function *xyzToGraph*.	Symbol of the atom (char[2])	Atomic number of the atom (node)

Table 3.3. Brief description, input, and output parameters of the used functions on the subgraph search program.

The value from the variables passed by reference is modified inside the function.

It has to be detailed how the shell is obtained from the function *graphToThickness.* Firstly, it is obtained the centre, which is nearly the coordinate (0,0,0) for all nanoparticles; however, it is computed to exactly find its position. Then, it is found the furthest atom from the centre ($dist_{max}$). Then, it is computed the $core_{distance}$ (Å) as:

$$core_{distance}(\text{Å}) = dist_{\max(\text{Å})} - (thickness\ (\text{Å}) + \ \text{E})$$

Where *E* is a distance of 4 Å empirically obtained, and its utility is to ensure that atoms belonging to the shell maintain their bonds to atoms inside the core. The objective is to maintain all atoms which distances from the center are above the $core_{distance}$, but just consider on the *NanoFingerprint* count those in the thickness (shell), counting their bonds to atoms on the core. This process is explained on 3.3. Final approach.

Another defined function was not specified on Table 3.2, the *cmpfunc* function [5]. It is used on the *qsort* function, on the *countAppearances* function, as:

qsort(values, numatoms_subgraph, sizeof(int), cmpfunc);

On the function *countAppearances*, the node ids from the graph on the output vf3 from each line are ordered in ascending order using *qsort,* so they can be compared to previous lines and determine if it is a new appearance or not. The compare argument is a pointer to the comparison function, which is called with two arguments that point to the elements being compared. The application shall ensure that the function returns an integer less than, equal to, or greater than 0, if the first argument is considered respectively less than, equal to, or greater than the second. If two members compare as equal, their order in the sorted vector is unspecified.

Giving detail to the output (returned parameter) meaning, on Table 3.4 the errors of the main program *substructuresearch_main.c* are detailed, on Table 3.5 the *graphToThickness* errors, on Table 3.6 the *xyzToGraph* errors, on Table 3.7 the *countAppearances* errors and, on Table 3.8, the *newGraph* errors.

Returned parameter	Error
0	No errors on the execution
-1	Incorrect input parameters
-2	Error in opening the read file
-3	Atomic number of the element not found

-4	Error on the vf3 execution
-5	Error in opening the write file
-6	Thickness value too low. Not atoms found.
-7	Memory allocation error

Table 3.4. Main program (*substructuresearch_main.c*) error description.

Returned parameter	Error
0	No errors on the execution
-1	Error in opening the read file
-2	Error in opening the write file
-3	Thickness value too low. Not atoms found
-4	Memory allocation error

Table 3.5. *graphToThickness* function error description.

Returned parameter	Error
0	No errors on the execution
-1	Error in opening the read file
-2	Atomic number of the element not found
-3	Error in opening the write file
-4	Memory allocation error

Table 3.6. *xyzToGraph* function error description.

Returned parameter	Error
0	No errors on the execution.
-1	Error in opening the read file
-4	Memory allocation error

Table 3.7. *countAppearances* function error description.

Returned parameter	Error
0	No errors on the execution
-1	Error in opening the read file
-2	Memory allocation error
-3	Error in opening the write file

Table 3.8. *newGraph* function error description.

The *distAtom* function returns the distance between two atoms when they form a bond, depending on its atomic number. This distance is specific for metal oxide nanoparticles and depends on the two atoms forming the bond. The elements considered on the study, which form metal oxide nanoparticles, have been Oxygen, Titanium, zinc, iron, aluminium, and copper. On Table 3.9 the symbols and atomic numbers of these elements are detailed and on Table 3.10 the distances between these atoms when they form a bond are specified (to atoms from the same type and atoms with Oxygen). These distances have been empirically obtained by using the *molviewer* MATLAB function and measuring them. For instance, on Figure 3.3 it is shown the nanoparticle In_2O_3 of size 30 Å, and the measurement of the bond distances of an Indium atom to another Indium atom and of and Indium atom to an Oxygen atom.

Element	Symbol	Atomic number
Oxygen	O	8
Titanium	Ti	22
Zinc	Zn	30
Iron	Fe	26
Aluminium	Al	13
Copper	Cu	29
Indium	In	49
Lanthanum	La	57
Silicon	Si	14

Zirconium	Zr	40
Yttrium	Y	39
Bismuth	Bi	83
Antimony	Sb	51
Niquel	Ni	28

Table 3.9. Elements that form metal oxide nanoparticles, its symbol and atomic number.

Bond	Distance (Å)	Bond	Distance (Å)
O-O	1.8	Ti-O	2.35
Ti-Ti	3	Zn-O	2.11
Zn-Zn	3.5	Fe-O	3.5
Fe-Fe	3.7	Al-O	2.2
Al-Al	2.86	Cu-O	3.9
Cu-Cu	3.1	In-O	2.4
In-In	3.5	La-O	3
La-La	4	Si-O	1.9
Si-Si	2.5	Zr-O	2.3
Zr-Zr	3.5	Y-O	2.3
Y-Y	3.6	Sb-O	2.7
Ni-Ni	2.4	Ni-O	2.3
Bi-O	3.1	Default	2.2

Table 3.10. Distances between atoms in metal oxide nanoparticles.

There was defined a default case, as an average distance, for those elements not figuring on the previous cases. However, for the nanoparticles presented on this Master Thesis, all distances are provided. Note that for some atoms of the same element the distances are not provided, such as for Bismuth (Bi) and Antimony (Sb), since any bond between them has been found on the nanoparticles.

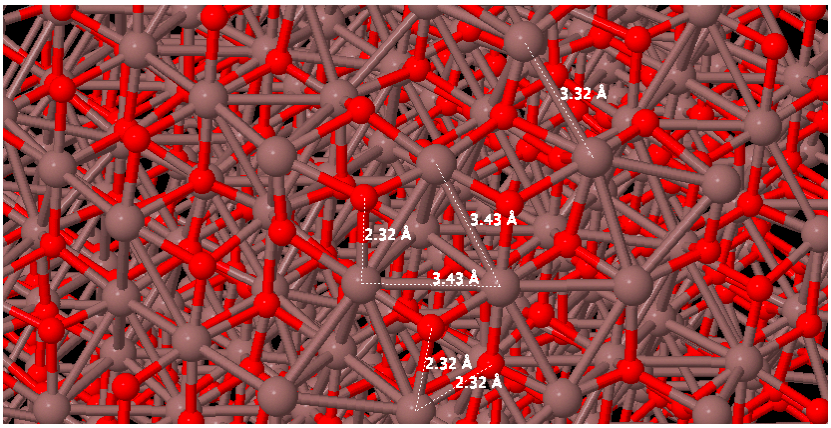


Figure 3.3. Visualization of In_2O_3 of size 30 Å. The distances between an Indium atom to an Oxigen atom and between two Indium atoms are measured in angstroms, using the *molViewer* instrument from MATLAB. Note that they can differ some angstroms; therefore, the largest one has been taken and rounded to the next decimal.

Regarding the functions *calculateDistance* and *distAtom*, on the case of their use on the function *xyzToGraph,* the purpose is to see if between two atoms a bond is there. In case that the distance returned by *distAtom* is less than the one returned by *calculateDistance*, there is a bond.

To observe an example, two nanoparticles of TiO_2 have been used to find one into the other one. On Figure 3.4 it is shown the visualization of TiO_2 of 2 nm. Then, on Figure 3.5 it is shown, on the right, the visualization of TiO_2 of size 0.6 nm whereas on the left it is shown the visualization of four appearances of TiO_2 of size 0.6 nm in the shell of the TiO_2 nanoparticle of size 2 nm with shell thickness of 0.4 nm (output of the *newgraph* function).

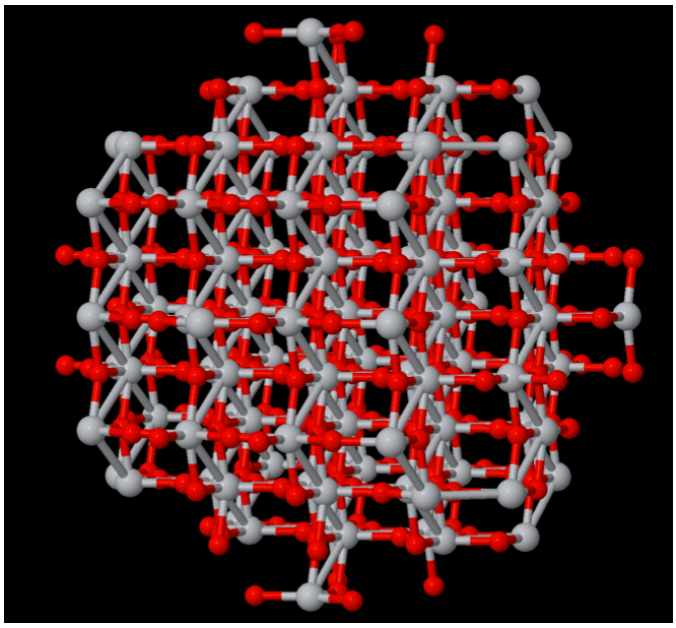


Figure 3.4. Visualization of TiO_2 of size 2 nm.

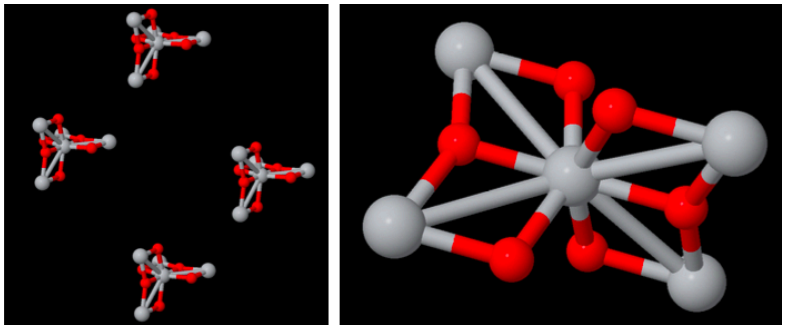


Figure 3.5. Visualization of TiO_2 of size 0.6 nm (right), visualization of the four appearances of TiO_2 of 0.6 nm in the shell (thickness 0.4 nm) of TiO_2 of 2 nm (left).

3.2.5. The program to address the local structure search

This program was designed to count the number of appearances of a local structure, defined as $O(x,y)$ or $M(x,y)$ (those defined in Section 2, the linking Information) into a graph, without repetitions on same nodes, and in addition generate an output graph in XYZ format containing a modification of the input graph just containing the local structure nodes found on the graph. The program is a modification of the *substructuresearch_main.c* and its name is *localstructuresearch_main.c*.

The input parameters of this program are, in order:

- *Graph*, in XYZ format.
- Name of the *output graph*, in XYZ format.
- *Thickness* of the shell, measured in angstroms (Å).
- *Atomic number* of the central atom of the local structure.
- Number of bonds to Oxygen atoms (*b1*).
- Number of bonds to metal atoms (*b2*).

Then, the programs outputs:

- The correctly generated *output graph*, in XYZ format.
- The *number of appearances* of the local structure into the graph and the *number of coincident nodes* in the graph and the local structure, on the Terminal.
- If there is any error, it specifies which one.

To execute the program from terminal, if the executable is located in the same folder as the input files, the following command line is needed to be executed:

./localstructuresearch_main graph.grf subgraph.grf out_file.xyz thickness atomic b1 b2

To migrate from the *substructuresearch_main.c* program to *localstructuresearch_main.c* program, a function to generate a subgraph from the local structure had to be implemented. The name of this function is *generateSubgraph*. Knowing the structure of a graph in VF format, its main purpose is to write on a .txt file the local structure as a graph.

On Figure 3.6 it is shown the main scheme of the Local Structure search algorithm.

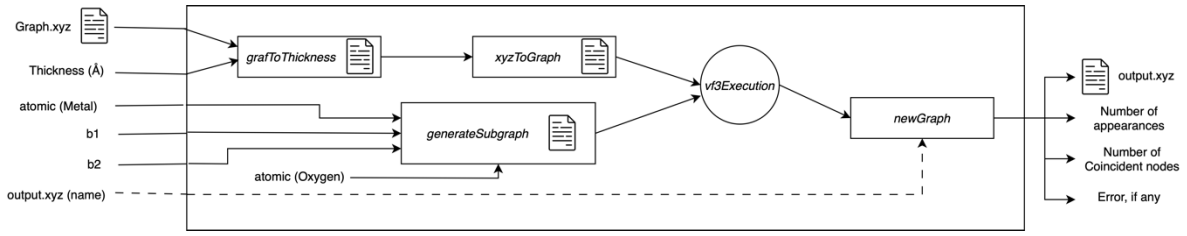


Figure 3.6. Main scheme of the Local Structure search algorithm.

On Table 3.11 the input and output parameters of the *generateSubgraph* function are described.

Input
• Path of the subgraph file to generate (char[100]) • Atomic number of an Oxygen atom (int) • Atomic number of the metal atom (int) • Local structure defined as atomic number of the central atom, bonds to Oxygen atoms and bonds to metal atoms (int[3]) • Number of atoms of the subgraph (int, by reference)
Output
-1: Error in opening the read file -2: Memory allocation error

Table 3.11. Input and output parameters of the function *generateSubgraph*.

3.2.6. Examination of the results

On this section it will be presented the results from the *localstructuresearch_main.c* program and the problematic of using the VF3 lib for the *Structural NanoFingerprint* generation.

On Table 3.12 the 3D-structure of the nanoparticle of TiO_2 of size 0.4 nm, the subgraph representing the local structure to search for, and its number of apparitions returned by the program *localstructuresearch_main.c* are presented.

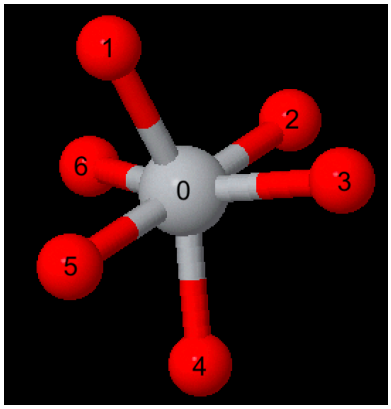
<table>
<tr><th>3D Structure</th><th>Subgraph</th><th>Local Structure</th></tr>
<tr><td rowspan="3"> 
</td><td rowspan="3">2
0 8
1 22
1
0 1
1
1 0</td><td>$O(0,1)$</td></tr>
<tr><td>Count</td></tr>
<tr><td>6</td></tr>
</table>

Table 3.12. 3D-structure of the nanoparticle of TiO_2 of size 0.4 nm, the subgraph representing the local structure to search for, and its number of apparitions returned by the program *localstructuresearch_main.c*. The returned number of coincident nodes is 7, since the subgraph matches have been present in all nodes of the graph.

As it can be seen, the output result is correct, since the local structure $O(0,1)$ defined by the previous subgraph appears 6 times.

However, the *VF3* approach is misleading for two main reasons:

1. On the first place, the use of a subgraph to be searched in a graph is problematic. Converting the local structure to a subgraph represents losing information about the central atom and its bonds to the surrounding Oxygen or metal atoms. Therefore, on the use of the *vf3* program a match of the subgraph into the subgraph may not represent the local structure previously defined. This problematic is presented on Table 3.13. As it can be seen, the result is 15, when it should have been 0. The reason is that the subgraph is 15 non-repeated times present into the graph, but it does not follow the definition of *Structural NanoFingerprint*, in which the central atom has exactly $x + y$ bonds.

2. On the second place, related to the first one, the local structure defining just a central atom and its bonds do not consider how the surrounding atoms are connected. Therefore, it can happen that the surrounding atoms are bonded between them and on the conversion from the local structure to the subgraph this is not considered.

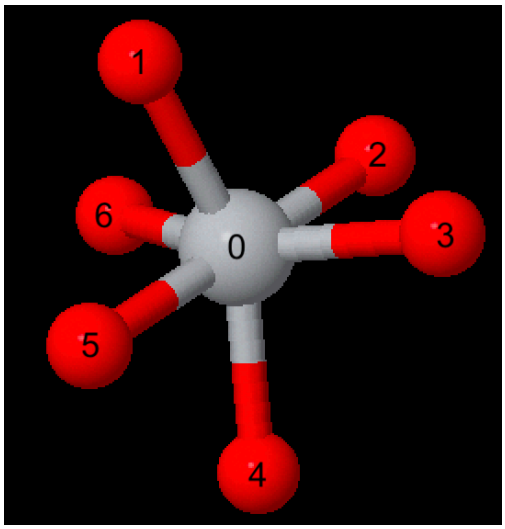
<table>
<tr><th>3D Structure</th><th>Subgraph</th><th>Local Structure</th></tr>
<tr><td rowspan="2"> 
</td><td rowspan="2">3
0 22
1 8
2 8
2
0 1
0 2
1
1 0
1
2 0</td><td>$M(2,0)$</td></tr>
<tr><td>Count</td></tr>
<tr><td></td><td></td><td>15</td></tr>
</table>

Table 3.13. 3D-structure of the nanoparticle of TiO_2 of size 0.4 nm, the subgraph representing the local structure to search for, and its number of apparitions returned by the program *localstructuresearch_main.c*. It is presented the first problematic: *M(2,0)* should be 0 instead of 15.

On Table 3.14 the 3D-structure of a nanoparticle of TiO_2 of size 0.6 nm and the results obtained by the program *localstructuresearch_main.c* for the local structure $O(0,2)$ are presented. Note that the result should be 4 instead of 6. To examine the result, on Figure 3.7 the output file of the *vf3* execution is shown. Note that on the solution, the nodes present are (9, 4, 0) and (8, 6, 3). The central node is the Oxygen (nodes 4 and 6), which in the two cases has more than two bonds and therefore the result is not correct (first problematic). To visually see better the result, on Figure 3.8 it is depicted the output graph of this case.

On the other hand, the local structure $O(0,2)$ central atom is located at nodes 10, 1, 7 and 8, which are not present on the output of the *vf3* execution. This is because of the second problematic and the subgraph definition: the metal atoms bonded to these Oxygens have also a bond between them.

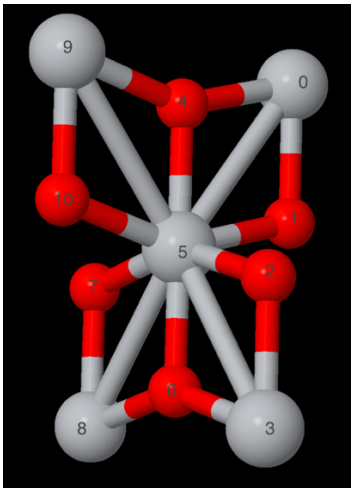
3D Structure	Subgraph	Local Structure
	3 0 22 1 8 2 8 2 0 1 0 2 1 1 0 1 2 0	$O(0,2)$
		Count
		2

Table 3.14. 3D-structure of the nanoparticle of TiO_2 of size 0.6 nm, the subgraph representing the local structure to search for, and its number of apparitions returned by the program *localstructuresearch_main.c*. On this case, both problematics have arisen. Therefore, *O(0,2)* should be 4 instead of 2.

```
Solution Found
4,0:0,1:9,2:
6,0:3,1:8,2:
6,0:8,1:3,2:
4,0:9,1:0,2:
4 4.62691e-06 8.72621e-06
```

Figure 3.7. *vf3* execution output file for the case shown on Table 3.13.

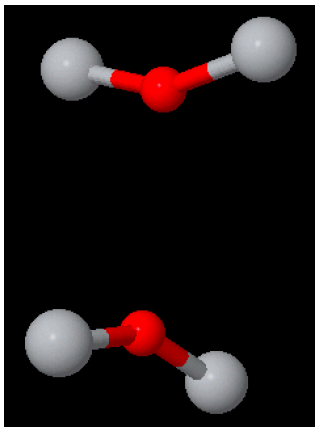


Figure 3.8. Output graph by the program *localstructuresearch_main.c* from the case presented on Table 3.14.

3.3. Final approach

Due to the problematics found using the VF3 library, another approach to the problem was taken without considering it.

The implementation has been done in C as well and it has the advantage from the previous one that it does not need any additional executable. Most of the previous functions, exactly or modified, are used on this new implementation, and the algorithm structure is not very far from the previous local structure search program, with the additional features regarding the *NanoFingerprint* generation.

Through this subsection it will be explained in detail the algorithm, including the four sections and structs needed to manage the implementation. In addition, the possible output versions of the *NanoFingerprint* generation will be presented.

The main program has received the name *nanofingerprint_generator_main.c* and has been designed to effectively generate a *Structural Nanofingerprint* for a specific metal-oxide nanoparticle.

The input parameters of this program are, in order:

- *Graph*, in XYZ format.
- *Thickness* of the shell, measured in angstroms (Å).
- Maximum number of bonds (*MAX*).

Then, the programs outputs:

- The correctly generated *Structural Nanofingerprint*, in two text files, one for each format: a *Non-Verbose* (NV) *NanoFingerprint*, and a *Verbose-Non-Zero* (VNZ) *NanoFingerprint*. The first one prints all the sections of the *NanoFingerprint* without explicitly identifying each element, one for each row. The second one specifies the meaning of each element and only prints the values different from 0 for sections 2, 3 and 4.
- On terminal, the execution time, and the paths of the generated *NanoFingerprints*.
- If there is any error, it specifies which one.

Additionally, another version of the program with an extra input argument was created, in which the parameter indicated the format of the output *NanoFingerprint*:

- *VZ (Verbose Zero)*. It is indicated, for each element, what is it describing. Moreover, from Section 1 to Section 4 it indicates the position of each element, starting on 1.
- *VNZ (Verbose Non-Zero)*. It is indicated, for each element, what is it describing. Moreover, from Section 1 to Section 4 it indicates the position of each element, starting on 1. In addition, elements that are 0 are not shown.
- *NV* (*Non-Verbose*). It does not indicate neither the definition of each element nor the position. It outputs the values directly, one different for each line.

To execute the program from terminal, if the executable generated is located in the same folder as the input files, the following command line is needed to be executed:

./nanofingerprint_generator graph.grf graph.grf thickness MAX

The main steps of the algorithm are described hereafter.

1. Given the graph (in XYZ format) and thickness (in Å), a reduced graph containing just the nodes of the shell is generated (*graphToThickness*). It is the same function used in the subgraph search and local structure search.
2. The graph generated on the previous step (just containing the shell), rather than being converted to VF format (and saved to a file), it is stored as a Graph struct, as detailed in 3.3.1. Graph structure. The function in charge of this task is *xyzToGraph*, which has been modified from the previous programs.
3. *Section 1* from Structural *NanoFingerprint* is computed, as described in 3.3.2. Section 1 computation. *Section 2* and *Section 3* from Structural *NanoFingerprint* are computed, as described in 3.3.3. Section 2 and Section 3 computation. *Section 4* from Structural *NanoFingerprint* is computed, as described in 3.3.5. Section 4 computation.
4. Finally, the *NanoFingerprint* is write to the text file, in NV and VNZ formats (*generateNanoFingerprint* function).

The execution time (hours, minutes, seconds, and milliseconds) and the folder where the *NanoFingerprints* are saved are outputted on Terminal.

Note that the function *newGraph* is not used anymore since the generation of a new graph just containing the nodes of each *Local Structure* found on the nanoparticle would imply the generation of $2 \cdot MAX + 2 \cdot (MAX + 1)^2 + 3 \cdot (MAX + 1)^4$ graphs.

On Figure 3.9 it is shown the main scheme of the Local Structure search algorithm.

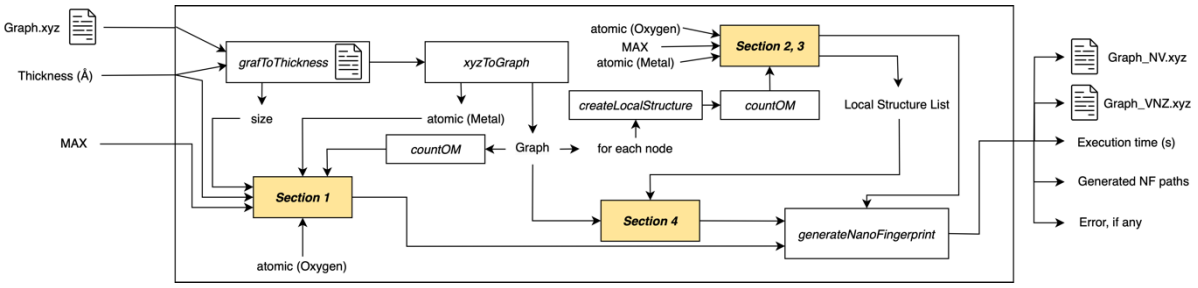


Figure 3.9. Main scheme of the *NanoFingerprint* generation algorithm.

On Table 3.15 a brief description of the mentioned functions and the rest of implemented ones that contribute on the *NanoFingerprint* generation are described.

Function	Description
graphToThickness	Reduces the graph by just getting the nodes belonging to the shell, defined by the thickness (in nm). It also calculates the maximum distance from the centre to an atom, from which it is calculated the size (two times this value), and the number of atoms of the shell (both passed by reference).
calculateDistance	Given the x, y, z coordinates from two atoms, it calculates the distance between two atoms (float) to know if there is a bond. Used in the functions *graphToThickness* and *xyzToGraph*.
createGraph	Given the number of atoms of the shell (obtained on *graphToThickness*), initialize the Graph Struct.
xyzToGraph	Stores the *xyz* nanoparticle shell as a Graph Struct. If no error is raised, it is removed the .txt thickness file on the main, using *remove*. It obtains the atomic number of the metal, passed by reference.

toAtomic	Given the symbol of an atom, it returns the atomic number (int) of an atom (node). Used in the function *xyzToGraph*.
addAtomic	Given a node and an atomic number, it stores the atomic number of the corresponding node to the Graph struct. Used in the function *xyzToGraph*.
distAtom	Returns the distance threshold between two atoms (float), given its atomic number. Used in the function *xyzToGraph.*
addEdge	Given two nodes, it adds an edge between them on the Graph (undirected). Used in the function *xyzToGraph.*
numVertices	Returns the number of atoms of the *Graph* (int). Used in the main.
getAtomics	Returns the atomic number of each atom of the *Graph* (int*). Used in the main.
isShell	Indicates if an atom passed by parameter is part or not of the shell, returning a '1' if it is and a '0' if it isn't. Used in the main.
setShell	Used in the *xyzToGraph* function, it sets an atom as being part of the shell ('1') or not ('0').
countOM	Given the atomic number of each node and the number of atoms of the Graph, it calculates the number of Oxygen atoms and metal atoms (int, passed by reference). Used in the main.
createLocalStructureList	Given the maximum number of elements (Local Structures) that it can have, it initialize the elements of a Local Structure List Struct. Used in the main.
degreeOfNode	Given a node, it returns its degree (int). Used in the main and in the functions *getSurroundingNodesAtomic* and *getSurroundingNodes*.

getSurroundingNodesAtomic	Given an atom (node) of the graph, it returns the atomic number of each neighbour atom (to which it has a covalent bond) (int*). Used in the main.
createLocalStructure	Given an atomic number (central atom), the number of connections to Oxygen atoms and the number of connections to metal atoms, it creates a Local Structure Struct. Used in the main.
addLocalStructure	It adds a Local Structure struct to the Local Structure List struct, also saving the node in which the Local Structure is. Used in the main.
atomicOfNode	Given a node (atom) of the graph, it returns its atomic number (int). Used in the main and in the function *getSurroundingNodesAtomic.*
getNumLocalStructures	Given a Local Structure List, it returns the number of Local Structures that it contains (int). Used in the main.
getNodeAtPosition	Given a Local Structure List and an index i, it returns the i-th Local Structure central atom (node) from the list (int). Used in the main.
getSurroundingNodes	Given an atom (node) of the graph, it returns the neighbour atoms (to which it has a covalent bond) (int*). Used in the main.
addElement	It correctly adds a count to the corresponding section 4 index, given the Local Structure List and the position of the two Section 4 Local Structures present at the list. Used in the main. It is furtherly explained on 3.3.5. Section 4 computation.
generateNanoFingerprint	Given the four sections vectors, it writes the *NanoFingerprint* to a specified file. It can be specified the format (NV, VNZ or VZ).

Table 3.15. Brief description of the used functions on the *NanoFingerprint* generation program.

The output generation errors are just considered on the main program and the functions *grafToThickness*, *xyzToGraph* and *generateNanoFingerprint*. On the case of the main program, *nanofingerprint_generator_main.c*, the errors are the same as the previous approach, but without considering the *vf3* execution error. They are detailed on Table 3.16.

On the case of the function *grafToThickness*, the errors are the same ones as described on Table 3.5 from 3.2.4. The program to address the subgraph search. Analogously, the errors of *xyzToGraph* are detailed on Table 3.6, but without the error referred to 'Error in opening the write file' (Error -3).

Returned parameter	Error
0	No errors on the execution
-1	Incorrect input parameters
-2	Error in opening the read file
-3	Atomic number of the element not found
-5	Error in opening the write file
-6	Thickness value too low. Not atoms found.
-7	Memory allocation error

Table 3.16. Main program (*nanofingerprint_generation_main.c*) error description.

Finally, regarding to the function *generateNanoFingerprint*, just error -1 is considered, which refers to error -5 in the main program (error in opening the write file).

Functions have been implemented on the file *Functions.c*, and the includes, constants, and function declarations are declared on the file *Functions.h*. Finally, there is an additional file, *Structs.h*, including the definition of the Local Structure, Local Structure List, and Graph structs.

3.3.1. Graph structure

The graphs to be obtained a *Nanofingerprint* from have been stored as a Graph struct. It includes the following data:

- *Number of vertices* of the graph.
- A vector indicating the *atomic number* of each atom (node) in the graph.

- A vector with the *degree* of each atom (node). It indicates the number of nodes in which each atom has a bond with.
- A vector indicating if the atom is part of the shell (value '1') or not (value '0').
- An *adjacency list* of nodes, in order to store the connections of the graph.

The *adjacency list* is made of *node structs*. Each *node struct* stores:

- The *vertex* of the node (index).
- The *next node struct* to which it is connected (*Null* in case it is the last node of the chain).

The process to store a nanoparticle in XYZ format to a Graph Struct is very similar to the process of converting a nanoparticle from XYZ format to VF format; firstly, the XYZ file is read, where the x, y, z coordinates and the atomic number of each node are stored. Then, an edge is formed if the distance between two atoms a_1 and a_2, calculated as the Euclidean distance (function *calculateDistance*):

$$d = \sqrt{\left(x_{a_2} - x_{a_1}\right)^2 + \left(y_{a_2} - y_{a_1}\right)^2 + \left(z_{a_2} - z_{a_1}\right)^2}$$

Is lower than the maximum needed distance between two atoms in order to form a bond, as described in Table 3.9 from 3.2.4. The program to address the subgraph search.

3.3.2. Section 1 computation

With the stored *Graph struct*, and from the atomic number of each atom (node in the graph), the number of Oxygen atoms and metal atoms are computed.

The input parameters of the *NanoFingerprint*, together with the size (maximum distance between two atoms of the graph) and the previously commented computations, are part of Section 1.

3.3.3. Section 2 and Section 3 computation

Section 2 and *3* were computed at the same time. Two variants were designed, despite of the computation time was very similar in both.

The first variant has the algorithm steps as follows.

1. For each atom, its degree, and its neighbour atoms atomic numbers (to which they have a covalent bond) are obtained.

2. For each atom, given the degree and the neighbour atoms atomic numbers, the number of connections to Oxygen atoms and metal atoms is computed.

3. If the degree is equal or lower the number of *MAX*, the local structure defined as the atomic number of the atom, its number of connections to Oxygen ($O_{appearances}$) and metal ($M_{appearances}$) atoms is added to a *list of local structures* (*addLocalStructure* function). This list is needed to compute *Section 4*, and it is described in 3.3.4. Local Structure struct.

4. Having two vectors, one for Section 2 and another for Section 3, counting the number of appearances of each corresponding local structure, if the atomic number of the node corresponds to the atomic number of an Oxygen atom, and the degree is lower or equal than *MAX*, Section 3 vector at index $(MAX+1)\cdot O_{appearances} + M_{appearances}$ (corresponding to local structure $O(O_{appearances}, M_{appearances})$) is increased one unit. If, in addition, the degree is higher than 0, Section 2 vector at index *degree-1* is increased one unit (corresponding to local structure *O(degree*)).

5. Analogously, if the atomic number of the node corresponds to the atomic number of the metal atom, and the degree is lower or equal than *MAX*, Section 3 vector at index $(MAX+1)^2+(MAX+1)\cdot O_{appearances} + M_{appearances}$ (corresponding to local structure $M(O_{appearances}, M_{appearances})$) is increased one unit. If, in addition, the degree is higher than 0, Section 2 vector at index *MAX+degree-1* is increased one unit (corresponding to local structure *M(degree*)).

The second variant algorithm is described as follows.

1. For the atomic number of an Oxygen and metal atom, for and index *j* and *k* from 0 to *MAX*, a *Local Structure* is created (*createLocalStructure* function). An additional index *i* is initialized to 0.

2. Then, the number of appearances of each local structure is counted (*countAppearances* function). This is done by iterating all the nodes on the graph. If the atomic number of a certain node (atom) is the same as the one in the *Local Structure*, the degree is obtained. If the degree equals the number of connections (bonds to Oxygen and metal connected atoms) of the *Local Structure*, the neighbour nodes (connected) of the node are obtained. Finally, if the Oxygen connections of the node are the same as the Oxygen

connections on the *Local Structure*, a counter is increased one unit, and the *Local Structure* is added to a *list of Local Structures*, as the first variant algorithm (*addLocalStructure* function).

3. Section 3 vector at index *i* is updated with the number of appearances counted, corresponding to the Local Structure *O(j,k)* or *M(j,k)*, depending on which atomic number forms the Local Structure previously created. Index *i* is increased one unit.

4. Finally, if the addition of j and k is different to 0 and lower or equal to MAX, Section 2 at index $j+k-1+MAX \cdot a$ is added the number of appearances counted, where *a* corresponds to 0 if the atomic number of the Local Structure is the Oxygen one, 1 otherwise.

On both, it is checked that the central atom is part of the shell since it could belong to the most external part of the core. If this happens, this atom is not considered for the counts of Section 2 and 3. This is because, as explained on 3.2.4. The program to address the subgraph search, the Graph has the shell atoms but also the core atoms connected to shell atoms.

Since the first algorithm is simpler than the second one, it was the one that it was used.

3.3.4. Local Structure struct

In a similar way than a *Graph*, *Local Structures* generated during the *NanoFingerprint* generation have been stored as a *Local Structure* struct, which includes:

- *Atomic number* of the central atom of the local structure.
- Number of bonds to Oxygen atoms.
- Number of bonds to metal atoms.

Moreover, in Section 3 generation the Local Structures that are present on the nanoparticle are needed to be saved to compute Section 4. Therefore, a *Local Structure list* struct has been designed, which includes:

- The number of *Local Structures* present at the list.
- A collection of *Local Structure* structs.
- A vector with the central node (atom) of each *Local Structure* struct saved in the *Local Structure* list.

3.3.5. Section 4 computation

Section 4 was computed using the list of *Local Structures* found in the nanoparticle during the computation of Section 2 and 3. The algorithm of this section is described as follows:

1. For each *Local Structure struct* on the *Local Structure list*, the degree of the central node forming the *Local Structure* is obtained, as well as the surrounding nodes (in which the atom has a bond with).

2. Then, for each neighbour node, it is searched the node in the Local Structure List. If it is found, meaning that there is a Section 4 Local Structure found, Section 4 vector at the corresponding index is increased one unit (*addElement* function). Note that the search in the *Local Structure list* is reduced one position as the *Local Structure* obtained from the *Local Structure List* on Step 1 advances one position, so that the search is shorter on each new iteration.

As described in Step 2, in order to get the index of Section 4 to increase one unit, the following process is followed (*addElement* function).

1. Given a Local Structure connected to x_1 Oxygen atoms and y_1 metal atoms and another Local Structure connected to x_2 Oxygen atoms and y_2 metal atoms, both from the list, the index is calculated as:

$$index = (MAX + 1)^3 \cdot x_2 + (MAX + 1)^2 \cdot y_2 + (MAX + 1) \cdot x_1 + y_1$$

2. If both atomic numbers correspond to the Oxygen atom, it is checked if the index has to be reversed, meaning that:

$$index = (MAX + 1)^3 \cdot x_1 + (MAX + 1)^2 \cdot y_1 + (MAX + 1) \cdot x_2 + y_2$$

   The conditions to be reversed are checking that $x_2 > x_1$, or if they are equal, $y_2 > y_1$, so that the final index formed is the lowest possible. This check is done in the function *hasToReverse*. If the index has to be reversed, the process is done in the function *reverseIndices*, which does the following operations:

$$digit_{last} = index \,\%\, (MAX + 1)$$

$$digit_{SecondLast} = \left(\frac{index}{MAX + 1}\right) \% \, (MAX + 1)$$

$$index = \frac{index}{(MAX+1)^2}$$

$$index = digit_{last} \cdot (MAX+1)^2 + digit_{SecondLast} \cdot (MAX+1)^3 + index$$

Where '%' is the modulus operation and '/' an integer division.

3. Otherwise, if both atomic numbers correspond to a metal atom, the same process as on the previous step is done. In addition, the final index is recalculated as:

$$index = index + (MAX+1)^4 + 1$$

Since it corresponds to the section $M(x_1, y_1) - M(x_2, y_2)$, with $x_2 > x_1$ or $x_2 = x_1$ and $y_2 > y_1$.

4. Otherwise, if the atomic number of the first Local Structure corresponds to the metal one, the final index is recalculated as:

$$index = index + 2 \cdot (MAX+1)^4 + 1$$

Since it corresponds to the section $O(x_2, y_2) - M(x_1, y_1)$.

5. Otherwise, if the atomic number of the second Local Structure corresponds to the metal one, the index is reversed, since the metal atom goes at the right ($O(x_1, y_1) - M(x_2, y_2)$), and final index is recalculated as:

$$index = index + 2 \cdot (MAX+1)^4 + 1$$

To give an example on how indices are reversed, suppose there is the following Section 4 *Local Structure*: $O(6,7) - O(3,1)$, where the first Local Structure is $O(3,1)$ and the second $O(6,7)$. Following the previous steps, firstly, the index would be computed as (supposing *MAX = 10*):

$$index = (10+1)^3 \cdot 6 + (10+1)^2 \cdot 7 + (10+1) \cdot 3 + 1 = 8867$$

Since $x_2 = 6 > x_1 = 3$, the index has to be reversed as:

$$digit_{last} = 8867 \,\%\, 11 = 1$$

$$digit_{SecondLast} = \left(\frac{8867}{11}\right) \%11 = 806 \,\%\, 11 = 3$$

$$index = \frac{index}{(MAX+1)^2} = \frac{8867}{11^2} = 73$$

$$index = 1 \cdot 11^2 + 3 \cdot 11^3 + 73 = 4187$$

Note that the new index 4187 is lower than 8867 and therefore it corresponds to the *Local Structure* reordered as $O(3,1) - O(6,7)$. To check that:

$$index = (10+1)^3 \cdot 3 + (10+1)^2 \cdot 1 + (10+1) \cdot 6 + 7 = 4187$$

3.4. References

[1] Carletti, V., Foggia, P., Saggese, A., & Vento, M. (2017). Challenging the time complexity of exact subgraph isomorphism for huge and dense graphs with VF3. IEEE transactions on pattern analysis and machine intelligence, 40(4), 804-818.

[2] Carletti, V., Foggia, P., Saggese, A., & Vento, M. (2017). Introducing VF3: A new algorithm for subgraph isomorphism. In Graph-Based Representations in Pattern Recognition: l11th IAPR-TC-15 International Workshop, GbRPR 2017, Anacapri, Italy, May 16–18, 2017, Proceedings 11 (pp. 128-139). Springer International Publishing.

[3] Eppstein, D. (2002). Subgraph isomorphism in planar graphs and related problems. In Graph Algorithms and Applications I (pp. 283-309).

[4] Graph Isomorphism (2023, March 6). Wikipedia. Available at https://en.wikipedia.org/wiki/Graph_isomorphism (Accessed: 31/03/2023)

[5] Universidad Autónoma de Madrid (2004-2005). Ordenación en C (qsort). Available at: http://arantxa.ii.uam.es/~swerc/ejemplos/csorting.html (Accessed: 31/03/2023)

CHAPTER IV
NANOFINGERPRINT GENERATION ANALYSIS

This Chapter introduces the used metal-oxide nanoparticles and analyses the performance of the *Structural NanoFingerprint* generation algorithm by presenting the results of some metal-oxide nanoparticles *NanoFingerprints* generation, a correctness checking algorithm and the study of the computational time spent by different sizes of metal-oxide nanoparticles.

At the end, a conclusion introducing the drawbacks of the presented algorithm will lead to the next Chapter.

4.1. Metal-oxide nanoparticles

The aim of this section is to present the origin of the XYZ-format nanoparticles that will serve to analyse the performance of the *NanoFingerprint* generation algorithm, as well as to generate Structural *NanoFingerprints* from them using a regression model, presented on the next Chapter.

The following subsections briefly describe the origin and studies in which different XYZ metal-oxide nanoparticle datasets have been previously used.

4.1.1. Atena dataset

This XYZ nanocompounds have been provided by the research group SSAI (URV) and it is based on the nanoparticle TiO_2 with different sizes. They are the ones that have been used during previous section to exemplify the explanations. Sizes range from 4 Å (TiO2_002.xyz) to 200 Å (TiO2_100.xyz), being multiple of 2. Therefore, by selecting 10 nm (100 Å) as shell thickness, it will be considered the whole nanoparticle, both core and shell. The same thickness will be applied to the rest of nanoparticles, having the same effect of covering the whole nanocompound.

4.1.2. Liu dataset

This dataset has been used for developing a classification-based cytotoxicity nanostructure-activity relationship (*nanoSAR*) [1] and consists of six metal-oxide nanoparticles (Table 4.1).

Nanoparticle	Size (Å)	File	Nanoparticle	Size (Å)	File
Al_2O_3	12	Al2O_012.xyz	TiO_2	18	TiO2_018.xyz
CuO	10	CuO_010.xyz	WO_3	10	WO3_010.xyz
SiO_2	18	SiO2_018.xyz	ZnO	10	ZnO_010.xyz

Table 4.1. Liu dataset nanoparticles.

4.1.3. Gajewicz dataset

This dataset has been used to develop an interpretative nano-QSAR model describing toxicity of 17 nano-metal oxides (Table 4.2) to the HaCaT cell line, which is a common in vitro model for keratinocyte response during toxic dermal exposure [2].

Nanoparticle	Size (Å)	File	Nanoparticle	Size (Å)	File
Al_2O_3	44	Al2O3_044.xyz	Mn_2O_3	30	Mn2O3_030
Bi_2O_3	90	Bi2O3_090.xyz	NiO	20	NiO_020.xyz
CoO	100	CoO_100.xyz	Sb_2O_3	150	Sb2O3_150.xyz
Cr_2O_3	60	Cr2O3_060.xyz	SiO_2	15	SiO2_015.xyz
Fe_2O_3	32	Fe2O3_032.xyz	SnO_2	46	SnO2_046.xyz
In_2O_3	30	In2O3_030.xyz	TiO_2	42	TiO2_042.xyz
La_2O_3	45	La2O3_045.xyz	WO_3	50	WO3_050.xyz
Y_2O_3	38	Y2O3_038.xyz	ZnO	71	ZnO_071.xyz
ZrO_2	46	ZrO2_046.xyz	-	-	-

Table 4.2. Gajewicz dataset nanoparticles.

4.1.4. Pathakoti dataset

On reference [3], a quantitative structure-activity relationship (QSAR) of seventeen metal oxide nanoparticles (Table 4.3), in regard to their photo-induced toxicity to bacteria *Escherichia coli*, has been developed by using quantum chemical methods.

4.1.5. Papa dataset

On the study of reference [4], local regression and classification models were developed for a set of Zinc oxide (ZnO) and Titanium oxide (TiO_2) nanoparticles tested at different

concentrations for their ability to disrupt the lipid membrane in cells. They are presented on Table 4.4.

Nanoparticle	Size (Å)	File	Nanoparticle	Size (Å)	File
Al_2O_3	55	Al2O3_055.xyz	La_2O_3	65	La2O3_065.xyz
Bi_2O_3	144	Bi2O3_144.xyz	NiO	14	NiO_014.xyz
CoO	55	CoO_055.xyz	Sb_2O_3	84	Sb2O3_084.xyz
Cr_2O_3	47	Cr2O3_047.xyz	SiO_2	20	SiO2_020.xyz
CuO	28	CuO_028.xyz	SnO_2	15	SnO2_015.xyz
Fe_2O_3	68	Fe2O3_068.xyz	TiO_2	42	TiO2_042.xyz
In_2O_3	60	In2O3_060.xyz	Zr_2O_3	27	Zr2O3_027.xyz
Y_2O_3	38	Y2O3_038.xyz	ZnO	71	ZnO_071.xyz
ZrO_2	27	ZrO2_027.xyz	-	-	-

Table 4.3. Pathakoti dataset nanoparticles.

Nanoparticle	Size (Å)	File	Nanoparticle	Size (Å)	File
TiO_2	30	TiO2_030.xyz	ZnO	50	ZnO_050.xyz
TiO_2	45	TiO2_045.xyz	ZnO	60	ZnO_060.xyz
TiO_2	125	TiO2_125.xyz	ZnO	70	ZnO_070.xyz

Table 4.4. Papa dataset nanoparticles.

4.1.6. Anantha dataset

Machine learning models use existing experimental data and learn quantitative feature−toxicity relationships to yield predictive models. In the work of reference [5], a principled approach to this problem by formulating a novel feature space based on intrinsic and extrinsic physicochemical properties is adopted, including periodic table properties but exclusive of in vitro characteristics such as cell line, cell type, and assay method.

The used nanoparticles on this work are presented on Table 4.5.

Nanoparticle	Size (Å)	File	Nanoparticle	Size (Å)	File
Al_2O_3	40	Al2O3_040.xyz	TiO_2	125	TiO2_125.xyz
CuO	46	CuO_046.xyz	ZnO	7	ZnO_007.xyz
Fe_2O_3	43	Fe2O3_042.xyz	ZnO	10	ZnO_010.xyz
TiO_2	10	TiO2_010.xyz	ZnO	32	ZnO_032.xyz
TiO_2	20	TiO2_020.xyz	ZnO	35	ZnO_035.xyz
TiO_2	21	TiO2_021.xyz	ZnO	45	ZnO_045.xyz
TiO_2	25	TiO2_025.xyz	ZnO	52	ZnO_052.xyz
TiO_2	30	TiO2_030.xyz	ZnO	60	ZnO_060.xyz
TiO_2	33	TiO2_033.xyz	ZnO	86	ZnO_086.xyz
TiO_2	35	TiO2_035.xyz	ZnO	115	ZnO_115.xyz
TiO_2	40	TiO2_040.xyz	-	-	-

Table 4.5. Anantha dataset nanoparticles.

Note that some on the nanoparticles are used in more than one dataset, such as TiO_2 or ZnO.

4.2. NanoFingerprint correctness checking

As well as implementing an algorithm for the generation of Structural *NanoFingerprints*, it is important to check its correctness to validate it. On the first place, as it was explained on section 3.3.5. Section 4 computation, and as it is shown on Figure 4.1, Section 4 Local Structure bonds from the *NanoFingerprint* are ordered decreasingly. Note, for example, that $M(6,2) - M(6,3)$ comes before $M(6,3) - M(6,4)$, or $M(5,2) - M(6,3)$ comes before $M(5,2) - M(6,4)$.

Additionally, to check the correctness of the generated *NanoFingerprint*, and the relationships between sections, an algorithm was designed, and a MATLAB function was implemented (*check_NanoFingerprint.m*). This function, given a *NanoFingerprint* vector in NV format, checks the number of errors (or incoherences, this is, incoherent counts of the *NanoFingerprint* that contradicts itself).

```
20426-> M[4,1]_M[6,4]: 4
21877-> M[5,2]_M[6,3]: 8
21878-> M[5,2]_M[6,4]: 8
23208-> M[6,2]_M[6,3]: 8
23330-> M[6,3]_M[6,4]: 8
23451-> M[6,4]_M[6,4]: 8
29718-> O[0,1]_M[4,1]: 8
29730-> O[0,1]_M[5,2]: 8
29741-> O[0,1]_M[6,2]: 8
29742-> O[0,1]_M[6,3]: 8
29839-> O[0,2]_M[4,1]: 4
29851-> O[0,2]_M[5,2]: 16
29862-> O[0,2]_M[6,2]: 8
29863-> O[0,2]_M[6,3]: 16
29864-> O[0,2]_M[6,4]: 4
29960-> O[0,3]_M[4,1]: 4
29972-> O[0,3]_M[5,2]: 16
29983-> O[0,3]_M[6,2]: 8
29984-> O[0,3]_M[6,3]: 24
29985-> O[0,3]_M[6,4]: 50
```

Figure 4.1. *NanoFingerprint* Section 4 from TiO_2 of size 1.4 nm (TiO2_007).

The following subsections describe the *NanoFingerprint* correctness checking algorithm. The first step, previously to check each section, is extracting from the *NanoFingerprint* three integer vectors, corresponding to Section 2, Section 3, and Section 4. At the end, an example of correctness checking of a TiO_2 generated *NanoFingerprint* is provided.

4.2.1. Checking Section 2

On Section 2 of the *Structural NanoFingerprint* generated it is checked that the sum of $O(x)$ counts is less or equal to the total number of Oxygen atoms of the nanoparticle, present at the fifth position of the *NanoFingerprint*, as well as the sum of $M(x)$ counts is less or equal to the total number of metal atoms of the nanoparticle, present at the sixth position of the *NanoFingerprint*. For each section, it is computed an error if the result is non equal, giving a maximum of *error = 2* on this section (one for each atom of different atomic number).

The reason why it cannot be equal is because the MAX parameter fixes the maximum number of bonds that an atom can have in order to be counted. If it is the case that the number of real bonds is higher than MAX, there will be atoms that will not figure in the count and thus, the sum of Section 2 atoms (Oxygen and/or metal, and vice versa) will be lower than the total number of Oxygen and/or metal atoms, and vice versa.

Figure 4.2 is depicting the 3D-structure of the nanoparticle CuO of size 10 Å. For example, if it was picked as MAX 10 bonds, the atom circled in green would not be counted, since it has more than 10 bonds.

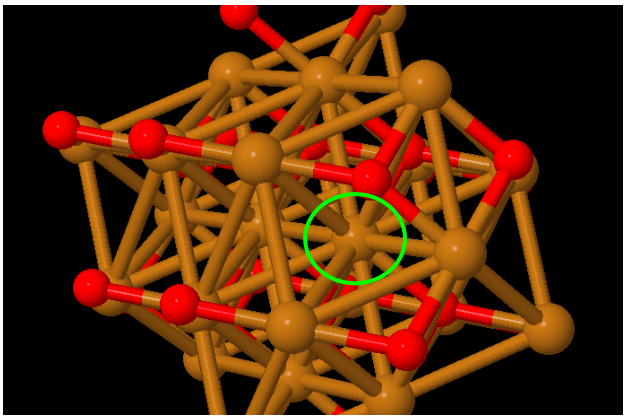


Figure 4.2. 3D-structure of CuO of size 10 Å (CuO_010).

4.2.2. Checking Section 3

Section 3 contains the Local Structures $O(x,y)$ and $M(x,y)$. For each *Local Structure* whose count is different than zero, it is checked:

- That the number of bonds to other atoms, this is, $x+y$, is equal or lower than MAX, since the maximum number of bonds it is described by this parameter. Otherwise, it adds one unit to the error counter.

- That the number of bonds to other atoms, this is, $x+y$, is different from 0 (it do not exist neither $O(0,0)$ nor $M(0,0)$, since it would mean having an isolate atom). Otherwise, it adds one unit to the error counter, one for each central atom element.

- That the sum of appearances of the Local Structures whose number of bonds to other atoms is equal, is equal to Section 2 *Local Structure* count whose central atom and number of bonds are the same. For example, having two Local Structures such as $O(2,3)$ and $O(4,1)$, whose number of appearances are 3 and 4, respectively, would be compared to the Local Structure of Section 2 $O(5)$. Note that $O(2,3)$ and $O(4,1)$ have 5 bonds to other atoms both, so their number of appearances would be summed, resulting in 7 (suppose there are no other $O(x,y)$ with 5 bonds). Then, in case that $O(5)$ did not have 7 appearances, an error would be computed.

To perform the second error checking, the position of the Local Structure at the vector is converted to x (connections to Oxygen atoms) and y (connections to metal atoms) as:

$$y = position \ \% \ (MAX+1)$$

$$x = \left(\frac{index}{MAX+1}\right) \%(MAX+1)$$

Where '/' it is an integer division. If the Local Structure central atom is a metal, previously, the following operation is done over the position:

$$position = position - (MAX + 1)^2$$

Since there are $(MAX + 1)^2$ combinations of $O(x, y)$ and $M(x, y)$.

Then, adding $z = x + y$, $O_{aux}(z)$ and $M_{aux}(z + MAX)$ are built in order to be compared, once all Section 3 has been elapsed, can be easily compared to $O(z)$ and $M(z + MAX)$. It is added MAX because on Section 2, metal atoms start at position $MAX + 1$.

4.2.3. Checking Section 4

On Section 4 it is checked that in all Local Structures, which can be $O(x_1, y_1) - O(x_2, y_2)$, $M(x_1, y_1) - M(x_2, y_2)$ and $O(x_1, y_1) - M(x_2, y_2)$, the Local Structures that form them, $O(x, y)$ and/or $M(x, y)$, appear on Section 3 with a number of appearances different than zero.

From the position of a Section 4 Local Structure, the indices x_1, y_1, x_2 and y_2 are calculated as:

$$y_2 = position \,\%\, (MAX + 1)$$

$$y_1 = \frac{position}{(MAX + 1)^2} \,\%\, (MAX + 1)$$

$$y_1 = \frac{position}{(MAX + 1)^2} \,\%\, (MAX + 1)$$

$$x_1 = \frac{position}{(MAX + 1)^3} \,\%\, (MAX + 1)$$

In case of being the Local Structure $M(x_1, y_1) - M(x_2, y_2)$, the following operation is previously done:

$$position = position - (MAX + 1)^4 - 1$$

And, if it is de case of $O(x_1, y_1) - M(x_2, y_2)$:

$$position = position - 2 \cdot (MAX + 1)^4 - 1$$

Then, from x_1 and y_1 a Section 3 Local Structure position can be found, as well as for x_2 and y_2:

$$position_{Section3_1} = (MAX + 1) \cdot x_1 + y_1 + 1$$

$$position_{Section3_2} = (MAX + 1) \cdot x_2 + y_2 + 1$$

In case of the Local Structure $M(x_1, y_1) - M(x_2, y_2)$, both positions are modified as:

$$position_{Section3_1} = position_{Section3_1} + (MAX + 1)^2$$

$$position_{Section3_2} = position_{Section3_2} + (MAX + 1)^2$$

And, if it is de case of $O(x_1, y_1) - M(x_2, y_2)$, just the second one is modified in the same way as previously.

Finally, it is checked that these positions are different than zero on Section 3. Otherwise, an error is computed.

It has to be commented that the first idea on this checking was to check that the count appearances of the Section 4 *Local NanoFingerprint* was equal or lower than the addition of the count appearances of two *Local NanoFingerprints* that form it and are part of Section 3. However, note that it is possible to have more count appearances on the Section 4 *Local NanoFingerprint* than the addition of the two forming ones. For instance, in the nanoparticle TiO_2 of size 14 nm (*TiO2_007.xyz*) there are 50 appearances of $O(0,3) - M(6,4)$, but 34 of $O(0,3)$ and 9 of $M(6,4)$, which adds up to 43 (lower than 50). This is totally possible, and the *NanoFingerprint* generation is successful.

4.2.4. NanoFingerprint correctness checking example

On Table 4.6 it is shown the output of the *check_NanoFingerprint* for a correct and incorrect (manually edited on purpose) *NanoFingerprint*, from the nanoparticle TiO_2 of size 0.6 nm (TiO2_003).

NanoFingerprint (correct)	**NanoFingerprint (incorrect)**
Shell: 100 MaxBonds: 10 Size: 5.988513 Atomic: 22 O: 6 M: 5 8-> O[2]: 4 9-> O[3]: 2 19-> M[3]: 4 26-> M[10]: 1 29-> O[0,2]: 4 30-> O[0,3]: 2 171-> M[2,1]: 4 218-> M[6,4]: 1 17764-> M[2,1]_M[6,4]: 4 29817-> O[0,2]_M[2,1]: 4 29864-> O[0,2]_M[6,4]: 4 29938-> O[0,3]_M[2,1]: 4 29985-> O[0,3]_M[6,4]: 2	Shell: 100 MaxBonds: 10 Size: 5.988513 Atomic: 22 O: 7 M: 8 8-> O[2]: 4 9-> O[3]: 2 19-> M[3]: 4 26-> M[10]: 1 29-> O[0,2]: 4 30-> O[0,3]: 2 171-> M[2,1]: 4 218-> M[6,4]: 10 17764-> M[2,1]_M[6,4]: 4 29817-> O[0,2]_M[2,1]: 4 29864-> O[0,2]_M[6,4]: 4 29938-> O[0,3]_M[2,1]: 4 29985-> O[0,3]_M[6,4]: 2
***check_NanoFingerprint* output**	***check_NanoFingerprint* output**
0	3

Table 4.6. Correct (left) and incorrect (right) generated VNZ *NanoFingerprint* of TiO_2 of size 6 nm, and its error checking count.

Note that for the incorrect generated *NanoFingerprint*, three errors have been accounted. The changed values have been O, M and $M(6,4)$, the reasons why these three errors have been computed.

4.3. NanoFingerprint computational time study

In order to see how computationally expensive, the designed algorithm is, an analysis of the computation time by parts of the program was done, as well as to analyse which parts of the code spend more time and try to reduce them as much as possible.

The times that are outputted by the program are:

- Computation time after being executed the function *graphToThickness* (in seconds).

- Computation time after being executed the function *xyzToGraph* (in seconds).
- Computation time after completing Section 1 (in seconds).
- Computation time after completing Section 2 (in seconds).
- Computation time after completing Section 3 (in seconds).
- Computation time after completing Section 4 (in seconds).
- Computation time taken by the whole program, in seconds and, in addition, in hours-minutes-seconds-milliseconds format.

Note that the computation time is measured from the beginning of the program until finishing each part.

On the following subsections the *NanoFingerprint* generation computational time spent by different nanoparticles is detailed. Two studies are done: one with the nanoparticles belonging to the Atena dataset and the one with the other datasets presented before.

4.3.1. Atena dataset computational time study

The first study is dedicated to the *Atena dataset* nanoparticles, which includes Titanium oxide nanoparticles (TiO_2) with sizes ranging from 0.4 nm to 20 nm and consists in two analysis: the first one studying the computational time spent by different parts of the *NanoFingerprint* generation algorithm, and the second one studying the computational time spent by most (or all) of the nanoparticles from this dataset. These first two studies *NanoFingerprints* generation were run on a MacBook Pro with the dual-core processor 2,9 GHz Intel Core i5.

Additionally, it is provided a third study run on a different and less power computer, an HP Compaq Elite 8300 SFF with processor Intel(R) Core(TM) i5-3470 CPU @ 3.20GHz.

On Table 4.7 it is shown the computation time of the different parts of the *NanoFingerprint* generation from TiO_2 of sizes 8.4 nm, 12.8 nm and 16.4 nm with $MAX = 10$.

In addition, on Figure 4.3 it is shown the output of the *NanoFingerprint* generation program specifying the computation times of each part for TiO_2 of size 16.4 nm.

It has to be commented that computational times are not exactly the same on different executions and can vary in the order of tens or even a unit of hundreds.

***NanoFingerprint* part**	**TiO2_042.xyz**	**TiO2_064.xyz**	**TiO2_082.xyz**
graphToThickness	0.079825	0.260237	0.312597
xyzToGraph	41.271719	510.148529	2136.094207
Section 1	41.272298	510.156883	2136.095631
Section 2 and 3	41.368136	510.600805	2136.531189
Section 4	52.848553	665.032996	2755.697835
Total time (s)	52.859205	665.045295	2755.707512
Total time (h, m, s, ms)	0h 0m 52s 859ms	0h 11m 5s 45ms	0h 45m 55s 707ms

Table 4.7. Computation time (in seconds, and the last row in hours, minutes, seconds and milliseconds) of the different *NanoFingerprint* parts for different sizes of TiO_2 (8.4 nm, 12.8 nm and 16.4 nm).

```
TiO2_082.xyz
Start
Finished with graphToThickness: 0.312597
Finished with xyzToGraph: 2136.094207
Finished with Section 1: 2136.095631
Finished with Section 2 and 3: 2136.531189
Finished with Section 4: 2755.697835
Finished writing NanoFingerprint: 2755.707512
Nanofingerprint generated in: 0h 45m 55s 707ms
Nanofingerprint VNZ file path: Generated_Nanofingerprints/TiO2_083_nanofingerprint_10_VNZ.txt
Nanofingerprint VNZ file path: Generated_Nanofingerprints/TiO2_083_nanofingerprint_10_NV.txt
```

Figure 4.3. *NanoFingerprint* generation output of TiO_2 nanoparticle of size 82 nm.

This first analysis took to some conclusions:

- The function *xyzToGraph* was spending the vast majority of the overall computation time (approximately a 75%), and Section 4 was spending some of the time, as well, but few than the first (approximately a 20%). The rest of parts were practically instantaneous. Therefore, the *xyzToGraph* function used in previous implementations (*Subgraph search*, *Local Structure search*) was redesigned according to the explanations given on 3.3. Final approach. These changes had an impact on reducing the overall computation time in an order of, approximately, 40-50%, depending also on the size of the graph (more effect on nanoparticles of more size), as depicted on Figure 4.4. Table 4.7 shows the computational times having modified the function *xyzToGraph.*

- *Section 4* was left as described on 3.3. Final approach, since no other improvement was thought to be possible.

- The first implementation of *Section 2 and 3* (second variant), was thought at the beginning to be computationally expensive, since required the iteration of the whole graph multiple times, so the first variant was implemented. However, both spend a fraction of tenths of second to compute.

The second analysis studies the overall *NanoFingerprint* generation computational time spent by different sizes of the nanoparticle TiO_2. Firstly, on Figure 4.4 it is shown the time spent by the generation of *NanoFingerprints* (NV and VNZ versions) of ascending sizes of this *nanoparticle*, from 0.4 nm to 16.6 nm and *MAX* = 10. It is plotting the generation time spent (in seconds) over the number of atoms of the nanocompound. In fact, each point corresponds to a different size, but rather than indicating the size, it is indicated the number of atoms. Therefore, the more atoms the nanoparticle has, the higher time it requires the *NanoFingerprint* to generate. Note that the function is quite exponential, but not with a base of very high value.

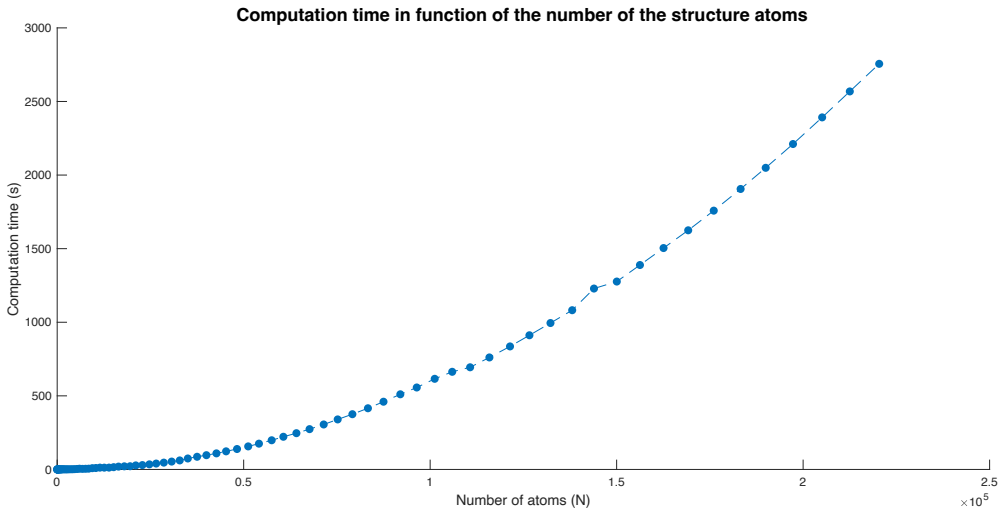


Figure 4.4. Computational time (in seconds) spent by TiO_2 *NanoFingerprint* generation of sizes from 0.4 nm to 16.6 nm.

To compare the computational time spent by the implementation of the *NanoFingerprint* generation with the previous *xyzToGraph* function and the actual one, on Figure 4.5 both generations are compared. Note that, despite not having much influence on lower sizes, it reduces approximately between a 40-50% the generation time from the old to the upgraded and actual version.

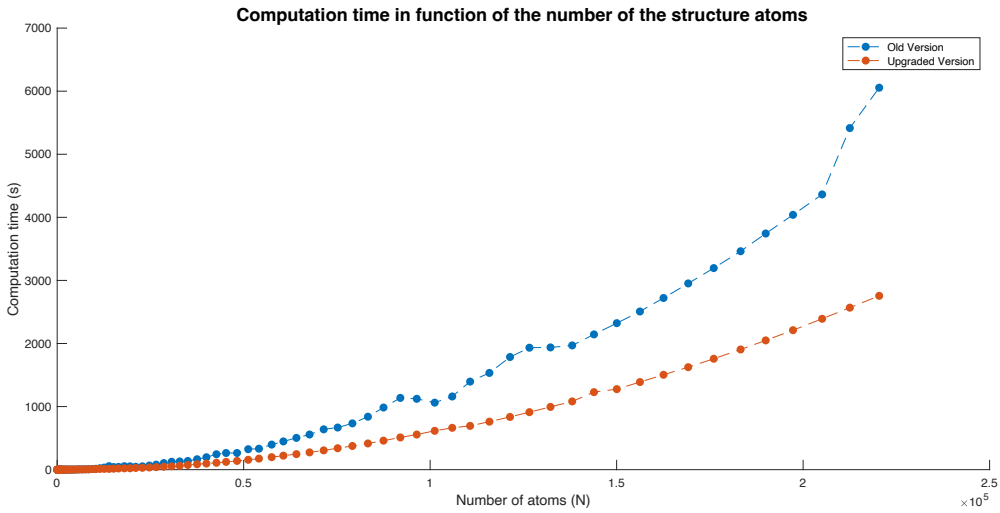


Figure 4.5. Computational time (in seconds) spent comparison (old vs. upgraded *xyzToGraph*) by TiO_2 *NanoFingerprint* generations of sizes from 0.4 nm to 16.6 nm.

A third test to see the computation time on a different computer was done. On this case, on an HP computer with a less powerful processor than the previous one. On Figure 4.6 it is depicted the computation time in seconds over sizes from 0.4 nm to 20 nm of the nanoparticle TiO_2 (full Atena dataset), with the upgraded *NanoFingerprint* generation version, in comparison to the computation times obtained with the TiO_2 nanoparticles of sizes ranging from 0.4 nm to 16.6 nm.

Note that the time spent by the HP computer is much higher compared to even the old *xyzToGraph* function run on the MacBook Pro computer. For instance, the computation time for TiO_2 of size 16.6 nm (TiO2_083), it is of 6954.905 s for the upgraded version with the HP computer, 6054.98 s for the old version with the MacBook Pro computer and 2755.708 s for the upgraded version with the MacBook Pro computer.

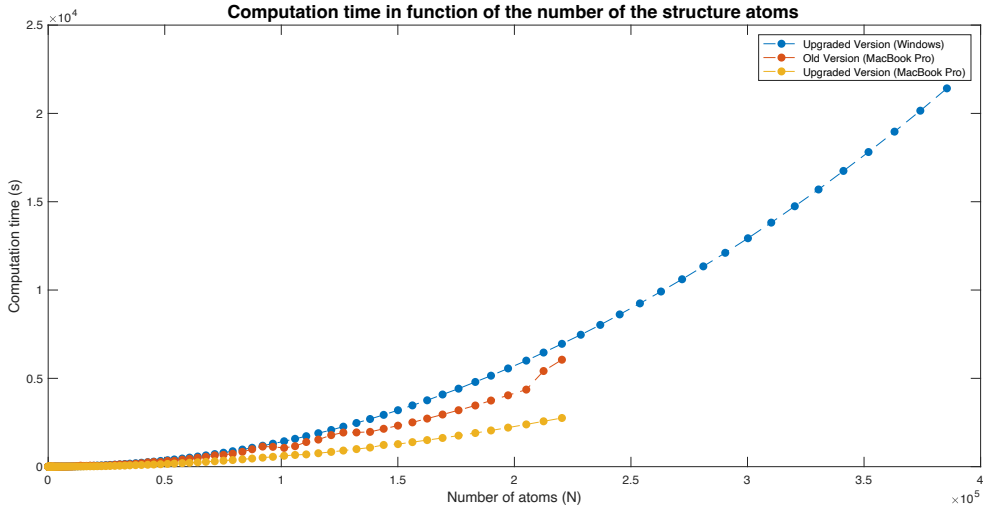


Figure 4.6. Computational time (in seconds) spent comparison by TiO_2 *NanoFingerprint* generations with an HP Compaq Elite 8300 SFF with processor Intel(R) Core (TM) i5-3470 CPU @ 3.20GHz (sizes from 0.4 nm to 20 nm) and a MacBook Pro with the dual-core processor 2,9 GHz Intel Core i5 (sizes from 0.4 nm to 16.6 nm).

4.3.2. Other datasets computational time study

The second study focuses on analysing the whole computational time spent by the *NanoFingerprint* generation for the rest of datasets.

The generations have been run on a MacBook Pro with the dual-core processor 2,9 GHz Intel Core i5.

Firstly, on Figure 4.6 it is shown a representation of the computation time spent by the generation of the *NanoFingerprints* from the other datasets over the number of atoms (N), not discretizing between the different metal-oxide nanoparticle and $MAX = 10$.

Figure 4.6 includes two representations: on the top, considering all the nanoparticles while, on the bottom, just nanoparticles with up to 6000 atoms. Note that the growth is exponential, but with not a high exponent.

Secondly, the computational times have been detailed for each nanoparticle and sorted by datasets, in different subsections.

Thirdly, for the nanoparticle TiO_2 of size 12.5 nm (tio2_125.xyz, from Papa and Anantha datasets), it has been computed the *NanoFingerprint* generation time with the old version changing the MAX input parameter, from 1 to 10 (Figure 4.7). This simulation helped to realize that the computational time, rather than depending on the MAX parameter, depends on the size

of the particle (number of atoms), since the computation time shown for different MAX values is quite constant.

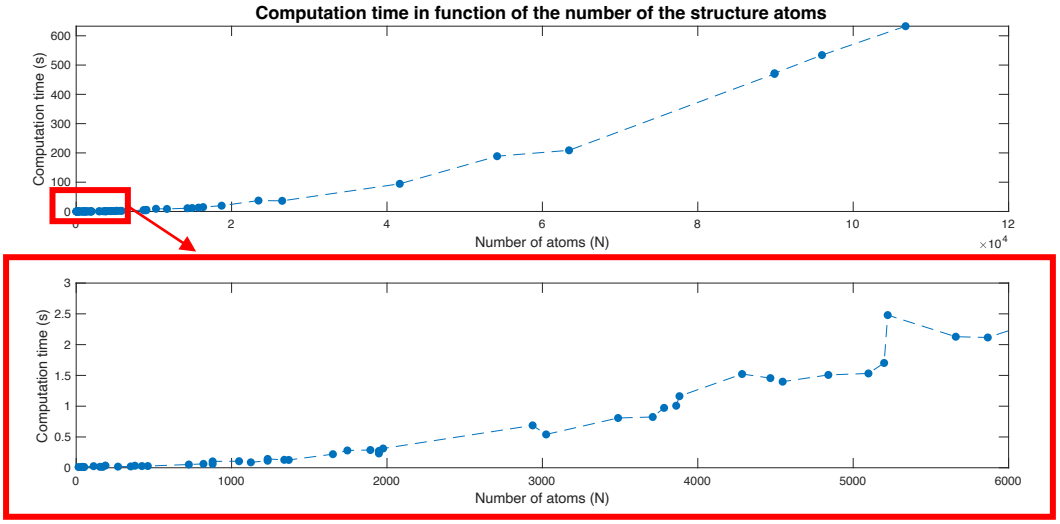


Figure 4.6. *NanoFingerprint* generation computational time (in seconds) spent by the nanoparticles from the Liu, Gajewicz, Pathakoti, Papa and Anantha. On the top, considering all the nanoparticles, while on the bottom, just up to 6000 atom nanoparticles.

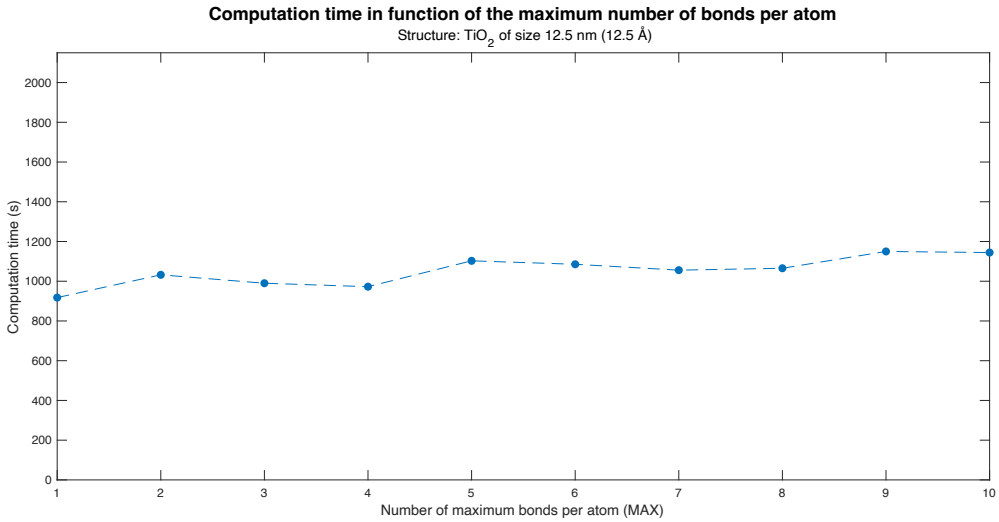


Figure 4.7. *NanoFingerprint* generation computational time (in seconds) spent by the nanoparticle TiO_2 of size 12.5 nm for different values of MAX (from 1 to 10).

On the following subsections the nanoparticles of the other datasets are presented. Note that some of them are repeated; thus, just one *NanoFingerprint* has been generated in those cases.

For all datasets it has been checked that there has not been any error in the *NanoFingerprint* generation, as explained in 4.2. NanoFingerprint correctness checking. Additionally, it has to be commented that for any dataset it has been found any $O(x,y) - O(x,y)$ *Local Structure*.

a) Liu dataset

On Table 4.8 the *NanoFingerprint* generation computational times (in seconds) spent by the different nanoparticles of the Liu dataset are depicted. Note that same sizes for different metal oxide nanoparticles do not imply having the same number of atoms and, therefore, the computation times change.

Nanoparticle	File	Size (Å)	N	Time (s)
Al_2O_3	Al2O3_012.xyz	12	112	0.025718
CuO	CuO_010.xyz	10	50	0.012048
SiO_2	SiO2_018.xyz	18	378	0.033073
TiO_2	TiO2_018.xyz	18	269	0.018322
WO_3	WO3_010.xyz	10	40	0.014812
ZnO	ZnO_010.xyz	10	33	0.011522

Table 4.8. NanoFingerprint generation computational time (in seconds) spent by the nanoparticles of the Liu dataset. The N refers to the number of nanoparticle nodes.

b) Gajewicz dataset

On Table 4.9 the *NanoFingerprint* generation computational times (in seconds) spent by the different nanoparticles of the Gajewicz dataset are depicted.

Nanoparticle	File	Size (Å)	N	Time (s)
Al_2O_3	Al2O3_044.xyz	44	5222	2.479780
Bi_2O_3	Bi2O3_090.xyz	90	23461	37.266645
CoO	CoO_100.xyz	100	54171	188.759883
Cr_2O_3	Cr2O3_060.xyz	60	11682	8.624138
Fe_2O_3	Fe2O3_032.xyz	32	1652	0.220355
In_2O_3	In2O3_030.xyz	30	1048	0.106289
La_2O_3	La2O3_045.xyz	45	3024	0.541028
Y_2O_3	Y2O3_038.xyz	38	1948	0.273012
ZrO_2	ZrO2_046.xyz	46	1948	0.273012

Mn_2O_3	Mn2O3_030	30	1339	0.128907
NiO	NiO_020.xyz	20	461	0.027632
Sb_2O_3	Sb2O3_150.xyz	150	106740	632.804488
SiO_2	SiO2_015.xyz	15	188	0.035748
SnO_2	SnO2_046.xyz	46	4285	1.522670
TiO_2	TiO2_042.xyz	42	3483	0.972705
WO_3	WO3_050.xyz	50	4840	1.507895
ZnO	ZnO_071.xyz	71	15729	13.168779

Table 4.9. *NanoFingerprint* generation computational time (in seconds) spent by the nanoparticles of the Gajewicz dataset. The N refers to the number of nanoparticle nodes.

c) Pathakoti dataset

On Table 4.10 the *NanoFingerprint* generation computational times (in seconds) spent by the different nanoparticles of the Pathakoti dataset are depicted.

Nanoparticle	File	Size (Å)	N	Time
Al_2O_3	Al2O3_055.xyz	55	10270	9.222741
Bi_2O_3	Bi2O3_144.xyz	144	95983	534.162329
CoO	CoO_055.xyz	55	9045	5.460743
Cr_2O_3	Cr2O3_047.xyz	47	5660	2.128445
CuO	CuO_028.xyz	28	1124	0.088104
Fe_2O_3	Fe2O3_068.xyz	68	16334	14.958653
In_2O_3	In2O3_060.xyz	60	8664	4.406766
Y_2O_3	Y2O3_038.xyz	38	1948	0.273012
ZrO_2	ZrO2_027.xyz	27	878	0.103482
La_2O_3	La2O3_065.xyz	65	8793	4.414079
NiO	NiO_014.xyz	14	171	0.014278
Sb_2O_3	Sb2O3_084.xyz	84	18722	19.777542
SiO_2	SiO2_020.xyz	20	819	0.062818

SnO_2	SnO2_015.xyz	15	153	0.015265
TiO_2	TiO2_042.xyz	42	3483	0.972705
ZnO	ZnO_071.xyz	71	15729	13.168779

Table 4.10. *NanoFingerprint* generation computational time (in seconds) spent by the nanoparticles of the Gajewicz dataset. The N refers to the number of nanoparticle nodes.

d) Papa dataset

On Table 4.11 the *NanoFingerprint* generation computational times (in seconds) spent by the different nanoparticles of the Papa dataset are depicted.

Nanoparticle	Size (Å)	File	N	Time
TiO_2	30	TiO2_030.xyz	1232	0.114110
TiO_2	45	TiO2_045.xyz	4467	1.456116
TiO_2	125	TiO2_125.xyz	89875	469.877111
ZnO	50	ZnO_050.xyz	5199	1.702239
ZnO	60	ZnO_060.xyz	8985	4.490942
ZnO	70	ZnO_070.xyz	14321	11.150260

Table 4.11. *NanoFingerprint* generation computational time (in seconds) spent by the nanoparticles of the Papa dataset. The N refers to the number of nanoparticle nodes.

e) Anantha dataset

On Table 4.12 the *NanoFingerprint* generation computational times (in seconds) spent by the different nanoparticles of the Anantha dataset are depicted.

Nanoparticle	Size (Å)	File	N	Time
Al_2O_3	40	Al2O3_040.xyz	3882	1.162499
CuO	46	CuO_046.xyz	5098	1.531652
Fe_2O_3	43	Fe2O3_042.xyz	3710	0.824693
TiO_2	10	TiO2_010.xyz	51	0.011577
TiO_2	20	TiO2_020.xyz	351	0.021320
TiO_2	21	TiO2_021.xyz	423	0.027311

TiO_2	25	TiO2_025.xyz	725	0.052596
TiO_2	30	TiO2_030.xyz	1232	0.114110
TiO_2	33	TiO2_033.xyz	1745	0.280845
TiO_2	35	TiO2_035.xyz	1975	0.312510
TiO_2	40	TiO2_040.xyz	2937	0.688119
TiO_2	125	TiO2_125.xyz	89875	469.877111
ZnO	7	ZnO_007.xyz	15	0.011145
ZnO	32	ZnO_032.xyz	1368	0.126974
ZnO	35	ZnO_035.xyz	1893	0.288230
ZnO	45	ZnO_045.xyz	3861	1.007512
ZnO	52	ZnO_052.xyz	5866	2.116050
ZnO	60	ZnO_060.xyz	8985	4.490942
ZnO	86	ZnO_086.xyz	26509	36.327174
ZnO	100	ZnO_100.xyz	41641	94.475862
ZnO	115	ZnO_115.xyz	63429	208.899335

Table 4.12. *NanoFingerprint* generation computational time (in seconds) spent by the nanoparticles of the Anantha dataset. The N refers to the number of nanoparticle nodes.

4.4. Drawbacks of the NanoFingerprint generation proposed algorithm

As it has been explained, the proposed algorithm to correctly generate Structural *NanoFingerprints* needs three input parameters: a nanoparticle structure in XYZ format, the thickness of the shell and the maximum number of bonds (MAX). The last two ones must be thoroughly chosen, but while the MAX parameter can be fixed to a determined value (for instance, 10), thickness of the shell can be chosen using a specific software.

However, generating large XYZ structures is computationally very expensive, adding to the expense of generating a *NanoFingerprint*. For this reason, it is needed to find a faster way of generating *NanoFingerprints*, either without having the XYZ structures and generating *NanoFingerprints* given the nanoparticles and its size, or having the XYZ structures and just considering some fast-to-obtain parameters such as the total number of atoms or the number of Oxygen and metal atoms, for instance.

Therefore, the proposed algorithm is not valid for this problem (apart from being hard to improve in terms of computational time expense). The solution is the design of new models in which given the already known and generated *NanoFingerprints* can effectively generate new *NanoFingerprints* in a very fast way. The easiest and most suitable way is using a regression. However, it has to be assumed that an error rate is implied in this process, but having an approximation on the characterisation of the local structure through the atoms' connectivity is sufficient for *Structural NanoFingerprint* applications.

On the next Chapter the Multiple Regression models from which *NanoFingerprints* can be generated are proposed, and more in detail the reasons for its suitability to the proposed problem are explained.

4.5. References

[1] Liu, R., Rallo, R., George, S., Ji, Z., Nair, S., Nel, A. E., & Cohen, Y. (2011). Classification NanoSAR development for cytotoxicity of metal oxide nanoparticles. Small, 7(8), 1118-1126.

[2] Agnieszka Gajewicz, Nicole Schaeublin, Bakhtiyor Rasulev, Saber Hussain, Danuta Leszczynska, Tomasz Puzyn & Jerzy Leszczynski (2015) Towards understanding mechanisms governing cytotoxicity of metal oxides nanoparticles: Hints from nano-QSAR studies, Nanotoxicology, 9:3, 313-325, DOI: 10.3109/17435390.2014.930195

[3] Pathakoti, K., Huang, M. J., Watts, J. D., He, X., & Hwang, H. M. (2014). Using experimental data of Escherichia coli to develop a QSAR model for predicting the photo-induced cytotoxicity of metal oxide nanoparticles. Journal of Photochemistry and Photobiology B: Biology, 130, 234-240.

[4] E. Papa, J.P. Doucet & A. Doucet-Panaye (2015): Linear and non-linear modelling of the cytotoxicity of TiO2 and ZnO nanoparticles by empirical descriptors, SAR and QSAR in Environmental Research, DOI: 10.1080/1062936X.2015.1080186

[5] Subramanian, N. A., & Palaniappan, A. (2021). NanoTox: development of a parsimonious in silico model for toxicity assessment of metal-oxide nanoparticles using physicochemical features. ACS omega, 6(17), 11729-11739.

CHAPTER V
ARTIFICIAL GENERATION OF NANOFINGERPRINTS

This Chapter is focused on the artificial generation of *Structural NanoFingerprints*. On the first place, it is presented the motivation of artificially generating them and why a regression model is suitable. Then, the model is presented, including the explanation of the pre-processing and postprocessing phases, as well as the techniques and error metrics applied. Next, the four models generated are presented and studied, with results, and finally, some conclusions are taken.

5.1. Motivation

It is aimed to artificially generate the *NanoFingerprint* of specific nanocompound and size. As commented at the end of the previous Chapter, the generation of the three-dimensional structure is needed to deduce the *NanoFingerprint*, which is a computationally expensive task for medium to large nanocompounds.

Therefore, on this Chapter it is proposed a machine learning model capable of artificially predicting new *NanoFingerprints* model. In order to choose the machine learning model to use, it has been observed the tendency of different particular Local Structures over different sizes of TiO_2. As it can be seen on Figure 5.1, there is the representation of the apparition number of the Local Structures $O(3)$ (Section 2), $M(10)$ (Section 2), $O(0,3)$ (Section 3) and $M(6,4)$ (Section 4) throughout sizes ranging from 0.4 nm to 16.5 nm. It appears to be a clear tendency in the *NanoFingerprint*, exponential. Therefore, the suggested model is a regression since it is the most suitable approach.

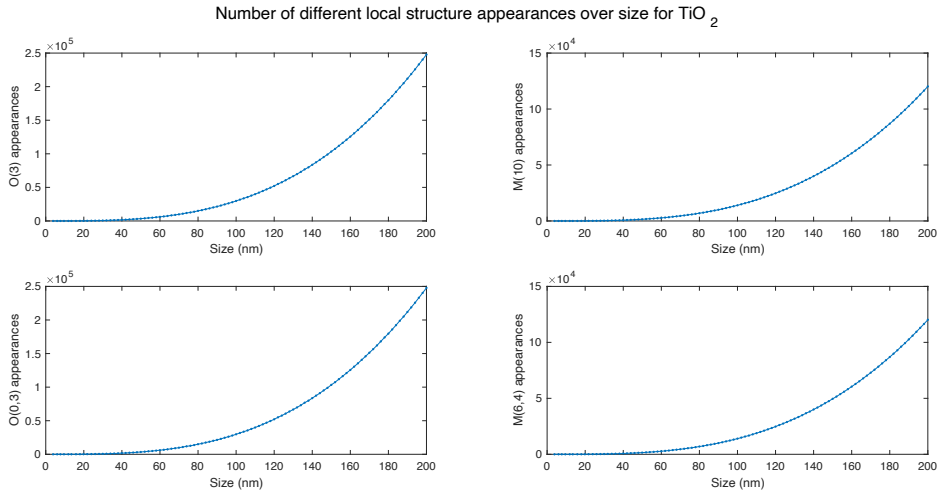


Figure 5.1. Representation of the apparition number of the Local Structures $O(3)$, $M(10)$, $O(0,3)$ and $M(6,4)$ over sizes ranging from 0.4 nm to 20 nm for TiO_2.

To artificially generate *NanoFingerprints*, two approaches are proposed on this Chapter:

1. Assuming that the XYZ structures are not given, the generation of the entire *NanoFingerprints* given the size and the metal atomic number. This approach has the advantage of not needing the three-dimensional structures, computationally expensive to obtain. It is divided in two studies, depending on the datasets used:

    1.1. Using the *Atena* dataset. On this case, all the nanoparticles are the same (TiO_2), so the metal atomic number is irrelevant.

    1.2. Using the other datasets. On this case, since the nanoparticles are not the same, the metal atomic number input is key.

2. Assuming that the XYZ structures are given, the generation of the entire *NanoFingerprints* given the size, metal atomic number, number of Oxygen atoms and number of metal atoms of the nanoparticle. These last two input parameters can be easily obtained in a fast computation time from the XYZ structure. This approach is divided in two studies, depending on the datasets as well:

    2.1. Using the *Atena* dataset. On this case, all the nanoparticles are the same (TiO_2), so the metal atomic number is irrelevant.

    2.2. Using the other datasets. On this case, since the nanoparticles are not the same, the metal atomic number input is key.

The *thickness* and *MAX* parameters from the *NanoFingerprint* Section 1 will be considered as constants, set to 100 nm (considering the whole nanoparticle, both core and shell) and 10, respectively.

The first approach implies using a Non-Linear Regression model, since it is needed a quadratic element to deduce the exponential form. On the other hand, as it is seen on Figure 5.2, the number of Oxygen and metal atoms grows exponentially with size (Athena dataset). Therefore, a Linear Regression is a good model for the second approach. Additionally, a Non-Linear Regression model will be modelled for the second approach.

Additionally, the difference between a Regression and a Multiple Regression has to be explained. On the case 1.1, just the size is used as input (one variable), so it is a simple regression. On the other cases, more than one input is used, and it receives the name of Multiple Regression.

The regression models from the different studies and approaches will be modelled with already generated *NanoFingerprints*, in order to obtain the regression coefficients (learn the tendencies) to use in new generations.

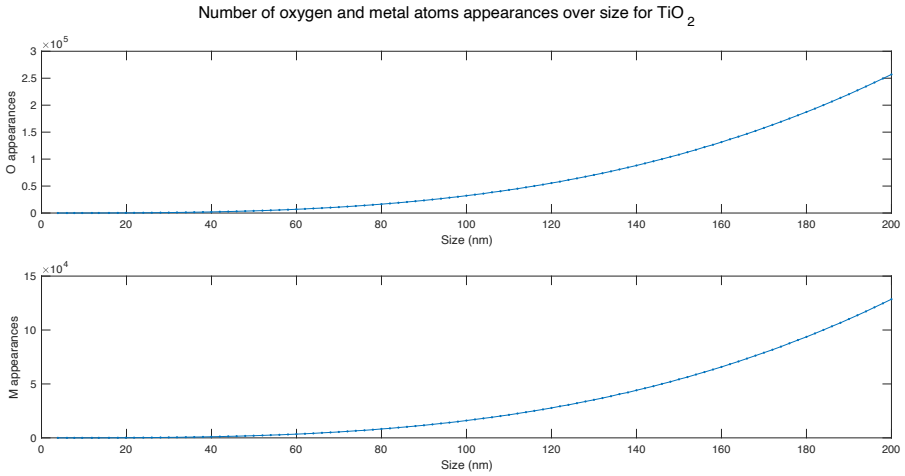


Figure 5.2. Representation of the apparition number of the Oxygen and metal atoms over sizes ranging from 0.4 nm to 20 nm for TiO_2.

However, not all the elements of the *NanoFingerprint* grow exponentially in function of size, as it is shown on Figure 5.3. Some of them follow an exponential grow, but not a perfect curve (points oscillate around it), while some other do not follow any tendency. Note these non-exponential tendencies are mainly for *Local Structures* in which its central atom is a metal connected to some Oxygens and another metal atom. Some of these results will be shown, and based on them, it has to be assumed that there will be a part of error on the model generation (in those cases in which there is not an exponential tendency, more error), since it will be not very suitable on these cases.

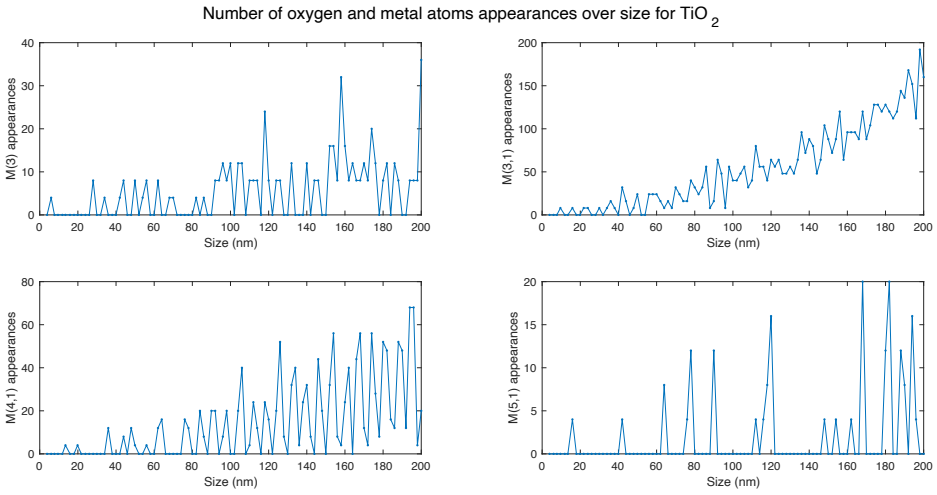


Figure 5.3. Representation of the apparition number of the Local Structures $M(3)$, $M(3,1)$, $M(4,1)$ and $M(5,1)$ over sizes ranging from 0.4 nm to 20 nm for TiO_2.

The two previous figures were using the Atena dataset (TiO_2 nanoparticles). However, for the other datasets, it will be more complicated.

On the first place, because of the low quantity of each different nanoparticle, which makes having a non-sufficient sample of *NanoFingerprints* to train and fit the model. The number of each different nanoparticle files is described on Table 5.1.

Nanoparticle	Quantity of files	Nanoparticle	Quantity of files
Al_2O_3	4	NiO	2
Bi_2O_3	2	Sb_2O_3	2
CoO	2	SiO_2	3
Cr_2O_3	2	SnO_2	2
CuO	3	TiO_2	12
Fe_2O_3	3	WO_3	2
In_2O_3	2	Y_2O_3	1
La_2O_3	2	ZnO	13
Mn_2O_3	1	ZrO_2	2

Table 5.1. Quantity of different nanoparticles for each nanoparticle.

Note that apart from TiO_2 and ZnO, the number of files for the nanoparticles is not sufficient. Therefore, having these two with much more quantity, makes the distribution unbalanced.

On the second place, because of the growth of the number of Oxygen atoms and metal atoms, as well as for the Local Structures, since it is not clearly exponential. For the first case, it is shown on Figure 5.4. It follows an exponential growth, but it is not a perfect curve.

For the cases of the Local Structures, on Figure 5.5 it is shown for some that tend to grow over size, but with many ups and downs and for some big sizes being zero (note that the right bottom representation view is limited to 90 nm), and on Figure 5.6 for Local Structures that do not follow any tendency. Additionally, there is the presence on the *NanoFingerprints* of many

common parts where most of values are zero, and there are some of them different, being some of them low quantities and others big.

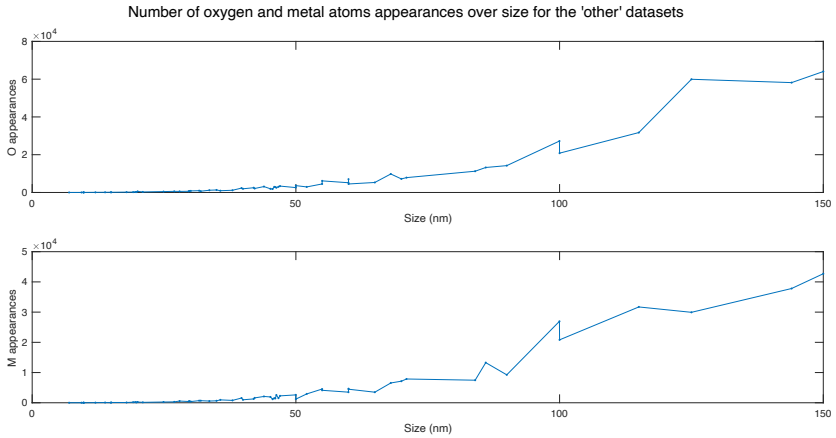


Figure 5.4. Representation of the apparition number of the Oxygen and metal atoms over for 'other' datasets.

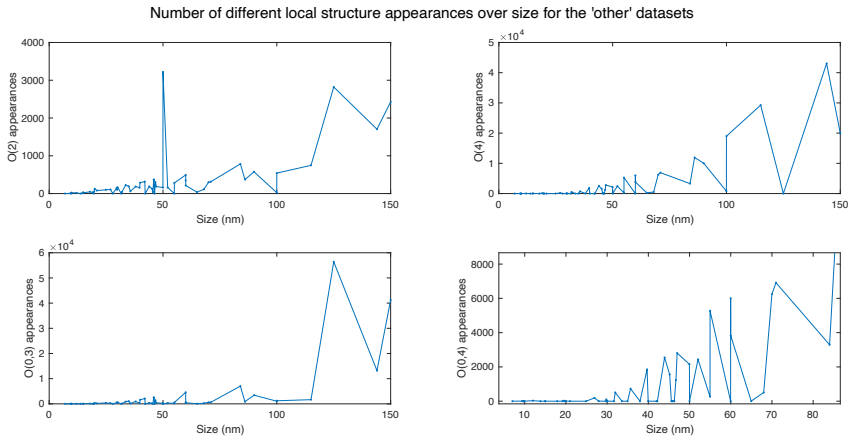


Figure 5.5. Representation of the apparition number of the Local Structures $O(2)$, $O(4)$, $O(0,3)$ and $O(0,4)$ over size for 'other' datasets.

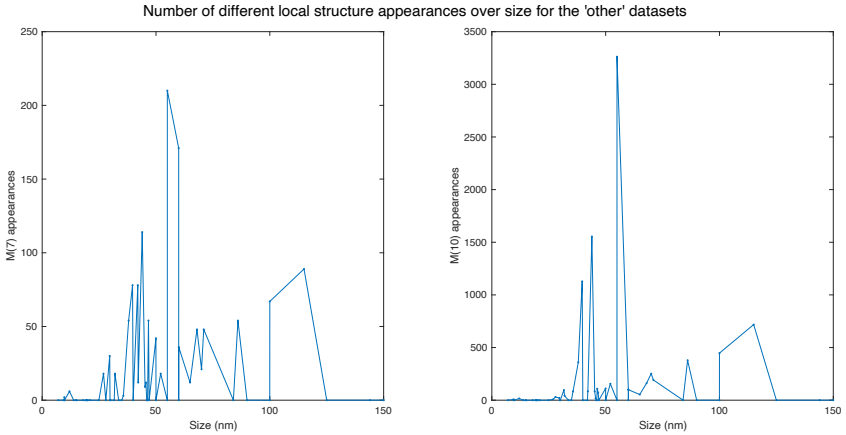


Figure 5.6. Representation of the apparition number of the Local Structures $M(7)$ and $M(10)$ over size for 'other' datasets.

5.2. The regression model

A regression is a supervised-learning machine learning algorithm. Given a set of one or more predictors, and their responses, a regression model is fitted in order to predict new responses given new predictors.

It has been used MATLAB in order to fit a regression model and make predictions with it, using the function *fitlm* [1]. This function, given the following inputs:

- *X*, matrix of predictors (one column for each one, each row is a different pattern).
- *y*, vector of responses.

Then, it returns the model. The model display includes the model formula, estimated coefficients, and model summary statistics.

The model formula in the display, for a Linear Model and two predictors, is in the following format (*Wilkinson Notation* [2]):

$$y \sim 1 + x1 + x2$$

Which corresponds to:

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \epsilon$$

Where ϵ represents the error that is assumed.

The estimated coefficients β_i can be obtained from the Coefficients property (*model.Coefficients*). The coefficients property includes these columns:

- *Estimate*. Coefficient estimates for each corresponding term in the model. It also includes the constant term (intercept), β_0.
- *SE*. Standard error of the coefficients.
- t*Stat. t*-statistic for each coefficient to test the null hypothesis that the corresponding coefficient is zero against the alternative that it is different from zero, given the other predictors in the model. It is calculated as:

$$tStat = \frac{Estimate}{SE}$$

- *pValue*. *p*-value for the *t*-statistic of the two-sided hypothesis test. For example, if the *p*-value of the *t*-statistic for a predictor X_j is greater than 0.05, this term is not significant at the 5% significance level given the other terms in the model.

The model also displays its summary statistics, which are:

- *Number of observations*. Number of rows without any NaN values.
- *Error degrees of freedom (n, p)*, where *n* is the number of observations, and *p* is the number of coefficients in the model, including the intercept.
- *Root mean squared error*. Square root of the mean squared error, which estimates the standard deviation of the error distribution.
- *R-squared and Adjusted R-squared*. Coefficient of determination and adjusted coefficient of determination, respectively. The R-squared value suggests that the model explains approximately a certain percentage of the variability in the response variable (goodness of the fit). The adjusted R-squared takes into account the number of predictors in the model, penalizing the inclusion of unnecessary predictors.
- F-statistic vs. constant model. Test statistic for the *F*-test on the regression model, which tests whether the model fits significantly better than a degenerate model consisting of only a constant term (overall significance of the regression model).
- p-value. *p*-value for the *F*-test on the model. It represents the probability of observing an F-statistic as extreme as the once calculated, assuming the null hypothesis that the true model has no predictive power. Then, it can be said that the model is significant with a certain *p*-value.

These statistics can be obtained in the model properties (*model.NumObservations*, *model.DFE*, *model.RMSE* and *model.Rsquared*) and by using the anova function as *anova(model,'summary')*.

The previous *fitlm* function usage is for a *Linear* model, which contains an in intercept and linear term for each predictor. Additionally, a non-linear quadratic model is wanted to be used. Therefore, the model specification has to add the parameter *'quadratic'* after the X and y parameters. This new model contains an intercept term and linear and squared terms for each predictor.

Once the model is obtained, with the function *predict*, given the model and the matrix of predictors, the responses vector can be obtained as:

$$y = predict(model, X)$$

Since the predicted values can be decimal, and *NanoFingerprint* values represent an integer count, values are rounded after prediction and negative ones are set to 0.

Depending on the case, as it has been explained, the matrix of predictors will consider more or less parameters. Note that, for each output, which correspond to a value from the *NanoFingerprint*, a different model needs to be fitted and generated, so that it can be used in the *predict* function and an output can be generated.

5.3. Pre-processing and postprocessing

Before introducing the appropriate data into the *fitlm* function in order to obtain the model, a pre-processing step takes places, involving a pre-treatment on the data.

Using MATLAB, firstly, the *NanoFingerprints* are imported and saved to a matrix, ordered by size (using the *sortrows* function, with both datasets).

Then, the following parameters are selected:

- n_{start}: the index from the *NanoFingerprint* from which on it will be considered. The *NanoFingerprints* below, will not be considered. This is used only for the *Atena dataset*, in which *NanoFingerprints* of lower sizes were not good for the regression since they contributed to giving noise rather than improving the results. For the other datasets, this parameter is set to 1.

- n_{train}: the number of *NanoFingerprints* used for training. The rest will be part of the test set.

- $m_{predictor\_params}$: the number of predictors used. Depending on the dataset, and regression, it will be one specific number.

- *standardize*: this parameter is set to '1' if the data is standardized before being introduced to the model, and '0' otherwise. When input and output variables have a different range of variation, to not give a priori more importance to some of the variables with respect to the others, they should be scaled scale to the same range of variation. This can be done by normalization ([min, max] range) or standardization.

- *shuffle*: this parameter is set to '1' if the *NanoFingerprints* are shuffled (moved randomly from its position) or not. It is recommended to shuffle the data to avoid any kind of sorting.

- *cross-validation*: this parameter is set to '1' if *cross-validation* is done, '0' otherwise. In case that *cross-validation* is applied, a K-Fold *cross-validation* or Leave-One-Out *cross-validation* is done, selecting the wanted K. Otherwise, the data is simply divided into a training and test set.

The detection of outliers, another key step in pre-processing for Machine Learning, has not been explained since no outliers were detected and thus considered. Next, the *Thickness* and *MAX* columns are discarded, since they are the same ones for the *NanoFingerprints*, regardless the dataset. Additionally, the first n_{start} *NanoFingerprints* are discarded.

It was noticed that many positions of the *NanoFingerprints* were 0 for all of them. Thus, these columns have been discarded. On the *Atena* dataset, from 44191 initial positions (*NanoFingerprint* length), it has gone to 115 non-zero positions (44076 positions from the whole *NanoFingerprint* are 0-columns for all the *Atena* dataset *NanoFingerprints*). On the case of the other datasets, it has gone from 44191 initial positions to 604 non-zero positions (43536 positions from the whole *NanoFingerprint* are 0-columns for all the other datasets *NanoFingerprints*). These 0-columns will be reinserted in their correct position after the prediction of the non-zero columns, thus, the overall error will be substantially reduced, since the probability of not having a zero in those columns is very low.

The scaling of data is a method to standardize the data, and a step done on the pre-processing stage [3]. It is an important treatment on Machine Learning (ML). The selected method has been standardization, which consists of subtracting a quantity related to a measure of localisation or distance and dividing by a measure of the scale. It has been applied the z-score or standard score, where z is the initial matrix of predictors and responses and z' the matrix after standardization.

$$z' = \frac{z - \mu}{\sigma}$$

Where μ is the mean of a certain *NanoFingerprint* position (column) and σ is the standard deviation of a certain *NanoFingerprint* position (column). Note that both μ and σ are vectors, since each value of the vector corresponds to a different *NanoFingerprint* column.

In the case of this work, the data is standardized to obtain the model and predict, correctly de-standardizing the outputs later. However, standardizing have had very few effects or none. In those cases that the results have varied, it has been just in some decimals, and because when de-standardizing values these are rounded so as not to have decimal values. Mainly, it has an effect on the model statistics, since non-standardizing effects on having difference in values, thus, higher model errors (SE, for instance). Standardizing additionally can reduce multicollinearity, which increases the precision of the model and can make p-values become significant.

The next step is to shuffle the *NanoFingerprints* saved on the matrix (randomly change the row positions).

The last step of the pre-processing stage is splitting the data into a train and test set as:

- X_{train}: matrix of predictors, where rows range from position 1 to n_{tran} and columns from 1 to $m_{predictor\_params}$.
- y_{train_real}: matrix of real responses, where rows range from position 1 to n_{tran} and columns from $m_{predictor\_params}$ + 1 to the end.
- X_{test}: matrix of predictors, where rows range from position n_{train} +1 to the end, and columns from 1 to $m_{predictor\_params}$.
- y_{train_real}: matrix of real responses, where rows range from position n_{train} + 1 to the end, and columns from $m_{predictor\_params}$ + 1 to the end.

Then, there are two steps. The first one performs a k-Fold or Leave-One-Out *cross-validation*, as explained on 5.4. Cross-validation. It consists on splitting the data into a training and validation set K times, leaving some of the data for testing. On each fold, it is fitted a model with the train set and the outputs are predicted with the validation set. Doing a cross-validation, the error metrics are averaged and better approximated to see if the model and data is suitable for this problem. The second steps uses the last fold model generated on cross-validation to make a prediction with the same fold validation set together with the test set model, calculating the error metrics as well and additionally presenting some plots of the results. In this way, a completely new set is tested. Both approaches will be presented on 5.6. NanoFingerprint generation without XYZ nanoparticle structures and 5.7. NanoFingerprint generation having the XYZ nanoparticle structures.

Finally, it comes the postprocessing stage. In both approaches, once the outputs have been calculated, if the input data had been standardized, the outputs (y') for both training and test sets are set to the original range (y) and the predictors (X') for both training and test sets to their initial values (X) as well, as:

$$y = y' \cdot \sigma + \mu$$

$$X = X' \cdot \sigma + \mu$$

Where y is the y_{train_pred} and y_{test_pred} (output predictions).

Then, the *NanoFingerprint* is reconstructed using X and y, being X the matrix of predictors and y the matrix of predicted responses, for both training and test sets. Additionally, the columns that were eliminated (zero-columns, Thickness and MAX) are restored to their initial column-position.

Next, the *NanoFingerprint* matrix is reordered by *NanoFingerprint* size (increasingly).

Finally, the error metrics are calculated as detailed in 5.5. Error metrics.

It has to be commented that previously to any function that uses randomness the instruction rng('default') is used to initialize the random seed to a fixed value and obtain the same random values. This has the advantage of always obtaining the same results, for reproducibility, and being able to compare, for instance, plots on the same *NanoFingerprints* for the different approaches.

5.4. Cross-validation

Cross-validation is a model assessment technique used to evaluate a machine learning algorithm's performance in making predictions on new datasets that it has not been trained on [4]. This is done by partitioning the known dataset, using a subset to train the algorithm and the remaining data for validation.

Each round of cross-validation involves randomly partitioning the original dataset into a training set and a validation set. The training set is then used to train a prediction algorithm and the validation set is used to evaluate its performance. This process is repeated several times and the average cross-validation error is used as a performance indicator.

When training a model, it is important not to overfit or underfit it with algorithms that are too complex or too simple. The choice of training set and validation set are critical in reducing this

risk. However, dividing the dataset to maximize both learning and validity of validation results is difficult. This is where cross-validation comes into practice. Cross-validation offers several techniques that split the data differently, to find the best algorithm for the model.

Cross-validation also helps with choosing the best performing model by calculating the error using the validation dataset, which has not been used to train. The testing dataset helps calculate the accuracy of the model and how it will generalize with future data.

Many techniques are available for cross-validation. Among the most common, the used ones on this work have been:

- *k-fold*: Partitions data into k randomly chosen subsets (or folds) of roughly equal size. One subset is used to validate the model trained using the remaining subsets. This process is repeated k times such that each subset is used exactly once for validation. The error is averaged across all k partitions. This is one of the most popular techniques for cross-validation but can take a long time to execute because the model needs to be trained repeatedly.

- *Leave-One-Out* (*LOOCV*): Partitions data using the k-fold approach where k is equal to the total number of observations in the data and all data will be used once as a validation set.

MATLAB allows partitioning a set into k partitions using the function *cvpartition* as:

cv = cvpartition(N,'KFold',k)

Which creates a *cvpartition* object *cv* defining a random partition for k-fold cross-validation on N observations. The partition divides N observations into k disjoint subsamples (folds), chosen randomly but with roughly equal size. Then, the testing set for each fold can be chosen by getting the indices as *cv.test(i)*, and then selecting them on the data, while the training set for each fold can be obtained by getting the indices as *~cv.test(i)*, and then selecting them on the data.

The methodology followed on this work, in the case that the variable cross-validation is set to '1', is explained as follows. For cross-validation, the training data is the one used (X_{train}, y_{train_real}), which is then split into K subsets. The validation data ($X_{validation}$, $y_{validation\_real}$), then, is left as new data to test the model once the cross-validation has been done, having selected the appropriate metrics (K, standardization, train set size, in our case).

Additionally, if cross-validation is set to '0', cross-validation is skipped and the train set is used to fit the model, and the test set to test it.

5.5. Error metrics

In order to obtain a metric to observe the correctness of the model generated and compare them, three different error metrics have been used [5]:

- *Normalized Mean Square Error* (*NMSE*). It is implemented in the function *calculateMSE.m*. Given a matrix of real (y_{real}) and predicted (y_{pred}) responses, rows being different *NanoFingerprints* and columns the different output values of the *NanoFingerprint*, it calculates the NMSE as:

  1. It subtracts each element from y_{pred} to y_{real} and divides the result by the maximum of the absolute value of y_{pred} and y_{real}. Thus, the result of each value is between 0 and 1, being 0 the best result (no error) and 1 the worst result. This last one, means either that one of them is 0, or that one of them is much bigger than the other.

$$result_1 = \frac{y_{real} - y_{pred}}{max(|y_{real}|, |y_{pred}|)}$$

  2. All Nans values are set to 0 since both numerator and denominator could be 0.

  3. The previous result, $result_1$, is multiplied by its transposed and divided by the total number of responses ($N_{outputs}$).

$$result_2 = \frac{result_1 \cdot result_1{}'}{N_{outputs}}$$

  4. Finally, since $result_2$ is a matrix (the input were two matrices), its diagonal is obtained (it corresponds to the value obtained for each pair of responses, real and predicted, from the same *NanoFingerprint*), summed, and divided (normalized) by its size ($size_{diag}$), obtaining the NMSE:

$$NMSE = \frac{sum(diag(result_2)}{size_{diag}}$$

- *Mean Absolute Error* (*MAE*). It is another useful measure of the prediction error. It is implemented in the function *calculateMae.m*. Given a matrix of real (y_{real}) and

predicted (y_{pred}) responses, rows being different *NanoFingerprints* and columns the different output values of the *NanoFingerprint*, it calculates the MAE as:

$$MAE = \frac{\sum_{\mu} \left| {y_{pred}}^{\mu} - {y_{real}}^{\mu} \right|}{\sum_{\mu} {y_{real}}^{\mu}}$$

Where μ represents each different *NanoFingerprint* (pattern).

After, it converts all *Nans* and infinite values to 0, it is summed, since there is a value for each *NanoFingerprint*, and finally it is normalized by its size.

- *NanoFingerprint Correctness Checking*. As it has been explained in 4.2. NanoFingerprint correctness checking, it is checked the correctness of each artificially generated *NanoFingerprint* and then averaged, to obtain an error rate.

The NMSE and MAPE are computed for:

1. Train and Test predicted responses (y_{train_pred} and y_{test_pred}), comparing to its real outputs (y_{train_real} and y_{test_real}).

2. After the predicted *NanoFingerprint* is constructed, it is compared to the real *NanoFingerprint*. Note that the big presence of zero will make this error to be really small, thus, difficult to give an idea on how good the prediction was.

The correctness checking of the *NanoFingerprint* is only computed once, using the predicted *NanoFingerprint*.

5.6. NanoFingerprint generation without XYZ nanoparticle structures

Without having the XYZ structures, there are two cases: the first one, which is modelled using the *Atena* dataset, and the second one, modelled by the other datasets (described in 4.3.2. Other datasets computational time study).

The method used to obtain these models has been Non-Linear Regression (NLR), concretely, it is a quadratic model. On the case of the *Atena* dataset, it is used just the size as predictor. On the case of the other datasets, it is used both size and atomic number (from the element different from Oxygen) of each nanoparticle as predictors, thus, it is a Multiple Non-Linear Regression (MNLR).

5.6.1. Atena dataset

The aim of this model generation is to predict the elements of the *Atena* dataset *NanoFingerprints* (TiO_2 of sizes ranging from 2 Å to 200 Å) using the size as predictor variable. The atomic number parameter is omitted since it is the same one for each nanoparticle.

Therefore, the model is described as:c

$$y_{nm} = \beta_{0_m} + \beta_{1_m} X_n + \beta_{2_m} X_n^2$$

Where:

- m is the number of output variables (columns).
- n is the number of *NanoFingerprints*.
- β_{0_m} is the constant term (intercept) for each output column m of the *NanoFingerprint*.
- β_{1_m} is the first coefficient term for each output column m of the *NanoFingerprint*, corresponding to the size input predictor (X_n).
- β_{2_m} is the second coefficient term for each output column m of the *NanoFingerprint*, corresponding to the squared size input predictor (X_n).
- X_n is the predictor of each *NanoFingerprint* n (size).
- y_{nm} is the output predicted value of each output column of the *NanoFingerprint*.

The returned model in *Wilkinson* notation by MATLAB, equivalent to the previous one, is:

$$y \sim 1 + x1 + x1\text{^}2$$

The apparition of a quadratic element makes the regression being non-linear. The program used on this case has been *nanofingerprint_generator_Ti.m*. Apart from Thickness and MAX columns, the atomic number column is also discarded on the pre-processing stage. Then, the function *compute_regression.m* is used to generate the model, predict, and make the error calculations.

Having 99 *NanoFingerprints*, the first 19 (n_{start}-1) have been discarded and 70 have been used for training (train set). The 10 *NanoFingerprint* left have been used for testing (test set).

Firstly, with the training set, a k-Fold cross-validation with k=7 has been performed. This means that, on each fold, 60 observations are used for training and 10 for validation. On each fold, the errors described on 5.5. Error metrics have been calculated and then averaged, obtaining a more representative error rates (the same procedure has been followed for the other approach and dataset).

On Table 5.2 these error metrics are shown.

Error Metric	Result
$NMSE(y_{train\_real}, y_{train\_pred})$	0.200362
$NMSE(y_{validation\_real}, y_{validation\_pred})$	0.205995
$MAE(y_{train\_real}, y_{train\_pred})$	0.358517
$MAE(y_{validation\_real}, y_{validation\_pred})$	0.417629
$NMSE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.000612
$MAE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.001357
Average correctness checking error	2.038776

Table 5.2. Error metrics for k-Fold (k=7) cross-validation on the non-XYZ approach for Atena dataset.

Where the $nanofingerprint_{prediction}$ is the predicted *NanoFingerprint* matrix, reconstructed from the predictions on the training and validation sets and $nanofingerprint_{real}$ is the real *NanoFingerprint* matrix, generated using the algorithm described on 3.3. Final approach.

Note that there is randomness in the process (shuffling of the *NanoFingerprints*, selection of the folds), thus, results could vary; however, the aim of cross-validation is in fact flattening the changes that randomness could bring and imply the whole dataset.

The MAE errors are higher than the NMSE errors. Regarding the NMSE errors, note that when comparing the outputs, the results are both 0.2, for training and validation datasets. This is a good result, since it means that the validation sets, which take no part in the training, have the same error as the train sets, which take part on the training. The main errors come from those *NanoFingerprint* columns in which the tendency is not exponential but does not follow any tendency in specific. On the MAE case, the validation error is higher than the train error.

After reconstructing the *NanoFingerprint*, note that the errors are very low. This is mainly due to the presence of a high quantity of zeros, that are the same on both original and predicted *NanoFingerprints*.

Finally, the average correctness checking error is of only 2.03, which is a very good result. This means that the artificially generated *NanoFingerprints* are mostly generated following the *NanoFingerprint* correct structure.

On the second place, a *Leave-One-Out* (k=70) cross-validation has been performed. On Table 5.3 the results are shown.

On this new case, the results are very similar to the previous ones.

Error Metric	Result
$NMSE(y_{train\_real}, y_{train\_pred})$	0.202248
$NMSE(y_{validation\_real}, y_{validation\_pred})$	0.208018
$MAE(y_{train\_real}, y_{train\_pred})$	0.360677
$MAE(y_{validation\_real}, y_{validation\_pred})$	0.421849
$NMSE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.000615
$MAE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.001354
Average correctness checking error	2.059184

Table 5.3. Error metrics for Leave-One-Out (k=70) cross-validation on the non-XYZ approach for Atena dataset.

Finally, following the previous methodology, it has been used one of the cross-validation fitted models with 60 *NanoFingerprints* for training, and then it has been tested joining the validation (set corresponding with the training set) and test sets (20 *NanoFingerprints*), in order to observe the performance with completely new data, showing some representations of the results as well.

On Table 5.4 the results on the validation and test part are shown (NMSE, MAE and average correctness checking error). Note that the results are very close to the average ones obtained on the 7-Fold cross-validation for the validation set.

Error Metric	Result
$NMSE(y_{validation\_real}, y_{validation\_pred})$	0.217899
$MAE(y_{validation\_real}, y_{validation\_pred})$	0.419190
$NMSE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.000525
$MAE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.000941
Average correctness checking error	2.05

Table 5.4. Validation and test set error metrics for the for the non-XYZ approach for Atena dataset.

On Figure 5.7 a scatter plot representing the real outputs against the predicted ones for different *Local Structures* is depicted, for the validation and test outputs (number of appearances). The closer to the regression line, the better; thus, note that for these *Local Structures*, the results are very good. These are the *Local Structures* presented on Figure 5.1, for which the growth in function of size was clearly exponential.

On Figure 5.8 it is represented, for the same *Local Structures* from Figure 5.7, the real and predicted outputs (number of appearances) for each validation and test *NanoFingerprint* (pattern). The first 10 ones correspond to the validation set, whereas the last 10 ones correspond to the test set. Due to the shuffle of data, *NanoFingerprints* are not ordered by size, and this is the reason of the non-exponential appearance. It can be seen that the predicted results perfectly fit the real ones, and just differ in some counts; for instance, zooming on pattern 10, the real output is 211928 appearances whereas the predicted one is 210501. They differ on 1427 counts, but the difference is negligible at this scale for the application of *NanoFingerprints* use.

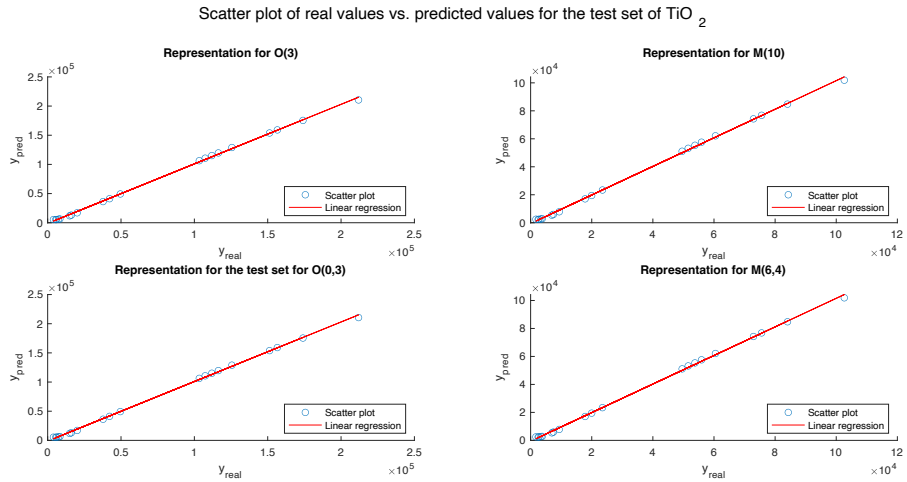


Figure 5.7. Scatter plot of real outputs vs. predicted ones for different *Local Structures* (validation and test sets).

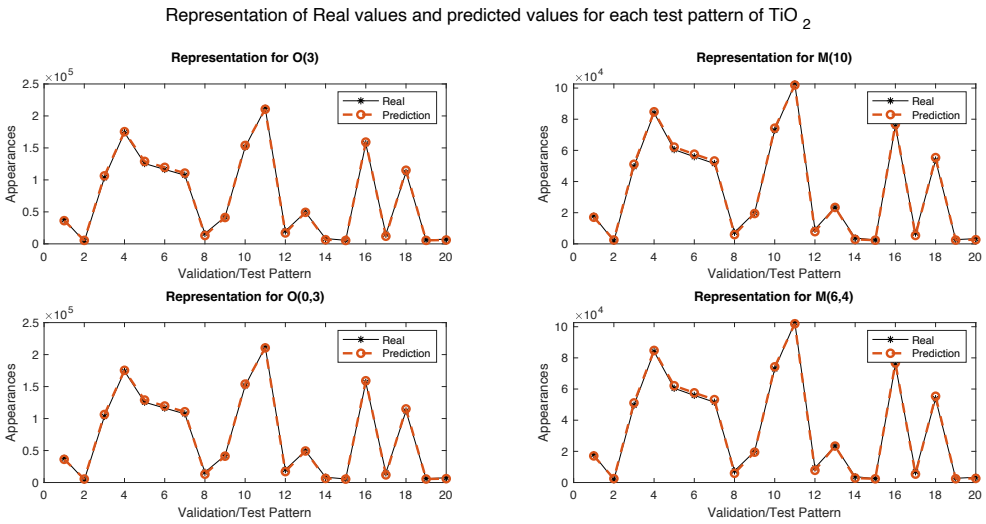


Figure 5.8. Representation of real and predicted values (validation and test sets) for each different *NanoFingerprint* (pattern).

Then, the following two figures (Figure 5.9 and 5.10) represent the same as the two previous ones, respectively, but for Local Structures which do not present a perfect exponential growth or do not present any kind of exponential growth at all. They are the *Local Structures* presented on Figure 5.3.

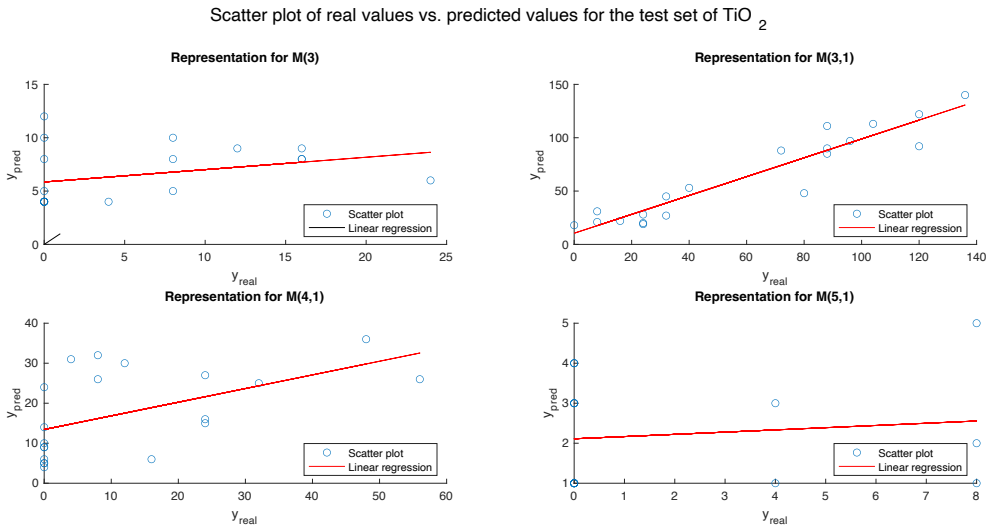


Figure 5.9. Scatter plot of real outputs vs. predicted ones for different *Local Structures* (validation and test sets).

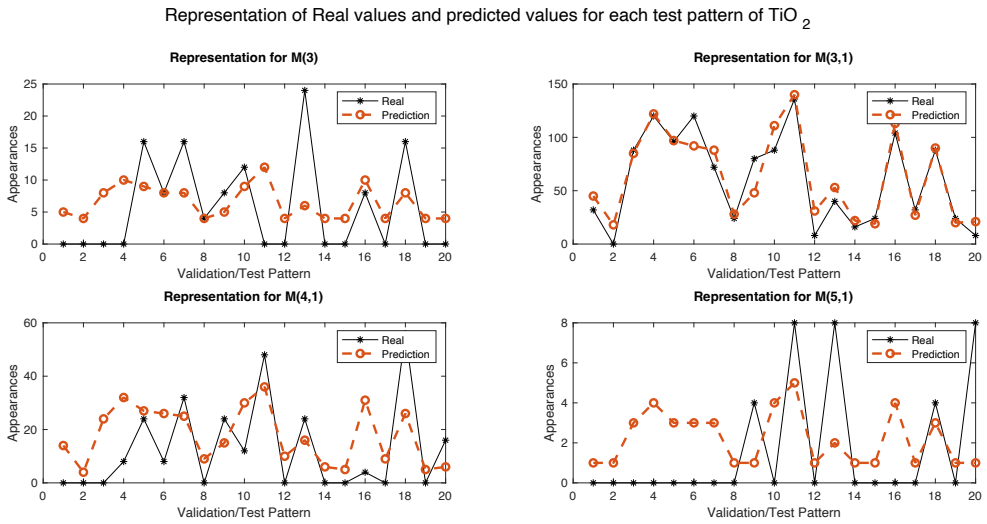


Figure 5.10. Representation of real and predicted values (validation and test sets) for each different *NanoFingerprint* (pattern).

Finally, two more representations are presented. On Figure 5.11 it is depicted the number of Oxygen and meal atoms in function of size (from 40 nm to 200 nm, since the first 20 *NanoFingerprints* were discarded). On this case, the predicted train outputs include the validation set (in green), and the predicted test outputs are represented on red. In blue, the real outputs are represented. Note that the predicted test outputs are mixed with the predicted train outputs, due to the initial shuffle of the *NanoFingerprints*. The result is very good.

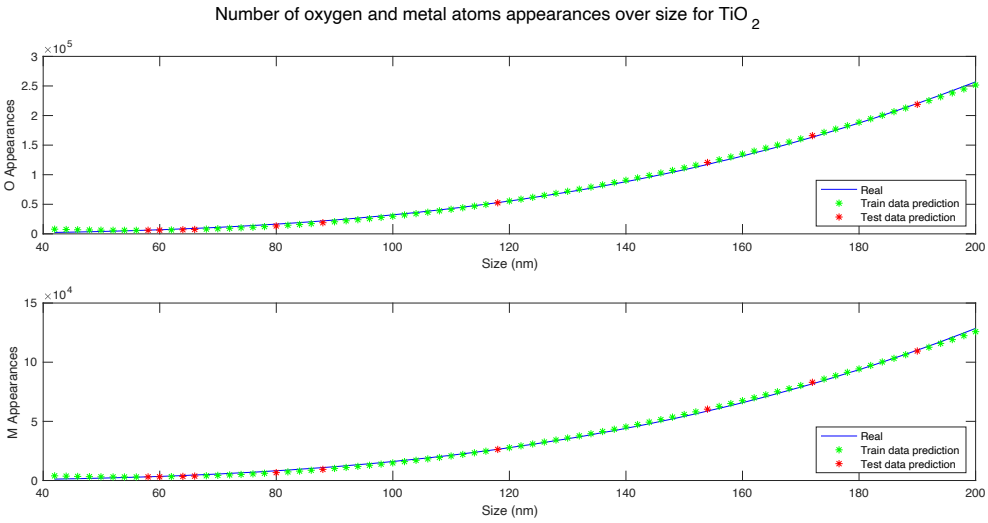


Figure 5.11. Representation of the number of Oxygen and metal atoms in function of size.

For the number of Oxygen atoms prediction, the NMSE=0.0273 and MAE=0.0271, while for the number of Metal atoms prediction, the NMSE=0.0273 and MAE=0.0269. Both results are very good.

On Figure 5.12, the representation is for different *Local Structures*, from best to worst. The first one ($O(0,3)$) perfectly fits the curve; the second one ($M(3,1)$), since the growth is exponential, although not perfect, the result is good as well, since it can serve to see the approximated evolution on the count. Thirdly, for $M(4,1)$, the evolution in function of size tends to grow, but with many ups and downs to high and low levels. Therefore, the predicted outcomes average the maximum and minimums. The last one ($M(5,1)$) does not follow any exponential tendency, so the result is not acceptable. These *Local Structures* have been shown as well on Figures 5.7, 5.8, 5.9 and 5.10.

Note that, on Figures 5.11 and 5.12, if the test set would not have been put together with the validation set, these last points set would have not appeared on the representations. The indices order obtained from the shuffle have been used to colour the first *n_train* points (which include the train and validation sets) in green and the rest ones in red (which include the test set).

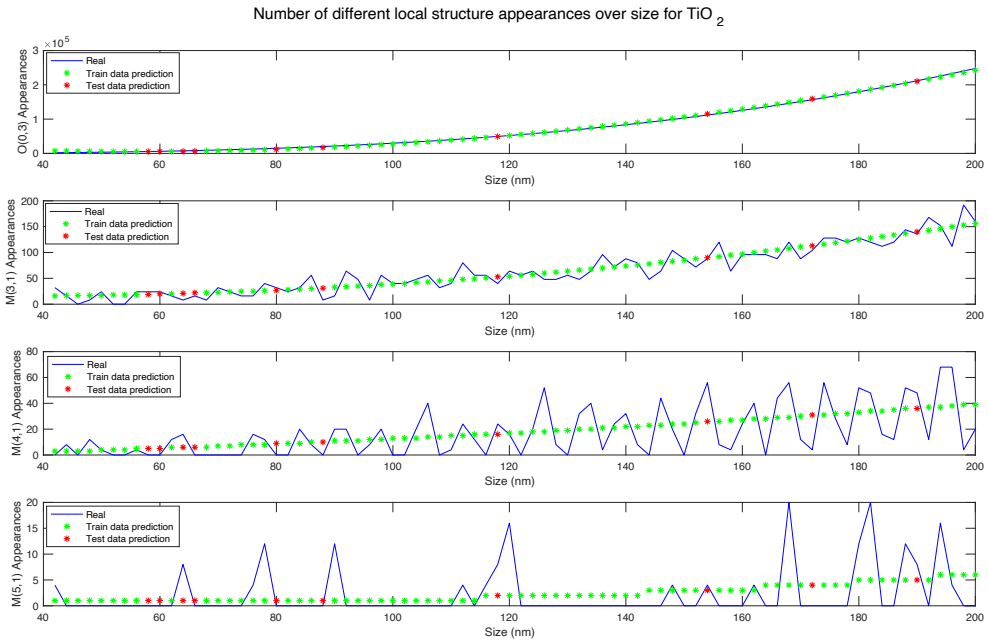


Figure 5.12. Representation of output appearances for different *Local Structures* in function of size.

Regarding to the statistics of the model generated to predict the number of Oxygens, on Figure 5.13 they are presented.

```
Estimated Coefficients:
                   Estimate       SE         tStat       pValue
                   ________    _________    _______    __________

    (Intercept)    -0.33097    0.0066059    -50.102    7.9358e-49
    x1              0.95841    0.0045651     209.94    4.8887e-84
    x1^2            0.33641    0.0048299      69.65    7.5142e-57


Number of observations: 60, Error degrees of freedom: 57
Root Mean Squared Error: 0.0352
R-squared: 0.999,  Adjusted R-Squared: 0.999
F-statistic vs. constant model: 2.52e+04, p-value = 1.05e-84
```

Figure 5.13. Model statistics for *NanoFingerprint* 'Number of Oxygen atoms' output.

The SE values are very low, which indicate that the estimated coefficients are relatively precise with low variability, providing more confidence in their accuracy.

The positive value of the *tStat* for x1 and x1^2 suggests that there is a significant positive relationship between the coefficients and the response variable, and a strong statistical significance of these predictor variables in explaining the variation in the response variable (substantial impact on the response variable in the regression model).

The *p-value* for the intercept is 7.9358e-49, which is very small, indicates strong evidence against the null hypothesis. For the coefficient of x1 is 4.8887e-84 and for the coefficient of x1^2 is 7.5142e-57, also indicating strong evidence against the null. The p-value associated with the F-statistic is 1.05e-84, which is extremely small, providing strong evidence against the null hypothesis and indicating that the regression model is statistically significant, as well as each Estimate.

The R-squared and Adjusted R-squared values are both of 0.999, indicating that the model explains 99.9% of the variance in the response variable.

The F-statistic is 2.52e+04, indicating a highly significant relationship between the predictors and the response variable.

Additionally, the model can be plotted, as shown on Figure 5.14. Note that the model perfectly fit into the data.

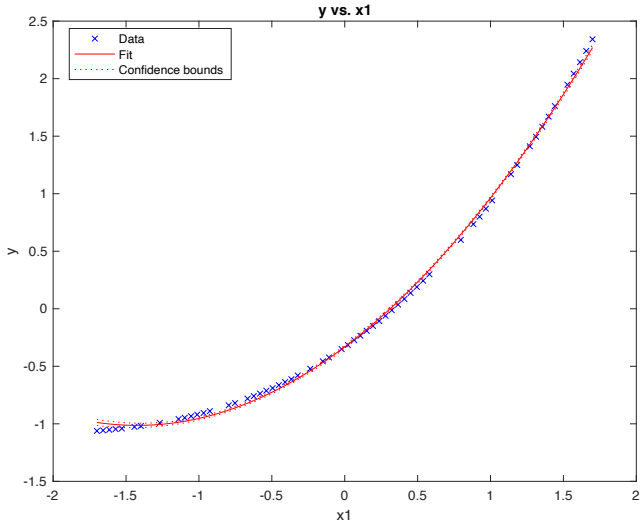


Figure 5.14. Plot of the model for *NanoFingerprint* 'Number of Oxygen atoms' output.

On the other hand, to show a bad model, it is shown the obtained to predict $M(5,1)$ on Figure 5.15.

```
Estimated Coefficients:
                   Estimate      SE        tStat       pValue
                   ________    ________    _________    ________

    (Intercept)     -0.111      0.20018    -0.55451      0.5814
    x1             0.28492      0.13834      2.0595    0.044025
    x1^2           0.14857      0.14637       1.015     0.31438


Number of observations: 60, Error degrees of freedom: 57
Root Mean Squared Error: 1.07
R-squared: 0.0876,  Adjusted R-Squared: 0.0556
F-statistic vs. constant model: 2.74, p-value = 0.0733
```

Figure 5.15. Model statistics for *NanoFingerprint Local Structure M(5,1)* output.

For the standard error (SE) case, all of them are much higher than on the previous case, showing that there is considerable uncertainty associated with this estimate.

The t-statistics for all of them show that the coefficients are not significantly different from zero. The relatively high p-values of 0.5814 and 0.31438, for the intercept and x1^2, respectively, indicate that they may not have a significant impact on the response variable, contrarily to x1, which may have a significant relationship with the response variable.

The RMSE value of 1.07 represents the standard deviation of the residuals, providing an estimate of the average prediction error. This is a large average prediction error.

The R-squared value is 0.0876, indicating that approximately 8.76% of the variance in the response variable is explained by the regression model. The adjusted R-squared value is

0.0556, which takes into account the number of predictors in the model and suggests that the predictors may not be strongly related to the response variable.

The F-statistic is 2.74, indicating a relatively weak relationship between the predictors and the response variable. The associated p-value of 0.0733 suggests that the regression model may not be statistically significant at a conventional significance level of 0.05.

The model is plotted on Figure 5.16. Note that the model do not fit into the data, being a bad model.

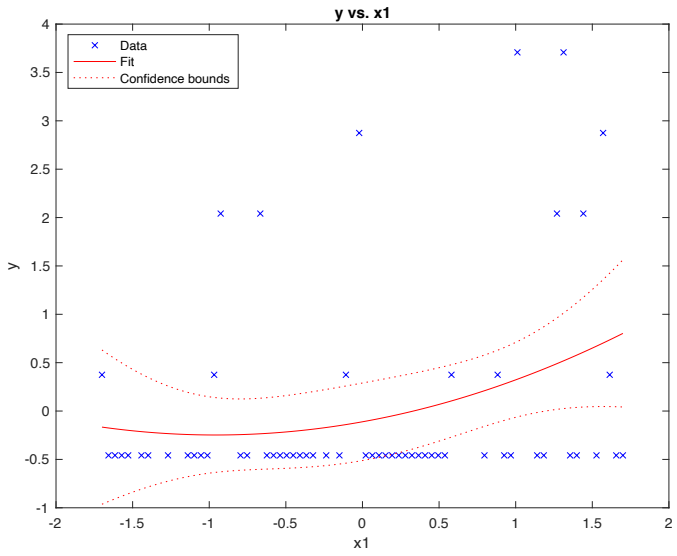


Figure 5.16. Plot of the model for *NanoFingerprint Local Structure M(5,1)* output.

The statistics of the model are only shown for this case, and not for the rest of approaches and datasets.

5.6.2. Other datasets

The aim of this model generation is to predict the elements of the other datasets *NanoFingerprints* using the size and atomic number (from the element different from Oxygen) of each nanoparticle as predictor variables.

Therefore, the model is described as:

$$y_{nm} = \beta_{0_m} + \beta_{1_m}X_{1n} + \beta_{2_m}X_{2n} + \beta_{3_m}X_{1n}X_{2n} + \beta_{4_m}X_{1n}^2 + \beta_{5_m}X_{2n}^2$$

Where:

- m is the number of output variables (columns).
- n is the number of *NanoFingerprints*.
- β_{0_m} is the constant term (intercept) for each output column m of the *NanoFingerprint*.
- β_{1_m} is the first coefficient term for each output column m of the *NanoFingerprint*, corresponding to the size input predictor (X_{1n}).
- β_{2_m} is the second coefficient term for each output column m of the *NanoFingerprint*, corresponding to the atomic number input predictor (X_{2n}).
- β_{3_m} is the third coefficient term for each output column m of the *NanoFingerprint*, corresponding to the interaction between the size input predictor (X_{1n}) and the atomic number input predictor (X_{2n}).
- β_{4_m} is the fourth coefficient term for each output column m of the *NanoFingerprint*, corresponding to the squared size input predictor (X_{1n}).
- β_{5_m} is the fifth coefficient term for each output column m of the *NanoFingerprint*, corresponding to the squared atomic number input predictor (X_{2n}).
- X_{1n} is the size predictor of each *NanoFingerprint* n.
- X_{2n} is the atomic number predictor of each *NanoFingerprint* n.
- y_{nm} is the output predicted value of each output column of the *NanoFingerprint*.

The returned model in *Wilkinson* notation by MATLAB, equivalent to the previous one, is:

$$y{\sim}1 + x1 * x2 + x1\hat{}2 + x2\hat{}2$$

Having more non-zero columns, thus, more outputs, the fit of the models will take longer time on this case than for the Atena dataset.

The apparition of a quadratic element makes the regression being non-linear. The program used on this case has been *nanofingerprint_generator_other.m*, which has very few modifications with respect to *nanofingerprint_generator_Ti.m* (basically on the parameters to choose, non-discarding the atomic number and the selection of the representations, and it was separated to another program for organization). The variable $m_{predictor\_params}$ has been set to 2. The function *compute_regression_other.m* is used to generate the model, predict, and make the error calculations (in the reconstruction part it was modified one index in comparison to *compute_regression.m*, but for organization reasons it was created a new function).

Having 60 *NanoFingerprints*, the first 14 (n_{start}-1) have been discarded and 44 have been used for training (train set). The 2 *NanoFingerprint* left have been used for testing (test set). The fact of discarding the less sized *NanoFingerprints* slightly improves the results, since the smaller ones introduced more noise than helped in fitting the models (the best is having from all sizes, not many on lower ones and few on higher). On the other hand, reducing the sample, it means having less *NanoFingerprints*.

Concretely, the following number of nanoparticles are not considered with this cut:

- 2 ZnO nanoparticles.
- 3 TiO_2 nanoparticles.
- 1 CuO nanoparticle.
- 1 WO_3 nanoparticle.
- 1 Al_2O_3 nanoparticle.
- 2 NiO_2 nanoparticles (in fact, the only two that are present in the sample).

However, these are the nanoparticles that are computationally non-expensive to create.

Firstly, with the training set, a k-Fold cross-validation with k=6 has been performed. This means that, on each fold, 37 observations are used for training and 7 for validation. On each

fold, the errors described on 5.5. Error metrics have been calculated and then averaged, obtaining a more representative error rates (the same procedure has been followed for the other approach and dataset).

On Table 5.5 these error metrics are shown.

Error Metric	Result
$NMSE(y_{train\_real}, y_{train\_pred})$	0.441335
$NMSE(y_{validation\_real}, y_{validation\_pred})$	0.480069
$MAE(y_{train\_real}, y_{train\_pred})$	1.477536
$MAE(y_{validation\_real}, y_{validation\_pred})$	1.886506
$NMSE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.006351
$MAE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.026144
Average correctness checking error	48.314394

Table 5.5. Error metrics for k-Fold (k=6) cross-validation on the non-XYZ approach for 'other' datasets.

Note that in general the errors are much higher than on the Atena dataset case. The MAE is higher than 1, meaning that the predicted and real values are much greater in ones compared to the others. The NMSE error show that there is also a big difference in most of the values (predicted or real are half of the other ones in average).

After reconstructing the *NanoFingerprint*, note that the errors are very low. However, the MAE is 0.02, which is starting to be considerable (about 2%), considering that there are many 0 columns.

Finally, the average correctness checking error is also huge, 48.31, describing that the generated *NanoFingerprints* do not follow the correct structure.

On the second place, a *Leave-One-Out* (k=44) cross-validation has been performed. On Table 5.6 the results are shown. On this new case, the results are very similar to the previous one, being slightly higher than on the previous case.

Error Metric	Result
$NMSE(y_{train\_real}, y_{train\_pred})$	0.459195
$NMSE(y_{validation\_real}, y_{validation\_pred})$	0.489819
$MAE(y_{train\_real}, y_{train\_pred})$	1.566858
$MAE(y_{validation\_real}, y_{validation\_pred})$	1.987549
$NMSE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.006618
$MAE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.025585
Average correctness checking error	48.922004

Table 5.6. Error metrics for Leave-One-Out (k=44) cross-validation on the non-XYZ approach for 'other' datasets.

Finally, following the previous methodology, it has been used one of the cross-validation fitted models with 37 *NanoFingerprints* for training, and then it has been tested joining the validation (set corresponding with the training set) and test sets (9 *NanoFingerprints*), in order to observe the performance with completely new data, showing some representations of the results as well.

On Table 5.7 the results on the validation and test part are shown (NMSE, MAE and average correctness checking error). Note that the results are very close to the average ones obtained on the 6-Fold cross-validation for the validation set.

Error Metric	Result
$NMSE(y_{validation\_real}, y_{validation\_pred})$	0.459888
$MAE(y_{validation\_real}, y_{validation\_pred})$	1.856827
$NMSE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.005994
$MAE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.022525
Average correctness checking error	44.978261

Table 5.7. Validation and test set error metrics for the for the non-XYZ approach for 'other' datasets.

On Figure 5.17 a scatter plot representing the real outputs against the predicted ones for different *Local Structures* is depicted, for the validation and test outputs (number of appearances). The first 7 values correspond to the validation set, whereas the last 2 ones belong to the test set. The closer to the regression line, the better; thus, note that for these *Local Structures*, the results are quite good. Note that there is one point on the right-extreme and many others on the left-extreme. This is because the validation and test sets, randomly chosen, incorporated a nanoparticle of big size, and many others of low size. Some of these *Local Structures* are presented on Figure 5.5.

On Figure 5.18 it is represented, for the same *Local Structures* from Figure 5.17, the real and predicted outputs (number of appearances) for each validation and test *NanoFingerprint* (pattern). As well as on Figure 5.17, the first 7 ones correspond to the validation set, whereas the last 2 ones correspond to the test set. Due to the shuffle of data, *NanoFingerprints* are not ordered by size from left to right. It can be seen that the predicted results are close to the real ones, despite differing in some thousands of counts (for instance, on the fifth pattern), but it guesses the tendency.

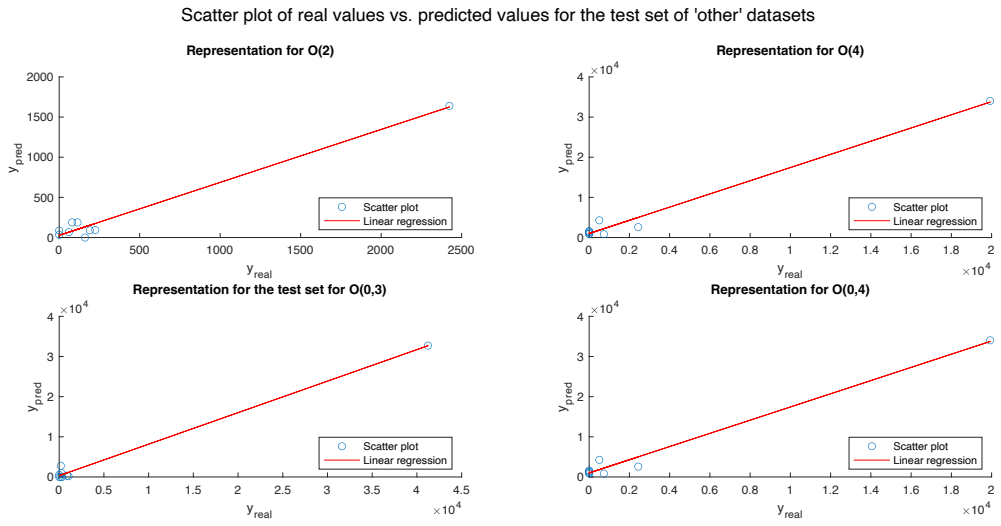


Figure 5.17. Scatter plot of real outputs vs. predicted ones for different *Local Structures* (validation and test sets).

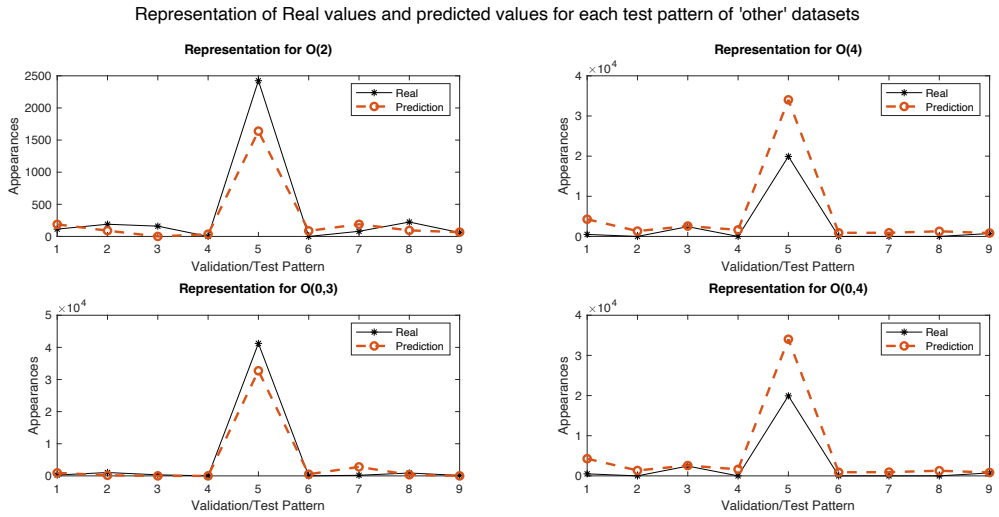


Figure 5.18. Representation of real and predicted values (validation and test sets) for each different *NanoFingerprint* (pattern).

Then, the following two figures (Figure 5.19 and 5.20) represent the same as the two previous ones, respectively, but for Local Structures which do not present any kind of exponential growth. The right representation does not even guess the tendency, whereas the left one, despite guessing it, differs in the order of 10^4 counts.

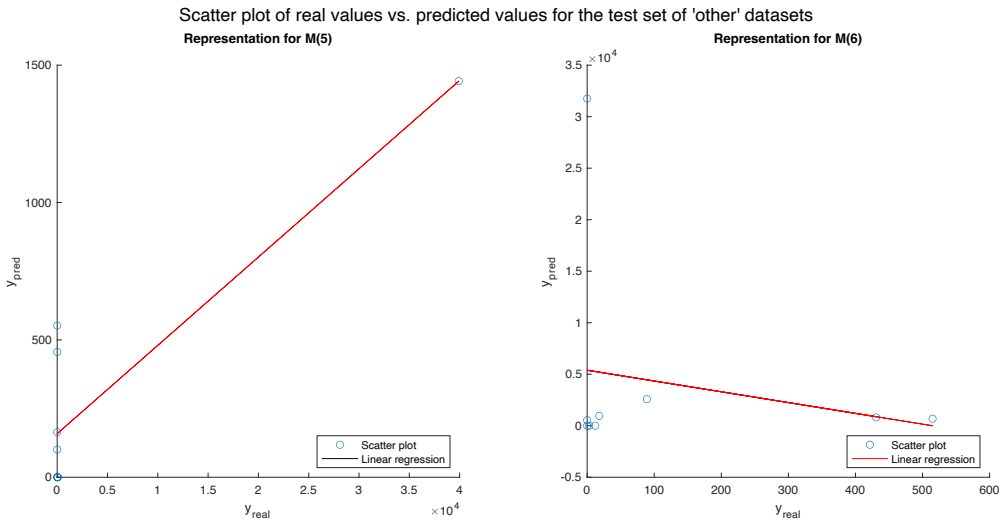


Figure 5.19. Scatter plot of real outputs vs. predicted ones for different *Local Structures* (validation and test sets).

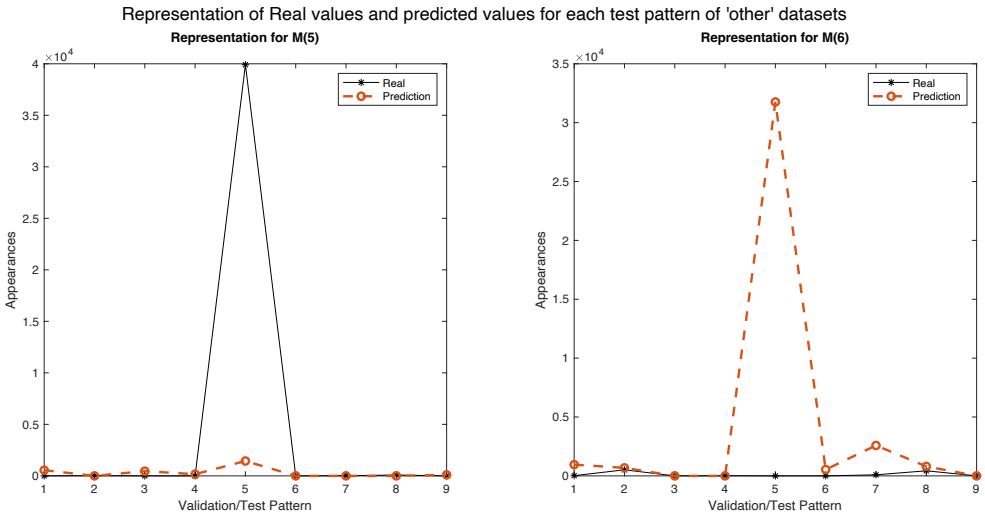


Figure 5.20. Representation of real and predicted values (validation and test sets) for each different *NanoFingerprint* (pattern).

Finally, four more representations are presented. On Figure 5.21.1 it is depicted the number of Oxygen and meal atoms in function of size (without considering the first n_{start} discarded ones). On this case, the predicted train outputs include the validation set (in green), and the predicted test outputs are represented on red. In blue, the real outputs are represented. Note that the predicted test outputs are mixed with the predicted train outputs, due to the initial shuffle of the *NanoFingerprints*. The result is very acceptable.

On Figure 5.21.2 it is presented the same as on Figure 5.21, but focusing on sizes up to 80 nm, to observe how is the prediction there. Note that, despite not being perfect, it approximates. The results are much better for the metal atoms count rather than the Oxygen atoms count: as it can be seen, for the Oxygen atoms case, the result for low sizes differ quite a lot.

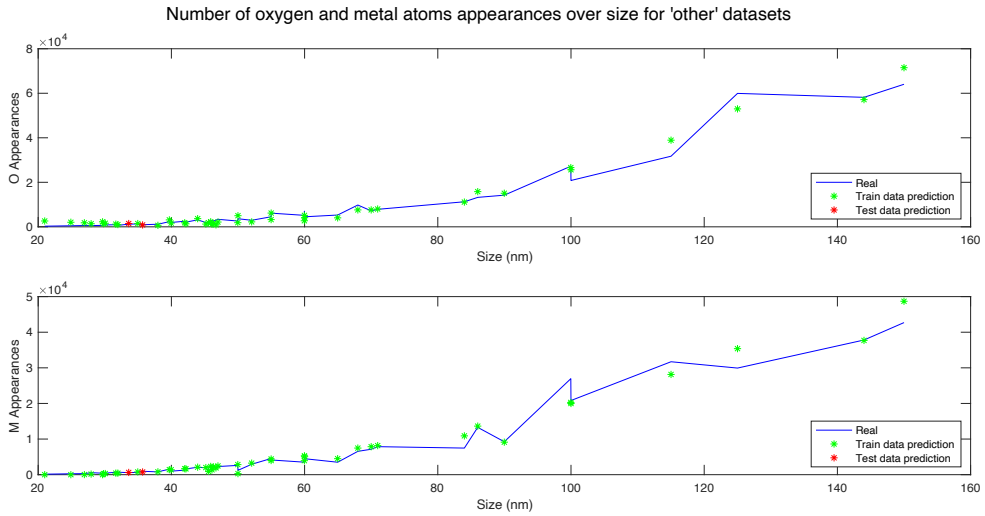


Figure 5.21. Representation of the number of Oxygen and metal atoms in function of size.

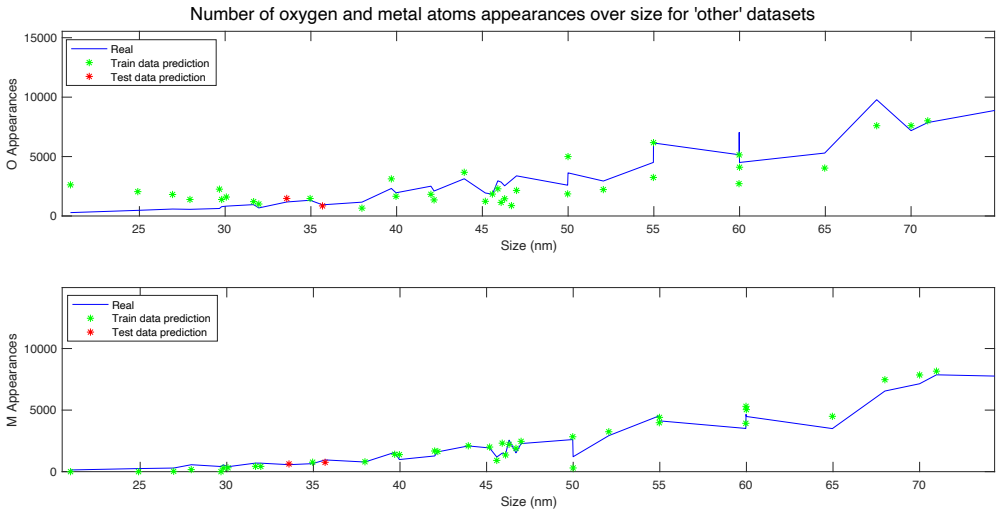


Figure 5.21. Representation of the number of Oxygen and metal atoms in function of size (up to 80 nm).

For the number of Oxygen atoms prediction, the NMSE=0.1314 and MAE=0.1584, while for the number of Metal atoms prediction, the NMSE=0.1362 and MAE=0.1279. Both results are acceptable.

On Figure 5.22, the representation is for different *Local Structures*. Predictions are not very good but try to approximate to the real ones. There are some cases in which the fit is perfect or very close, such as for the point on size = 144 nm, on the three representations.

Additionally, note that there are more samples of low size nanoparticles than big sized ones. This, as well as having more nanoparticles of one metal element than others, also leads to unbalanced training. Figure 5.23 is the same representations but with a window of sizes up to 80 nm. The main problem comes from the data, since it cannot be described using a concrete mathematical expression, making a regression not be the most suitable solution to this problem.

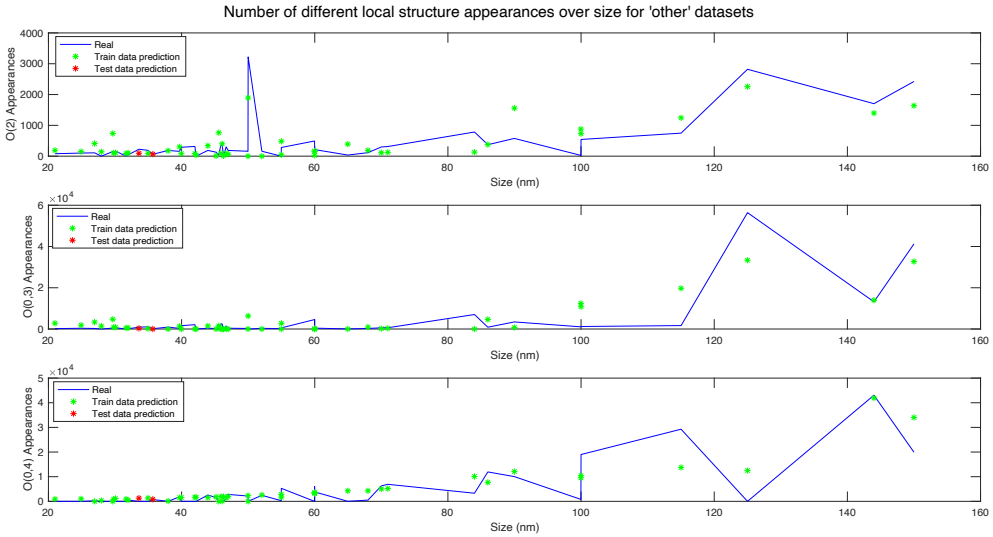


Figure 5.22. Representation of output appearances for different *Local Structures* in function of size.

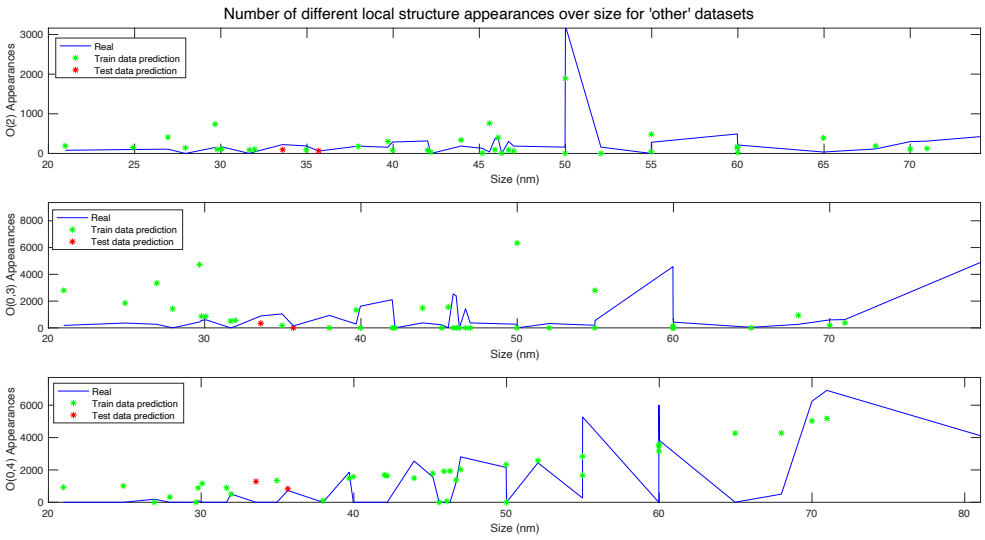


Figure 5.23. Representation of output appearances for different *Local Structures* in function of size (up to 80 nm).

5.7. NanoFingerprint generation having the XYZ nanoparticle structures

Having the XYZ structures, there are two cases as well: the first one, which is modelled using the *Atena* dataset, and the second one, modelled by the other datasets (described in 4.3.2. Other datasets computational time study).

On the case of the *Atena* dataset, it is used the size, and the number of Oxygen and Metal atoms of the nanoparticle as predictors. The method used has been Multiple Linear Regression (MLR) and Multiple Non-Linear Regression (MNLR). On the case of the other datasets, it is used the size, the number of Oxygen and Metal atoms of the nanoparticle and the atomic number as predictors. The method used has been Multiple Linear Regression (MLR).

5.7.1. Atena dataset

The aim of this model generation is to predict the elements of the *Atena* dataset *NanoFingerprints* (TiO_2 of sizes ranging from 2 Å to 200 Å) using the size, and the number of Oxygen and Metal atoms of the nanoparticle as predictor variables. The atomic number parameter is omitted since it is the same one for each nanoparticle.

Two models have been tested, using the same predictors, to observe if both are suitable and conclude which one is the most appropriate one. The first one is linear, while de second one is non-linear (quadratic).

The first one described as:

$$y_{nm} = \beta_{0_m} + \beta_{1_m} X_{1n} + \beta_{2_m} X_{2n} + \beta_{3_m} X_{3n}$$

Where:

- m is the number of output variables (columns).
- n is the number of *NanoFingerprints*.
- β_{0_m} is the constant term (intercept) for each output column m of the *NanoFingerprint*.
- β_{1_m} is the first coefficient term for each output column m of the *NanoFingerprint*, corresponding to the size input predictor (X_{1n}).
- β_{2_m} is the second coefficient term for each output column m of the *NanoFingerprint*, corresponding to the number of Oxygen atoms input predictor (X_{2n}).

- β_{3_m} is the third coefficient term for each output column m of the *NanoFingerprint*, corresponding to the number of Metal atoms input predictor (X_{3n}).
- X_{1n} is the size predictor of each *NanoFingerprint* n.
- X_{2n} is the number of Oxygen atoms predictor of each *NanoFingerprint* n.
- X_{3n} is the number of Metal atoms predictor of each *NanoFingerprint* n.
- y_{nm} is the output predicted value of each output column of the *NanoFingerprint*.

The returned model in *Wilkinson* notation by MATLAB, equivalent to the previous one, is:

$$y \sim 1 + x1 + x2 + x3$$

There is not a quadratic element, being a linear regression. The program used on this case has been *nanofingerprint_generator_Ti.m* (changing just the $m_{predictor\_params}$ parameter to 3 with respect to the previous approach and the model type to 'linear'). Apart from Thickness and MAX columns, the atomic number column is also discarded on the pre-processing stage. Then, the function *compute_regression.m* is used to generate the model, predict, and make the error calculations.

Having 99 *NanoFingerprints*, the first 19 (n_{start}-1) have been discarded and 70 have been used for training (train set). The 10 *NanoFingerprint* left have been used for testing (test set).

Firstly, with the training set, a k-Fold cross-validation with k=7 has been performed (60 observations are used for training and 10 for validation). On each fold, the errors described on 5.5. Error metrics have been calculated and then averaged, obtaining a more representative error rates (the same procedure has been followed for the other approach and dataset). The data is standardized to obtain the model and predict, correctly de-standardizing the outputs later.

On Table 5.8 these error metrics are shown.

Error Metric	Result
$NMSE(y_{train\_real}, y_{train\_pred})$	0.196265
$NMSE(y_{validation\_real}, y_{validation\_pred})$	0.202166
$MAE(y_{train\_real}, y_{train\_pred})$	0.350541
$MAE(y_{validation\_real}, y_{validation\_pred})$	0.412715
$NMSE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.000610
$MAE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.001375
Average correctness checking error	2.269388

Table 5.8. Error metrics for k-Fold (k=7) cross-validation on the first case XYZ approach for Atena dataset.

The MAE errors are higher than the NMSE errors. Regarding the NMSE errors, note that when comparing the outputs, the results are very similar for training and validation datasets. This is a good result, since it means that the validation sets, which take no part in the training, have the same error as the train sets, which take part on the training. As well on the previous approach for this dataset, the main errors come from those *NanoFingerprint* columns in which the tendency is not exponential but does not follow any tendency in specific. On the MAE case, the validation error is higher than the train error.

After reconstructing the *NanoFingerprint*, note that the errors are very low. This is mainly due to the presence of a high quantity of zeros, that are the same on both original and predicted *NanoFingerprints*.

Finally, the average correctness checking error is of only 2.269, which is a very good result. This means that the artificially generated *NanoFingerprints* are mostly generated following the *NanoFingerprint* correct structure.

On the second place, a *Leave-One-Out* (k=70) cross-validation has been performed. On Table 5.9 the results are shown. On this new case, the results are very similar to the previous one.

Error Metric	Result
$NMSE(y_{train\_real}, y_{train\_pred})$	0.197312
$NMSE(y_{validation\_real}, y_{validation\_pred})$	0.205308
$MAE(y_{train\_real}, y_{train\_pred})$	0.350077
$MAE(y_{validation\_real}, y_{validation\_pred})$	0.415013
$NMSE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.000612
$MAE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.001367
Average correctness checking error	2.170204

Table 5.10. Error metrics for Leave-One-Out (k=70) cross-validation on the first case XYZ approach for Atena dataset.

Finally, following the previous methodology, it has been used one of the cross-validation fitted models with 60 *NanoFingerprints* for training, and then it has been tested joining the validation (set corresponding with the training set) and test sets (20 *NanoFingerprints*), in order to observe the performance with completely new data, showing some representations of the results as well.

On Table 5.11 the results on the validation and test part are shown (NMSE, MAE and average correctness checking error). Note that the results are very close to the average ones obtained on the 7-Fold cross-validation for the validation set.

Error Metric	Result
$NMSE(y_{validation\_real}, y_{validation\_pred})$	0.216284
$MAE(y_{validation\_real}, y_{validation\_pred})$	0.414764
$NMSE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.000508
$MAE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.000911
Average correctness checking error	2.1375

Table 5.11. Validation and test set error metrics for the for the first case XYZ approach for Atena dataset.

On Figure 5.24 a scatter plot representing the real outputs against the predicted ones for different *Local Structures* is depicted, for the validation and test outputs (number of appearances). The closer to the regression line, the better; thus, note that for these *Local Structures*, the results are very good. These are the *Local Structures* presented on Figure 5.1, for which the growth in function of size was clearly exponential. The first 10 ones correspond to the validation set, while the last 10 ones to the test set.

On Figure 5.25 it is represented, for the same *Local Structures* from Figure 5.24, the real and predicted outputs (number of appearances) for each validation and test *NanoFingerprint* (pattern). As well as on Figure 5.24, the first 10 ones correspond to the validation set, whereas the last 10 ones correspond to the test set. Due to the shuffle of data, *NanoFingerprints* are not ordered by size, and this is the reason of the non-exponential appearance. It can be seen that the predicted results perfectly fit the real ones.

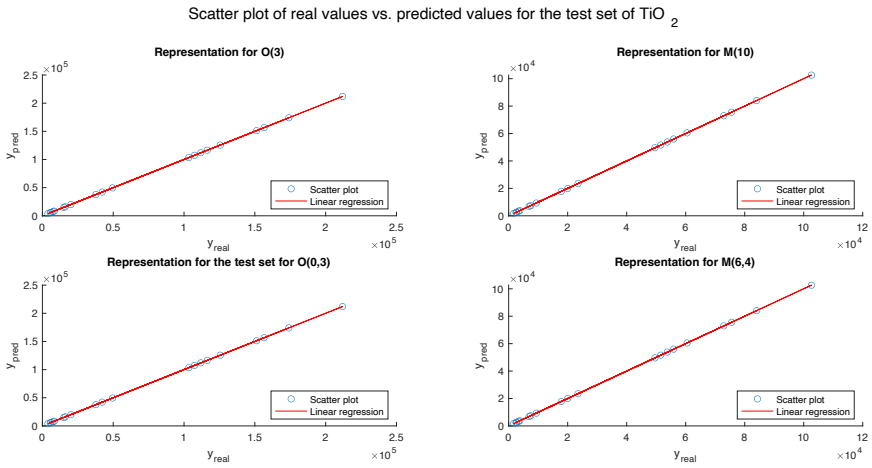


Figure 5.24. Scatter plot of real outputs vs. predicted ones for different *Local Structures* (validation and test sets).

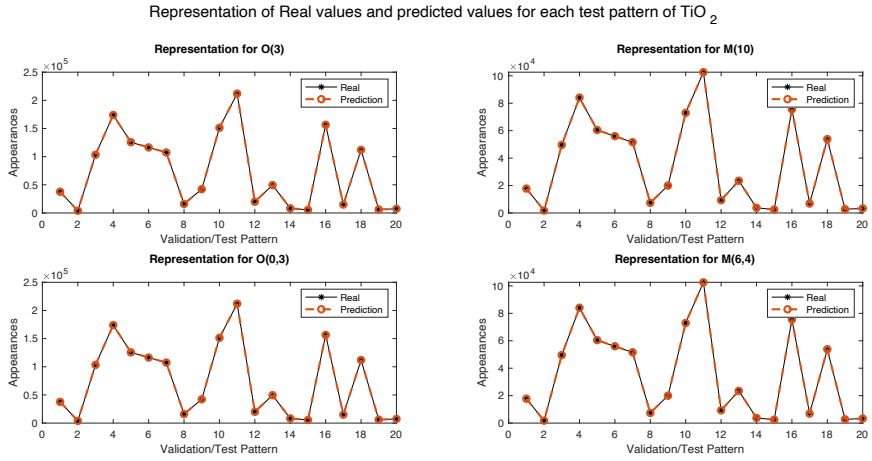


Figure 5.25. Representation of real and predicted values (validation and test sets) for each different *NanoFingerprint* (pattern).c

Then, the following two figures (Figure 5.26 and 5.27) represent the same as the two previous ones, respectively, but for *Local Structures* which do not present a perfect exponential growth or do not present any kind of exponential growth at all. They are the *Local Structures* presented on Figure 5.3.

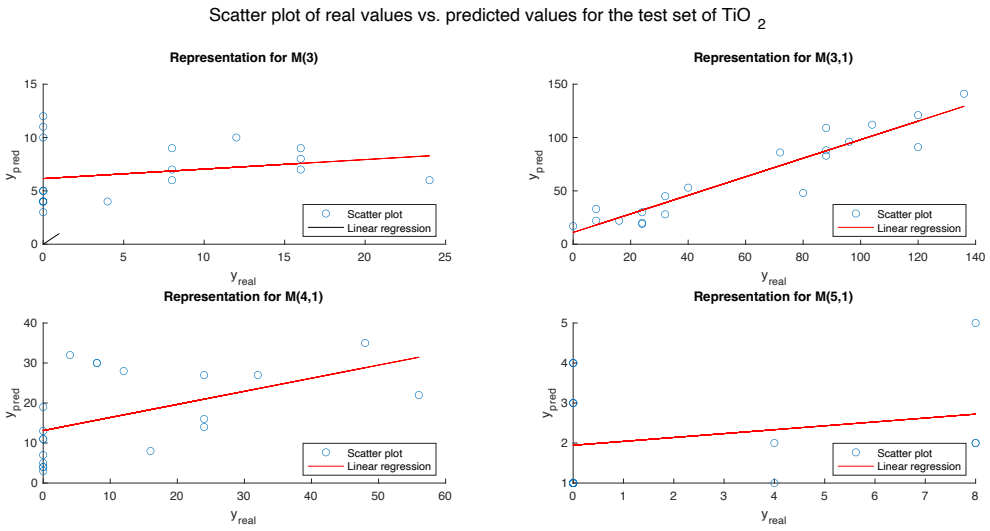


Figure 5.26. Scatter plot of real outputs vs. predicted ones for different *Local Structures* (validation and test sets).

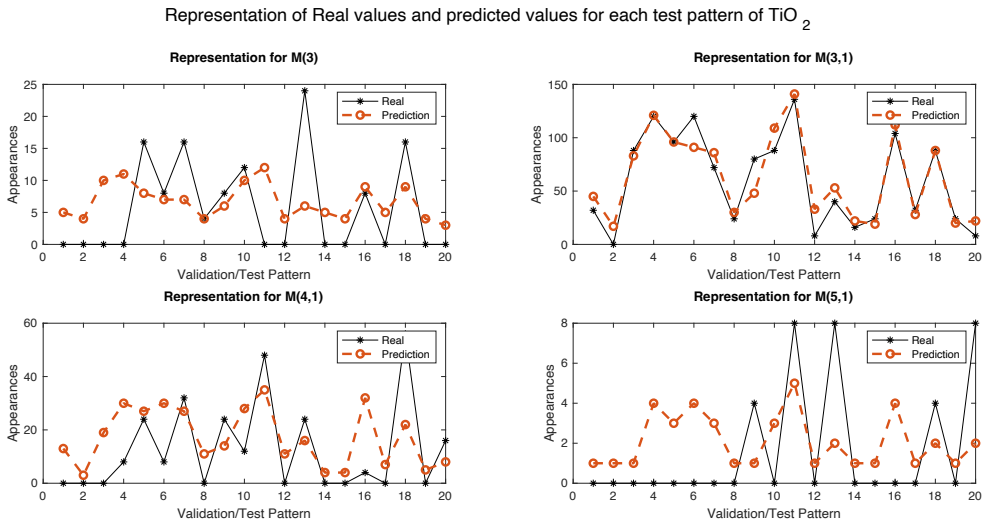


Figure 5.27. Representation of real and predicted values (validation and test sets) for each different *NanoFingerprint* (pattern).

Finally, one more representation is presented. Comparing to the approach in which the XYZ-file was not considered, the number of Oxygen and metal atoms are used as predictors on this approach, so they are not represented.

On Figure 5.28 it is depicted the representation of the apparition different *Local Structures*, from best to worst, in function of size (from 40 nm to 200 nm, since the first 20

NanoFingerprints were discarded). The predicted train outputs include the validation set (in green), and the predicted test outputs are represented on red. In blue, the real outputs are represented. Note that the predicted test outputs are mixed with the predicted train outputs, due to the initial shuffle of the *NanoFingerprints*.

The first one ($O(0,3)$) perfectly fits the curve; the result of the second one ($M(3,1)$), since the growth is exponential, although not perfect, is good as well, since it can serve to see the approximated evolution on the count. Thirdly, for $M(4,1)$, the evolution in function of size tends to grow, but with many ups and downs to high and low levels. With difference with the first approach, in which the XYZ files were not used for training (number of Oxygen and metal atoms), it is not a curve. Then, the addition of two predictors too the linear model makes the prediction approximate to the falls and rises of the real values, not being an exponential curve but exponential adaptive growth (in some cases it is good, while in others it goes to the contrary direction). The last one ($M(5,1)$) does not follow any exponential tendency, so the result is not acceptable. However, in this case, comparing again with the previous one, try to approximate to the maximums and minimums, with not many successes. Some of these *Local Structures* have been shown as well on Figures 5.24, 5.25, 5.26 and 5.27.

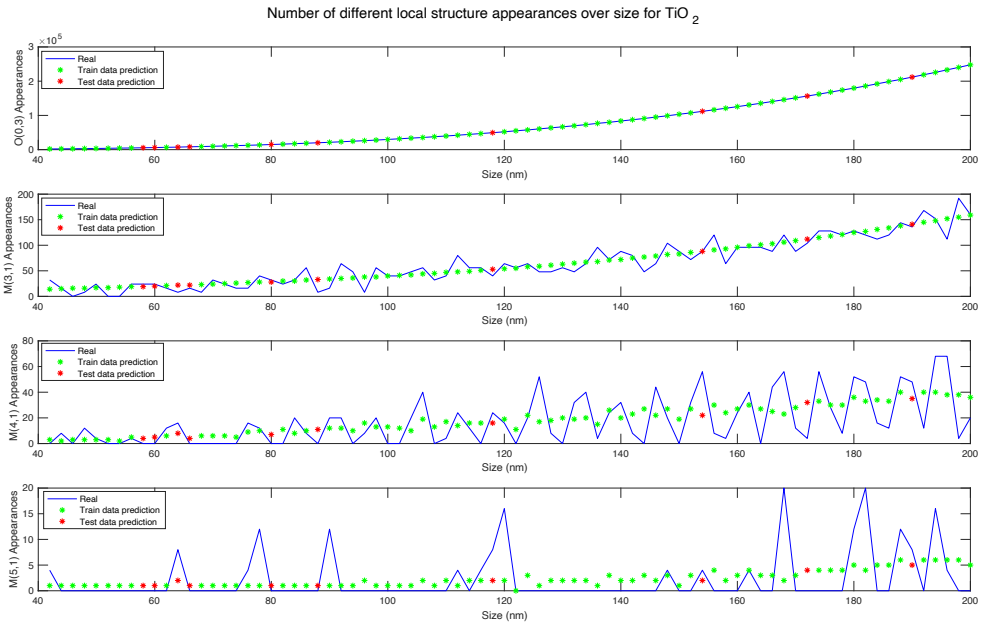


Figure 5.28. Representation of output appearances for different *Local Structures* in function of size.

The second model is quadratic and is described as:

$$y_{nm} = \beta_{0_m} + \beta_{1_m}X_{1n} + \beta_{2_m}X_{2n} + \beta_{3_m}X_{3n} + \beta_{4_m}X_{1n}X_{2n} + \beta_{5_m}X_{1n}X_{3n} + \beta_{6_m}X_{2n}X_{3n} + \beta_{7_m}X_{1n}^2 + \beta_{8_m}X_{2n}^2 + \beta_{9_m}X_{3n}^2$$

Where:

- m is the number of output variables (columns).
- n is the number of *NanoFingerprints*.
- β_{0_m} is the constant term (intercept) for each output column m of the *NanoFingerprint*.
- β_{1_m} is the first coefficient term for each output column m of the *NanoFingerprint*, corresponding to the size input predictor (X_{1n}).
- β_{2_m} is the second coefficient term for each output column m of the *NanoFingerprint*, corresponding to the number of Oxygen atoms input predictor (X_{2n}).
- β_{3_m} is the third coefficient term for each output column m of the *NanoFingerprint*, corresponding to the number of metal atoms input predictor (X_{3n}).
- β_{4_m} is the fourth coefficient term for each output column m of the *NanoFingerprint*, corresponding to the interaction between the size input predictor (X_{1n}) and number of Oxygen atoms input predictor (X_{2n}).
- β_{5_m} is the fifth coefficient term for each output column m of the *NanoFingerprint*, corresponding to the interaction between the size input predictor (X_{1n}) and number of Metal atoms input predictor (X_{3n}).
- β_{6_m} is the sixth coefficient term for each output column m of the *NanoFingerprint*, corresponding to the interaction between the number of Oxygen atoms input predictor (X_{2n}) and number of Metal atoms input predictor (X_{3n}).
- β_{7_m} is the seventh coefficient term for each output column m of the *NanoFingerprint*, corresponding to the squared size input predictor (X_{1n}).
- β_{8_m} is the eighth coefficient term for each output column m of the *NanoFingerprint*, corresponding to the squared number of Oxygen atoms input predictor (X_{2n}).

- β_{9_m} is the ninth coefficient term for each output column m of the *NanoFingerprint*, corresponding to the squared number of Metal atoms input predictor (X_{3n}).
- X_{1n} is the tenth predictor of each *NanoFingerprint* n.
- X_{2n} is the atomic number predictor of each *NanoFingerprint* n.
- y_{nm} is the output predicted value of each output column of the *NanoFingerprint*.

The returned model in *Wilkinson* notation by MATLAB, equivalent to the previous one, is:

$$y{\sim}1 + x1 * x2 + x1 * x3 + x2 * x3 + x1\text{^}2 + x2\text{^}2 + x3\text{^}2$$

There is a quadratic element, making it a non-linear regression. The program used on this case has been *nanofingerprint_generator_Ti.m* (changing just the $m_{predictor\_params}$ parameter to 3 with respect to the previous approach and the model type to 'quadratic'). The rest of parameters are the same as the previous model.

For the case of a k-Fold cross-validation with k=7 using the training set, the error metrics are shown on Table 5.12.

Error Metric	Result
$NMSE(y_{train\_real}, y_{train\_pred})$	0.1769440
$NMSE(y_{validation\_real}, y_{validation\_pred})$	0.199246
$MAE(y_{train\_real}, y_{train\_pred})$	0.324583
$MAE(y_{validation\_real}, y_{validation\_pred})$	0.432066
$NMSE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.000593
$MAE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.001421
Average correctness checking error	2.377551

Table 5.12. Error metrics for k-Fold (k=7) cross-validation on the second case XYZ approach for Atena dataset.

Note that the NMSE errors are lower than on the previous case (linear model) and the MAE for training outputs is lower whereas the validation one is higher. After reconstruction of the

NanoFingerprint, the NMSE and MAE are quite similar. Finally, the average correctness checking error is slightly higher on this case.

On the second place, a *Leave-One-Out* (k=70) cross-validation has been performed. On Table 5.13 the results are shown, which are very similar to the previous 7-Fold cross-validation.

Error Metric	Result
$NMSE(y_{train\_real}, y_{train\_pred})$	0.178791
$NMSE(y_{validation\_real}, y_{validation\_pred})$	0.198142
$MAE(y_{train\_real}, y_{train\_pred})$	0.328488
$MAE(y_{validation\_real}, y_{validation\_pred})$	0.432469
$NMSE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.000593
$MAE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.001409
Average correctness checking error	2.378367

Table 5.13. Error metrics for Leave-One-Out (k=70) cross-validation on the second case XYZ approach for Atena dataset.

Finally, it has been used one of the cross-validation fitted models with 60 *NanoFingerprints* for training, and then it has been tested joining the validation (set corresponding with the training set) and test sets (20 *NanoFingerprints*), in order to observe the performance with completely new data. On Table 5.14 the results on the validation and test part are shown (NMSE, MAE and average correctness checking error). Note that the results are very close to the average ones obtained on the 7-Fold cross-validation for the validation set.

Error Metric	Result
$NMSE(y_{validation\_real}, y_{validation\_pred})$	0.204082
$MAE(y_{validation\_real}, y_{validation\_pred})$	0.420951
$NMSE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.000463
$MAE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.000864
Average correctness checking error	2.3375

Table 5.14. Validation and test set error metrics for the for the second case XYZ approach for Atena dataset.

Only the plots for Local Structures (Figures 5.29 and 5.30) which do not present a perfect exponential growth or any at all are presented, because the good ones were the same comparing to the previous case (linear model).

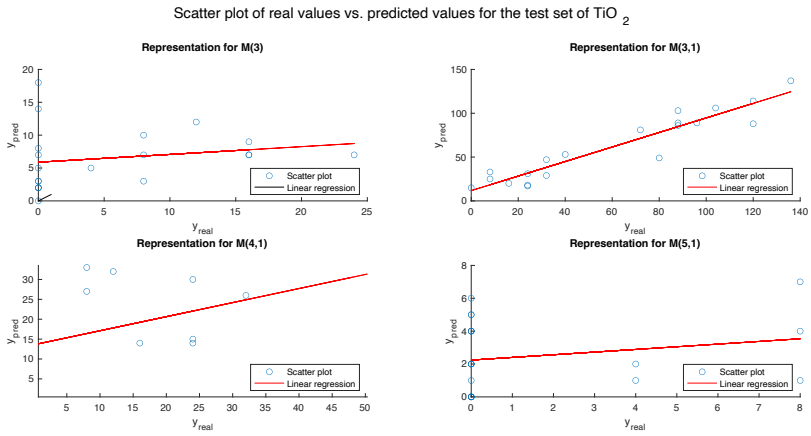


Figure 5.29. Scatter plot of real outputs vs. predicted ones for different *Local Structures* (validation and test sets).

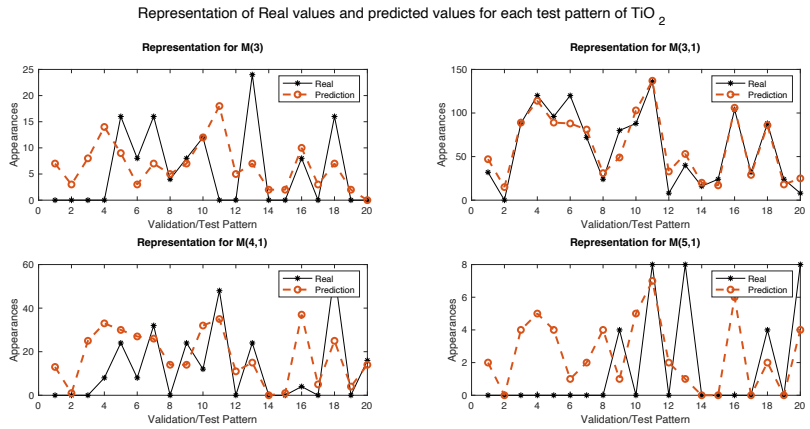


Figure 5.30. Representation of real and predicted values (validation and test sets) for each different *NanoFingerprint* (pattern).

To numerically measure the changes, the NMSE(y_{test_real}, y_{test_pred}) and MAE(y_{test_real}, y_{test_pred}) for the four last different local structures have been calculated for both cases, and additionally with the first approach, so they can be compared. The results are presented on Table 5.15.

Error Metric	1st approach	2nd approach (1st case)	2nd approach (2nd case)
NMSE(y_{test_real}, y_{test_pred}) - M(3)	0.6249	0.6219	0.5906
NMSE(y_{test_real}, y_{test_pred}) - M(3,1)	0.1284	0.1308	0.1355
NMSE(y_{test_real}, y_{test_pred}) - M(4,1)	0.6106	0.6091	0.5520
NMSE(y_{test_real}, y_{test_pred}) - M(5,1)	0.8547	0.8539	0.5922
MAE(y_{test_real}, y_{test_pred}) - M(3)	0.8615	0.8647	0.8779
MAE(y_{test_real}, y_{test_pred}) - M(3,1)	0.1764	0.1756	0.1772
MAE(y_{test_real}, y_{test_pred}) - M(4,1)	0.7079	0.7110	0.6851
MAE(y_{test_real}, y_{test_pred}) - M(5,1)	1.1818	1.1905	1.04

Table 5.15. Validation and test set error metrics for all approaches and cases for the Atena dataset for Local Structures M(3), M(3,1), M(4,1) and M(5,1).

Note that, in general, the 2nd approach (2nd case) obtains the best results, outperforming the others specially on the NMSE for the Local Structure M(5,1).

Finally, the last plot is on Figure 5.31, the representation of the apparition different *Local Structures*, from best to worst predictions, in function of size. On this case, visually looking at the plot, the predicted values tend to be closer to the real values. On the second subplot (*M(3,1)*), for instance, it is not a perfect curve, comparing to the previous case. Additionally, there are many test points that perfectly fit the real ones, better than on the previous case (see the first four test points on *M(4,1)*).

To numerically measure the changes, the NMSE($nanofingerprint_{real}$, $nanofingerprint_{prediction}$) and MAE($nanofingerprint_{real}$, $nanofingerprint_{prediction}$) for the four different local structures have been calculated for both cases, and additionally with the first approach, so they can be compared. The results are presented on Table 5.16.

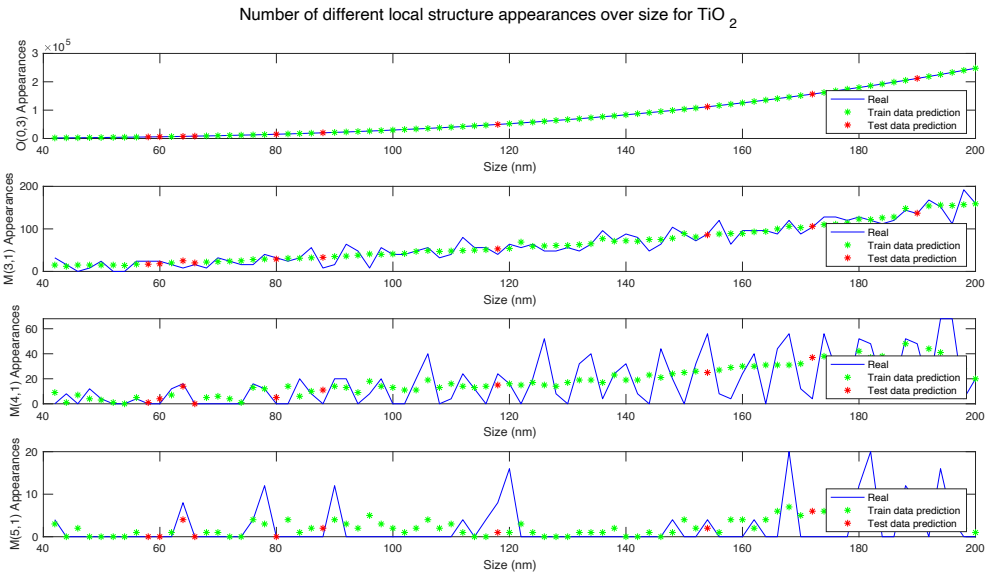


Figure 5.31. Representation of output appearances for different *Local Structures* in function of size.

Error Metric	1st approach	2nd approach (1st case)	2nd approach (2nd case)
NMSE - O(0,3)	0.032	1.0299e-4	2.9274e-06
NMSE - M(3,1)	0.1077	0.1086	0.1090
NMSE - M(4,1)	0.4735	0.4753	0.4382
NMSE - M(5,1)	0.8678	0.8563	0.6246
MAE - O(0,3)	0.0282	5.7764e-4	2.3957e-4
MAE - M(3,1)	0.1836	0.1814	0.1751
MAE - M(4,1)	0.7035	0.6928	0.6431
MAE - M(5,1)	1.3548	1.3757	1.1287

Table 5.16. NanoFingerprint error metrics for all approaches and cases for the Atena dataset for Local Structures O(0,3), M(3,1), M(4,1) and M(5,1).

Note that in general the best results metrics are achieved for the 2nd approach (2nd case). However, note that visually it looked like for M(3,1) the results were much better than previous cases, because it looked like it was adapting better to rough changes, but, in fact, the other two are really close or equal to it if we look at the metrics (equal for NMSE, different for MAE).

Concluding, it can be seen that both approaches and cases (linear and non-linear models, using and non-using the XYZ files) are very similar and equally suitable.

5.7.2. Other datasets

The aim of this last model generation is to predict the elements of the other datasets *NanoFingerprints* using the size, the number of Oxygen and Metal atoms and the atomic number (from the element different from Oxygen) of each nanoparticle as predictor variables.

Two models have been tested as well, using the same predictors, to observe if both are suitable and conclude which one is the most appropriate one. The first one is linear, while de second one is non-linear (quadratic).

The first one described as:

$$y_{nm} = \beta_{0_m} + \beta_{1_m} X_{1n} + \beta_{2_m} X_{2n} + \beta_{3_m} X_{3n} + \beta_{4_m} X_{4n}$$

Where:

- m is the number of output variables (columns).
- n is the number of *NanoFingerprints*.
- β_{0_m} is the constant term (intercept) for each output column m of the *NanoFingerprint*.
- β_{1_m} is the first coefficient term for each output column m of the *NanoFingerprint*, corresponding to the size input predictor (X_{1n}).
- β_{2_m} is the fourth coefficient term for each output column m of the *NanoFingerprint*, corresponding to the atomic number input predictor (X_{2n}).
- β_3 is the second coefficient term for each output column m of the *NanoFingerprint*, corresponding to the number of Oxygen atoms input predictor (X_{3n}).
- β_{4_m} is the third coefficient term for each output column m of the *NanoFingerprint*, corresponding to the number of Metal atoms input predictor (X_{4n}).
- X_{1n} is the size predictor of each *NanoFingerprint* n.
- X_{2n} is the atomic number predictor of each *NanoFingerprint* n.

- X_{3n} is the number of Oxygen atoms predictor of each *NanoFingerprint* n.
- X_{4n} is the number of Metal atoms predictor of each *NanoFingerprint* n.
- y_{nm} is the output predicted value of each output column of the *NanoFingerprint*.

The returned model in *Wilkinson* notation by MATLAB, equivalent to the previous one, is:

$$y \sim 1 + x1 + x2 + x3 + x4$$

The program used on this case has been *nanofingerprint_generator_other*. The variable $m_{predictor\_params}$ has been set to 4. The function *compute_regression_other.m* is used to generate the model, predict, and make the error calculations.

Having 60 *NanoFingerprints*, the first 14 (n_{start}-1) have been discarded and 44 have been used for training (train set). The 2 *NanoFingerprint* left have been used for testing (test set). The sample has been reduced as well as on 5.6.2. Other datasets.

Firstly, with the training set, a k-Fold cross-validation with k=6 has been performed (on each fold, 37 observations are used for training and 7 for validation). On Table 5.17 the error metrics from the cross-validation are shown.

Error Metric	Result
$NMSE(y_{train\_real}, y_{train\_pred})$	0.444544
$NMSE(y_{validation\_real}, y_{validation\_pred})$	0.480873
$MAE(y_{train\_real}, y_{train\_pred})$	1.450425
$MAE(y_{validation\_real}, y_{validation\_pred})$	1.976929
$NMSE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.006376
$MAE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.024889
Average correctness checking error	45.416667

Table 5.17. Error metrics for k-Fold (k=6) cross-validation on the first case XYZ approach for 'other' datasets.

Note that, again, in general the errors are much higher than on the Atena dataset case. The MAE is higher than 1, meaning that the predicted and real values are much greater in ones

compared to the others. The NMSE error show that there is also a big difference in most of the values (predicted or real are half of the other ones in average).

After reconstructing the *NanoFingerprint*, note that the errors are very low. However, the MAE is 0.02, which is starting to be considerable (about 2%), considering that there are many 0 columns.

Finally, the average correctness checking error is also huge, 45.41, describing that the generated *NanoFingerprints* do not follow the correct structure.

On the second place, a *Leave-One-Out* (k=44) cross-validation has been performed. On Table 5.18 the results are shown. On this new case, the results are very similar to the previous one, being slightly higher than on the previous case.

Error Metric	Result
$NMSE(y_{train\_real}, y_{train\_pred})$	0.459318
$NMSE(y_{validation\_real}, y_{validation\_pred})$	0.483824
$MAE(y_{train\_real}, y_{train\_pred})$	1.526730
$MAE(y_{validation\_real}, y_{validation\_pred})$	2.036986
$NMSE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.006499
$MAE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.023988
Average correctness checking error	47.047521

Table 5.18. Error metrics for Leave-One-Out (k=44) cross-validation on the first case XYZ approach for 'other' datasets.

Finally, it has been used one of the cross-validation fitted models with 37 *NanoFingerprints* for training, and then it has been tested joining the validation (set corresponding with the training set) and test sets (9 *NanoFingerprints*), in order to observe the performance with completely new data, showing some representations of the results as well.

On Table 5.19 the results on the validation and test part are shown (NMSE, MAE and average correctness checking error). Note that the results are very close to the average ones obtained

on the 6-Fold cross-validation for the validation set. Additionally, results are lower comparing with the first approach on 'other' datasets.

Error Metric	Result
$NMSE(y_{validation\_real}, y_{validation\_pred})$	0.464134
$MAE(y_{validation\_real}, y_{validation\_pred})$	1.756166
$NMSE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.006122
$MAE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.021823
Average correctness checking error	39.478261

Table 5.19. Validation and test set error metrics for the for the first case XYZ approach for 'other' datasets.

On Figure 5.32 a scatter plot representing the real outputs against the predicted ones for different *Local Structures* is depicted, for the validation and test outputs (number of appearances). The first 7 values correspond to the validation set, whereas the last 2 ones belong to the test set. The closer to the regression line, the better; thus, note that for these *Local Structures*, the results are quite good.

On Figure 5.33 it is represented, for the same *Local Structures* from Figure 5.32, the real and predicted outputs (number of appearances) for each validation and test *NanoFingerprint* (pattern). As well as on Figure 5.32, the first 7 ones correspond to the validation set, whereas the last 2 ones correspond to the test set. It can be seen that the predicted results are close to the real ones, despite differing in some thousands of counts (for instance, on the fifth pattern), but it guesses the tendency. Additionally, the predicted values tend to approximate better than on the first approach, visually.

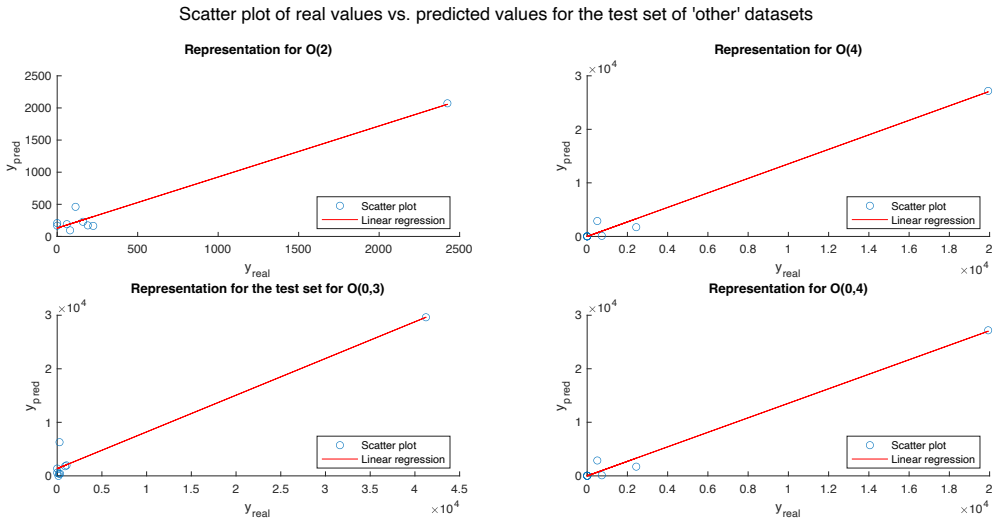


Figure 5.32. Scatter plot of real outputs vs. predicted ones for different *Local Structures* (validation and test sets).

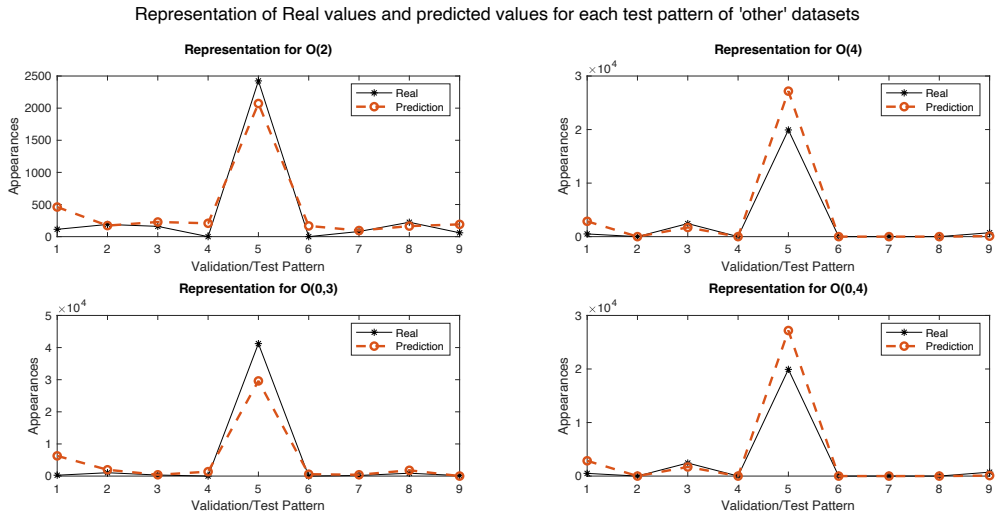


Figure 5.33. Representation of real and predicted values (validation and test sets) for each different *NanoFingerprint* (pattern).

Then, the following two figures (Figure 5.34 and 5.35) represent the same as the two previous ones, respectively, but for Local Structures which do not present any kind of exponential growth. The right representation does not even guess the tendency, whereas the left one, despite guessing it, differs in the order of 10^4 counts, as on the first approach.

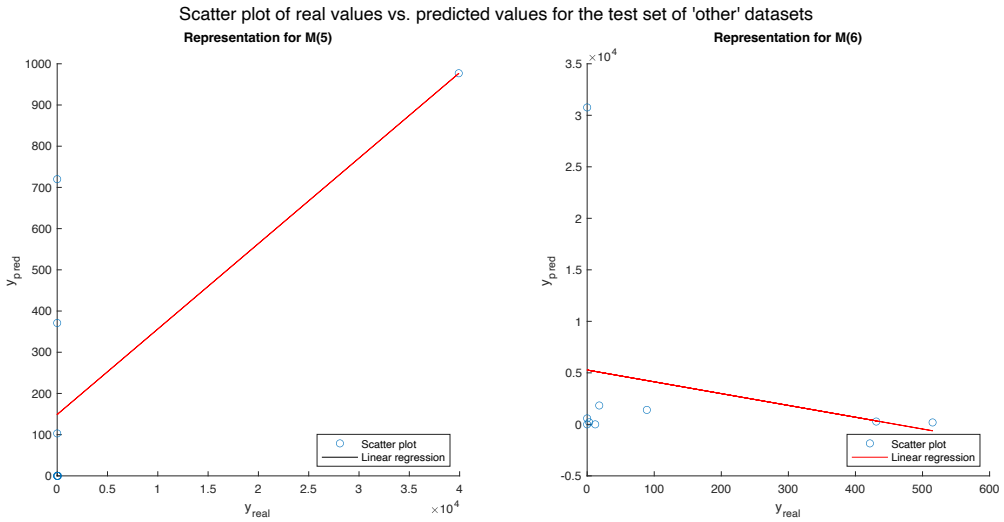


Figure 5.34. Scatter plot of real outputs vs. predicted ones for different *Local Structures* (validation and test sets).

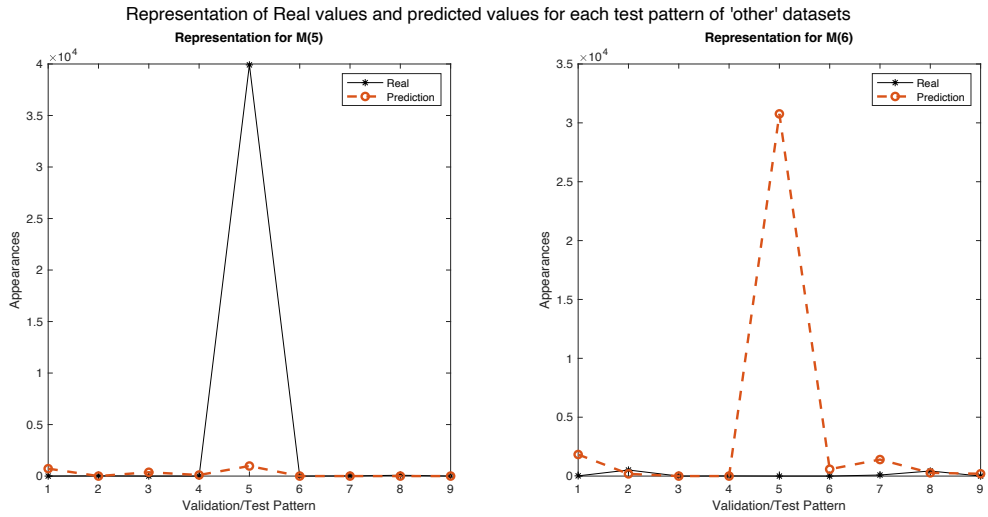


Figure 5.35. Representation of real and predicted values (validation and test sets) for each different *NanoFingerprint* (pattern).

Finally, two more representations are presented (the evolution of number of Oxygen and metal atoms is not presented since they are used as predictors). On Figure 5.36, the representation is for different *Local Structures*. Predictions are not very good but try to approximate to the real ones.

Figure 5.37 is the same representations but with a window of sizes up to 80 nm.

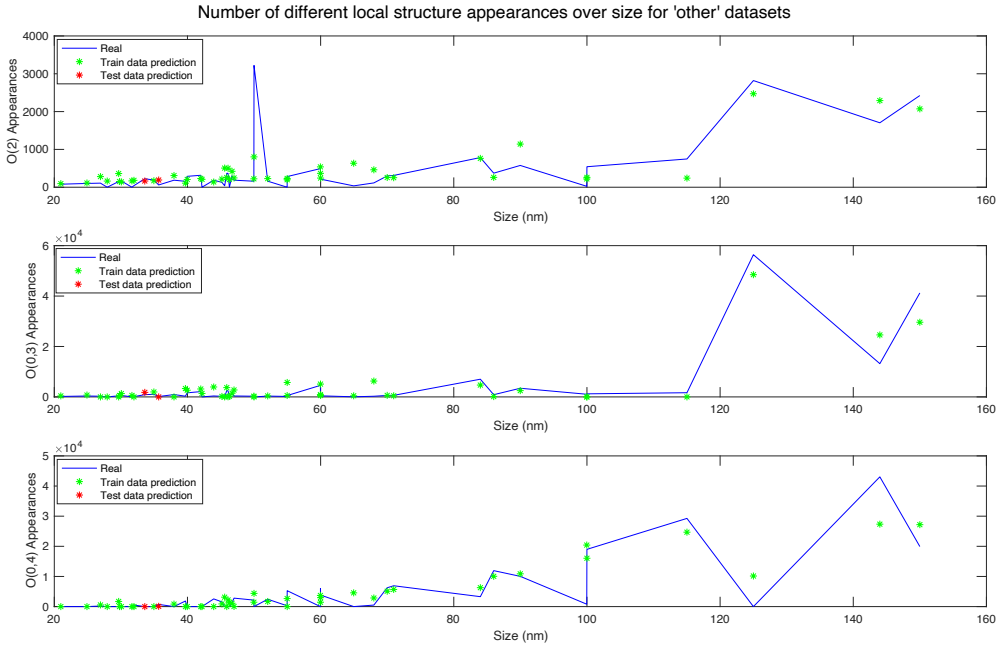


Figure 5.36. Representation of output appearances for different *Local Structures* in function of size.

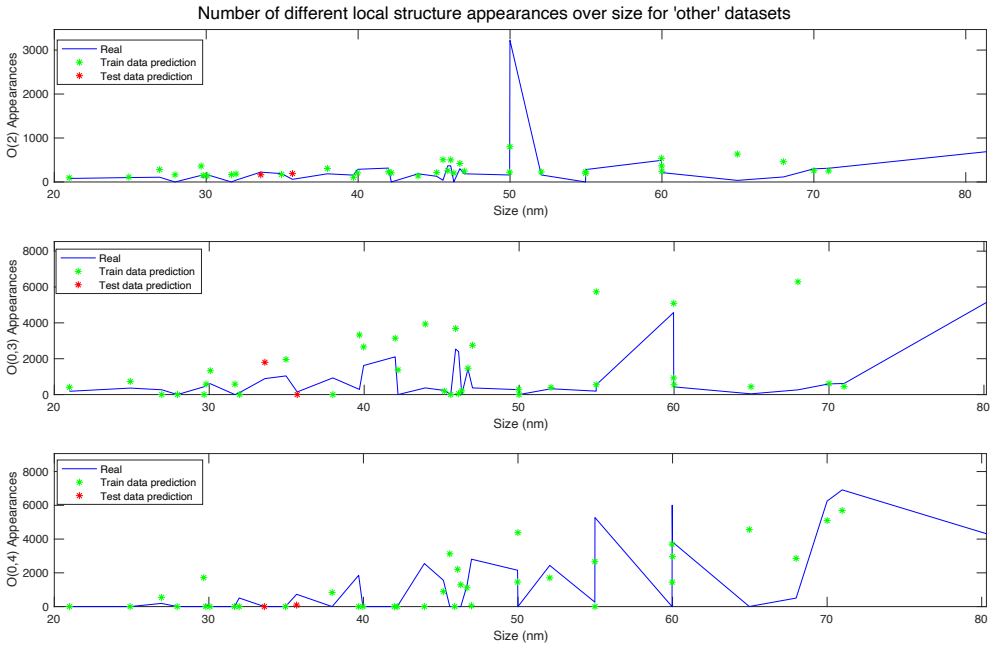


Figure 5.37. Representation of output appearances for different *Local Structures* in function of size (up to 80 nm).

The second model is quadratic and is described as:

$$y_{nm} = \beta_{0_m} + \beta_{1_m}X_{1n} + \beta_{2_m}X_{2n} + \beta_{3_m}X_{3n} + \beta_{4_m}X_{4n} + \beta_{5_m}X_{1n}X_{2n} + \beta_{6_m}X_{1n}X_{3n} + \beta_{7_m}X_{1n}X_{4n} + \beta_{8_m}X_{2n}X_{3n} + \beta_{9_m}X_{2n}X_{4n} + \beta_{10_m}X_{3n}X_{4n} + \beta_{11_m}X_{1n}^2 + \beta_{12_m}X_{2n}^2 + \beta_{13_m}X_{3n}^2 + \beta_{14_m}X_{4n}^2$$

Where:

- m is the number of output variables (columns).
- n is the number of *NanoFingerprints*.
- β_{0_m} is the constant term (intercept) for each output column m of the *NanoFingerprint*.
- β_{1_m} is the first coefficient term for each output column m of the *NanoFingerprint*, corresponding to the size input predictor (X_{1n}).
- β_{2_m} is the second coefficient term for each output column m of the *NanoFingerprint*, corresponding to the atomic number input predictor (X_{2n}).
- β_{3_m} is the third coefficient term for each output column m of the *NanoFingerprint*, corresponding to the number of Oxygen atoms input predictor (X_{3n}).
- β_{4_m} is the fourth coefficient term for each output column m of the *NanoFingerprint*, corresponding to the number of metal atoms input predictor (X_{4n}).
- β_{5_m} is the fifth coefficient term for each output column m of the *NanoFingerprint*, corresponding to the interaction between the size input predictor (X_{1n}) and atomic number input predictor (X_{2n}).
- β_{6_m} is the sixth coefficient term for each output column m of the *NanoFingerprint*, corresponding to the interaction between the size input predictor (X_{1n}) and number of Oxygen atoms input predictor (X_{3n}).
- β_{7_m} is the seventh coefficient term for each output column m of the *NanoFingerprint*, corresponding to the interaction between the size input predictor (X_{1n}) and number of Metal atoms input predictor (X_{4n}).

- β_{8_m} is the eighth coefficient term for each output column m of the *NanoFingerprint*, corresponding to the interaction between the atomic number input predictor (X_{2n}) and number of Oxygen atoms input predictor (X_{3n}).

- β_{9_m} is the ninth coefficient term for each output column m of the *NanoFingerprint*, corresponding to the interaction between the atomic number input predictor (X_{2n}) and number of Metal atoms input predictor (X_{4n}).

- β_{10_m} is the tenth coefficient term for each output column m of the *NanoFingerprint*, corresponding to the interaction between the number of Oxygen atoms input predictor (X_{3n}) and number of Metal atoms input predictor (X_{4n}).

- β_{11_m} is the eleventh coefficient term for each output column m of the *NanoFingerprint*, corresponding to the squared size input predictor (X_{1n}).

- β_{12_m} is the twelfth coefficient term for each output column m of the *NanoFingerprint*, corresponding to the squared atomic number input predictor (X_{2n}).

- β_{13_m} is the thirteenth coefficient term for each output column m of the *NanoFingerprint*, corresponding to the squared number of Oxygen atoms input predictor (X_{3n}).

- β_{14_m} is the fourteenth coefficient term for each output column m of the *NanoFingerprint*, corresponding to the squared number of Metal atoms input predictor (X_{4n}).

- X_{1n} is the tenth predictor of each *NanoFingerprint* n.

- X_{2n} is the atomic number predictor of each *NanoFingerprint* n.

- X_{3n} is the number of Oxygen atoms predictor of each *NanoFingerprint* n.

- X_{4n} is the number of Metal atoms predictor of each *NanoFingerprint* n.

- y_{nm} is the output predicted value of each output column of the *NanoFingerprint*.

The returned model in *Wilkinson* notation by MATLAB, equivalent to the previous one, is:

y ~ 1 + x1*x2 + x1*x3 + x1*x4 + x2*x3 + x2*x4 + x3*x4 + x1^2 + x2^2 + x3^2 + x4^2

The program used on this case has been *nanofingerprint_generator_other*. The variable $m_{predictor\_params}$ has been set to 4, and using a 'quadratic' moodel. The function *compute_regression_other.m* is used to generate the model, predict, and make the error calculations.

Having 60 *NanoFingerprints*, the first 14 (n_{start}-1) have been discarded and 44 have been used for training (train set). The 2 *NanoFingerprint* left have been used for testing (test set). The sample has been reduced as well as on 5.6.2. Other datasets.

Firstly, with the training set, a k-Fold cross-validation with k=6 has been performed (on each fold, 37 observations are used for training and 7 for validation). On Table 5.20 the error metrics from the cross-validation are shown.

Error Metric	Result
$NMSE(y_{train\_real}, y_{train\_pred})$	0.330066
$NMSE(y_{validation\_real}, y_{validation\_pred})$	0.432638
$MAE(y_{train\_real}, y_{train\_pred})$	1.179206
$MAE(y_{validation\_real}, y_{validation\_pred})$	1.567839
$NMSE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.005382
$MAE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.018939
Average correctness checking error	43.542313

Table 5.20. Error metrics for k-Fold (k=6) cross-validation on the second case XYZ approach for 'other' datasets.

On this case, the results are considerably better comparing with the first case. However, errors are still considerably high.

On the second place, a *Leave-One-Out* (k=44) cross-validation has been performed. On Table 5.21 the results are shown. On this new case, the results are very similar to the previous one, being slightly higher than on the previous case, but lower than the previous approach and case.

Error Metric	Result
$NMSE(y_{train\_real}, y_{train\_pred})$	0.360349
$NMSE(y_{validation\_real}, y_{validation\_pred})$	0.448399
$MAE(y_{train\_real}, y_{train\_pred})$	1.3667907
$MAE(y_{validation\_real}, y_{validation\_pred})$	1.776585
$NMSE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.005563
$MAE(nanofingerprint_{real}, nanofingerprint_{prediction})$	0.020393
Average correctness checking error	45.859283

Table 5.21. Error metrics for Leave-One-Out (k=44) cross-validation on the second case XYZ approach for 'other' datasets.

Finally, it has been used one of the cross-validation fitted models with 37 *NanoFingerprints* for training, and then it has been tested joining the validation (set corresponding with the training set) and test sets (9 *NanoFingerprints*), in order to observe the performance with completely new data, showing some representations of the results as well.

On Table 5.22 the results on the validation and test part are shown (NMSE, MAE and average correctness checking error). Note that the results are very close to the average ones obtained on the 6-Fold cross-validation for the validation set. Additionally, results are lower comparing with the first approach on 'other' datasets and the first case of the second approach as well.

Error Metric	Result
$NMSE(y_{validation\_real}, y_{validation\_pred})$	0.320134
$MAE(y_{validation\_real}, y_{validation\_pred})$	0.426204
$NMSE(nanofingerprint_{real}, nanofingerprint_{prediction})$	1.157333
$MAE(nanofingerprint_{real}, nanofingerprint_{prediction})$	1.637282
Average correctness checking error	38.475918

Table 5.22. Validation and test set error metrics for the for the second case XYZ approach for 'other' datasets.

On Figure 5.38 a scatter plot representing the real outputs against the predicted ones for different *Local Structures* is depicted, for the validation and test outputs (number of appearances). The first 7 values correspond to the validation set, whereas the last 2 ones belong to the test set.

On Figure 5.39 it is represented, for the same *Local Structures* from Figure 5.38, the real and predicted outputs (number of appearances) for each validation and test *NanoFingerprint* (pattern). As well as on Figure 5.38, the first 7 ones correspond to the validation set, whereas the last 2 ones correspond to the test set. It can be seen that the predicted results are close to the real ones. Additionally, it look like the predicted values tend to approximate better than on the first approach and on the second approach first case.

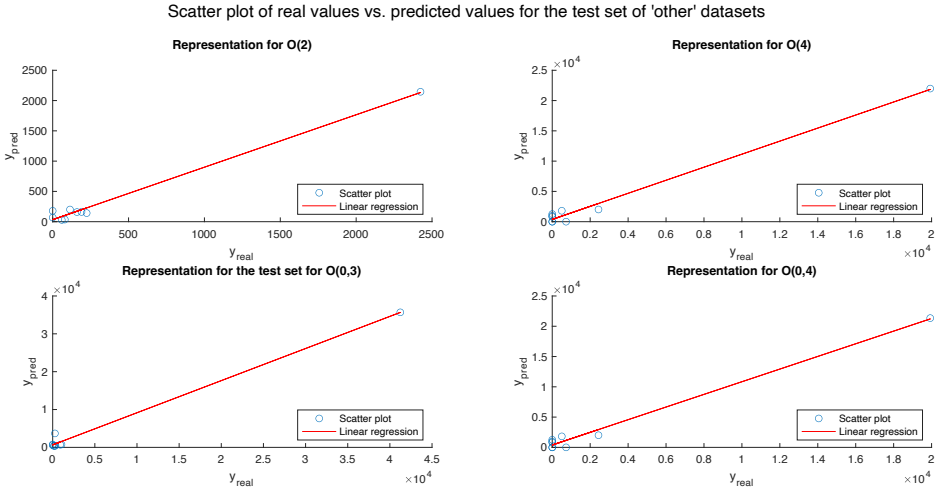


Figure 5.38. Scatter plot of real outputs vs. predicted ones for different *Local Structures* (validation and test sets).

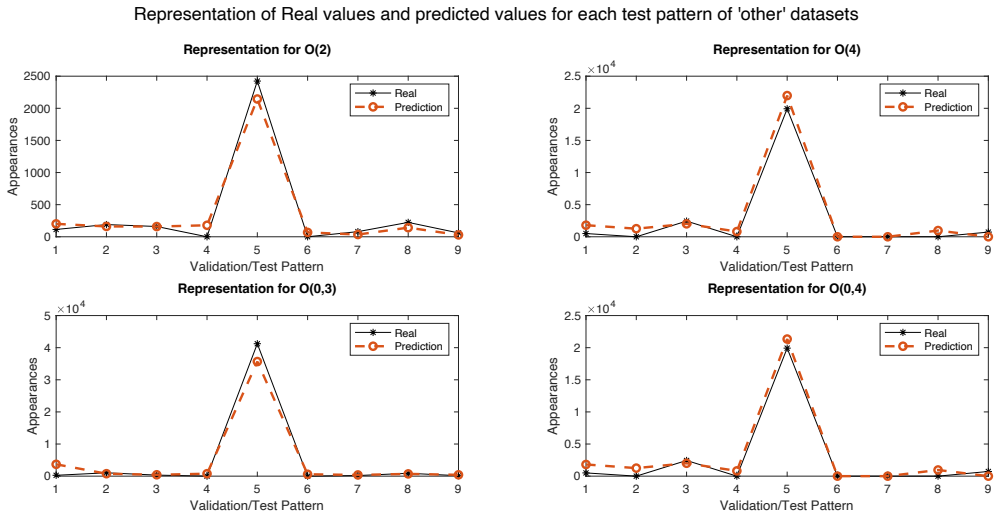


Figure 5.39. Representation of real and predicted values (validation and test sets) for each different *NanoFingerprint* (pattern).

To numerically measure the changes, the NMSE(y_{test_real}, y_{test_pred}) and MAE(y_{test_real}, y_{test_pred}) for these four different local structures have been calculated for both cases, and additionally with the first approach, so they can be compared. Note that on this case the comparisons are made for the 'good' cases. The results are presented on Table 5.23.

Error Metric	1st approach	2nd approach (1st case)	2nd approach (2nd case)
NMSE(y_{test_real}, y_{test_pred}) - O(2)	0.4686	0.3614	0.3268
NMSE(y_{test_real}, y_{test_pred}) - O(4)	0.6635	0.1775	0.2064
NMSE(y_{test_real}, y_{test_pred}) - O(0,3)	0.6120	0.5342	0.4139
NMSE(y_{test_real}, y_{test_pred}) - O(0,4)	0.6633	0.1790	0.2065
MAE(y_{test_real}, y_{test_pred}) - O(2)	0.6224	0.3635	0.2568
MAE(y_{test_real}, y_{test_pred}) - O(4)	0.5063	0.3444	0.2616
MAE(y_{test_real}, y_{test_pred}) - O(0,3)	0.3794	0.5154	0.2615
MAE(y_{test_real}, y_{test_pred}) - O(0,4)	0.5063	0.3451	0.2455

Table 5.23. Validation and test set error metrics for all approaches and cases for the 'other' datasets for Local Structures O(2), O(4), O(0,3) and O(0,4).

As it can be seen from these results, the second case of the second approach outperforms the rest. There is a special case, for O(4) and for O(0,4), in which the first case of the second approach outperforms the second case for the NMSE measure. This is because the prediction approximate better to the predicted values. However, on the MAE, the first case of the second approach does not outperform the second one because this measure does not normalize over the maximum of the two comparisons, and the greater values have more influence. For instance, for pattern 5, the predicted and real values are much closer, but the rest of test/validation patterns are not as good predicted as on the first case of the second approach.

Then, the following two figures (Figure 5.40 and 5.41) represent the same as the two previous ones, respectively, but for Local Structures which do not present any kind of exponential growth. The right representation does not even guess the tendency, whereas the left one, despite guessing it, it approximates much more to the real values comparing to the first approach and second approach first case.

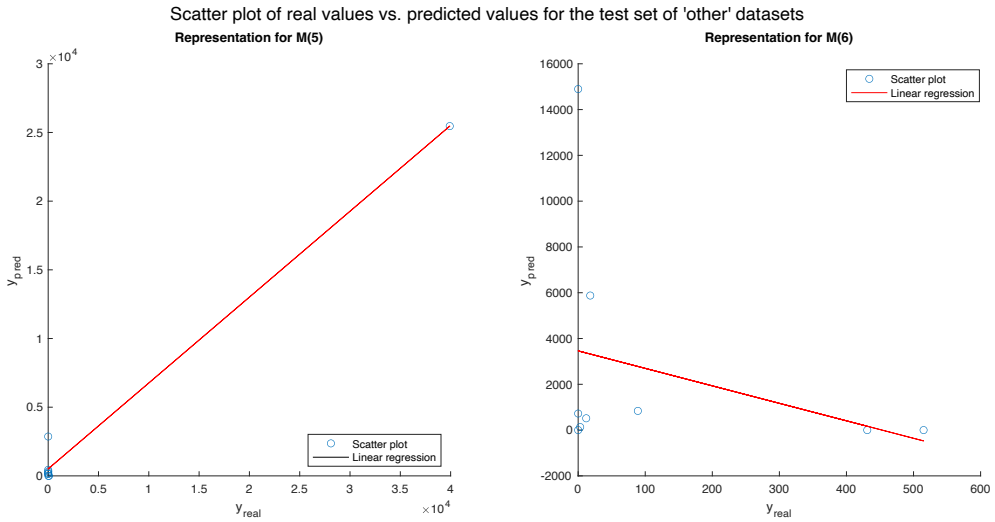


Figure 5.40. Scatter plot of real outputs vs. predicted ones for different *Local Structures* (validation and test sets).

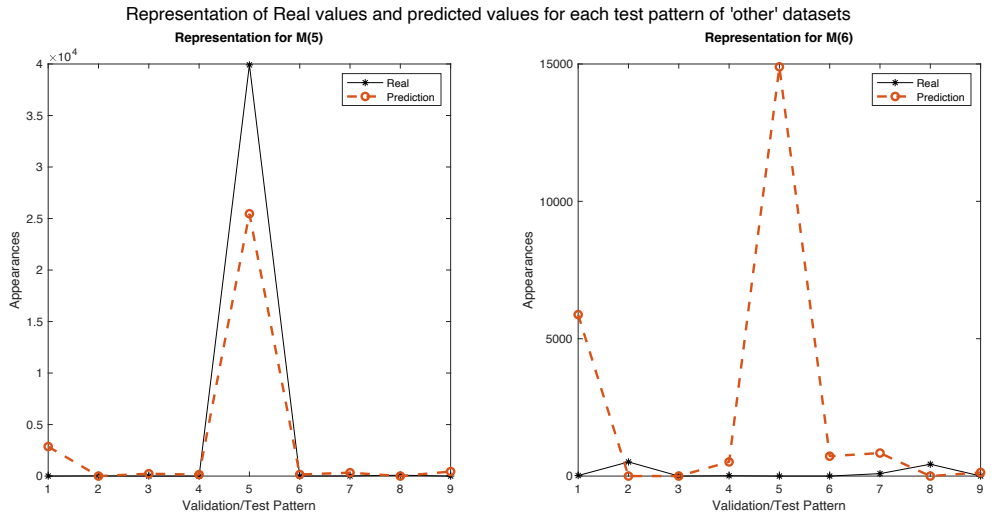


Figure 5.41. Representation of real and predicted values (validation and test sets) for each different *NanoFingerprint* (pattern).

Finally, two more representations are presented. On Figure 5.42, the representation is for different *Local Structures*. Note that predictions, especially for large counts (apparitions) fit very good to the real values, better than on the first approach and first case of second approach for 'other' datasets.

Figure 5.43 is the same representations but with a window of sizes up to 80 nm. For low counts, prediction also approximate to the real values.

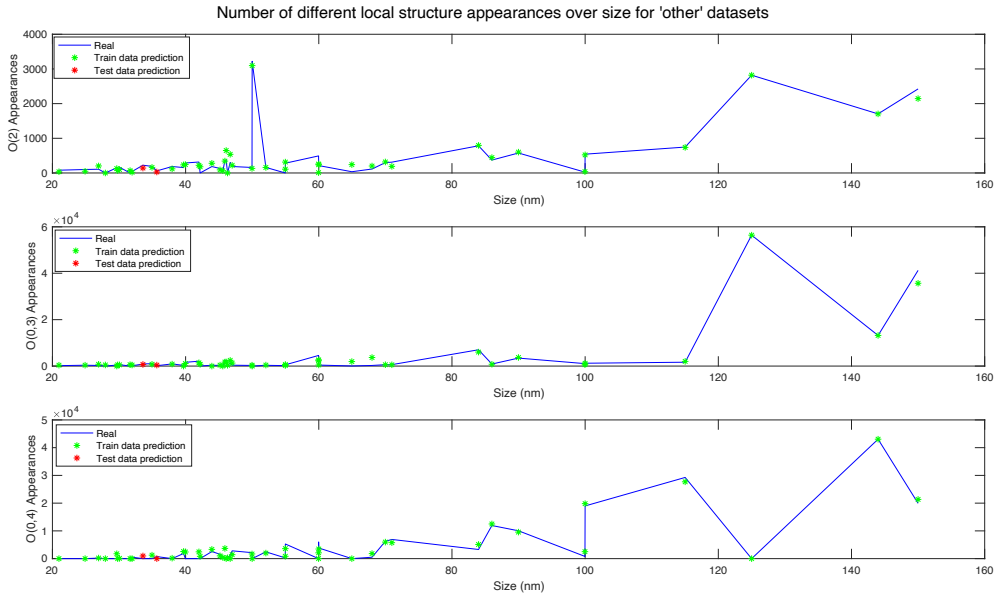


Figure 5.42. Representation of output appearances for different *Local Structures* in function of size.

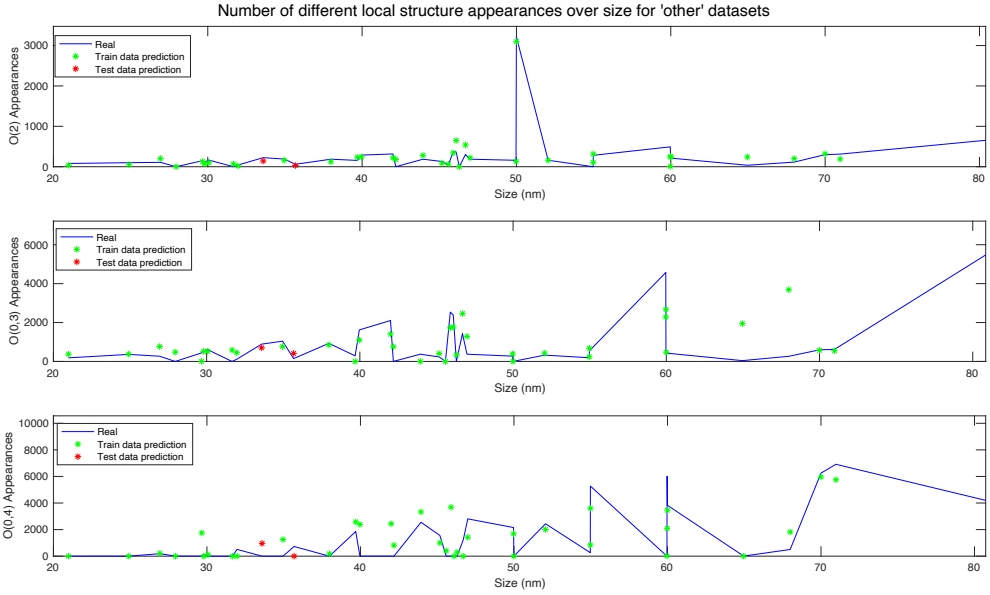


Figure 5.43. Representation of output appearances for different *Local Structures* in function of size (up to 80 nm).

To numerically measure the changes, the NMSE($nanofingerprint_{real}$, $nanofingerprint_{prediction}$) and MAE($nanofingerprint_{real}$, $nanofingerprint_{prediction}$) for the three different local structures have been calculated for both cases, and additionally with the first approach, so they can be compared. The results are presented on Table 5.24.

Error Metric	1st approach	2nd approach (1st case)	2nd approach (2nd case)
NMSE - O(2)	0.4467	0.2843	0.1786
NMSE - O(0,3)	0.7097	0.4967	0.3087
NMSE - O(0,4)	0.5132	0.4601	0.4727
MAE - O(2)	0.6567	0.4952	0.1812
MAE - O(0,3)	0.8229	0.5029	0.1843
MAE - O(0,4)	0.5406	0.5789	0.2031

Table 5.24. NanoFingerprint error metrics for all approaches and cases for the 'other' datasets for Local Structures O(2), O(0,3) and O(0,4).

Again, the best results are achieved for the second case of the second approach, happening the same as on Table 5.23: for O(0,4), the NMSE is better on the first case of the second approach than on the second case.

Note, additionally, that on both Table 5.23 and 5.24 the MAE is higher than NMSE, in some cases, for the 2n approach (2nd Case).

Concluding, it can be seen that second approach is better than the first one; however, they are not good enough to obtain valid results.

5.8. Conclusions

The aim of this chapter has been to artificially generate *NanoFingerprints* using two approaches, with and without the XYZ files, and with different datasets. During the presentation of the models, the results obtained have been commented and compared between them.

Using a regression model, the predicted results are not expected to be excellent but to approximate the real ones. In fact, not exactly having the relationship of the nanoparticle elements (*Local Structure* count) can contribute in deducing some of the presented properties of the nanoparticle.

To summarize the results, some aspects will be commented.

On the first place, regarding to the *Atena* dataset, the results obtained are quite good. The key aspect has been having a representative data which can be described, at most, using a regression (exponential growth in function of size). However, not all the *NanoFingerprint* parts describe a perfect exponential growth, or any kind of growth, in function of size, and this has penalized the results.

Having the XYZ files, and considering the number of Oxygen and Metal atoms, the performance of the model has increased, but this is an expected result since more predictors have been used. Slightly better results are achieved using a quadratic non-linear model (second case) than a linear one. However, the generation of the XYZ is computationally expensive, and then, the key aspect to think about is: is the improvement of results significantly good so it can be assumed the cost of generating the XYZ files? The answer is no, and the prove is that the results of the 2nd Approach are not much better than the 1st Approach, as the error metrics and representations show.

On the second place, regarding the 'other' datasets nanoparticles (*Liu*, *Gajewich*, *Panthoki*, *Papa* and *Anantha*), the main problem is related to the data. In *Machine Learning*, a key aspect is having a data that is representative in order to generate a model and, on this case, many more nanoparticles from different sizes and from each type were needed. Therefore, the *NanoFingerprint* parts did not follow a perfect curved exponential growth, or any kind at all, which made it difficult to generate a good model. Additionally, some the nanoparticles of low sizes had to be eliminated, since were not helpful to the regression. This reduced the sample even more. Moreover, there were few nanoparticles of larger sizes, as well as many nanoparticles of TiO_2 and ZnO, and few from the rest, leading to an unbalanced data. Then, the

model became biased towards the more frequent samples or metal elements, leading to potential inaccuracies or limited generalization when dealing with less represented nanoparticles.

On this case, the 2nd case of the 2nd approach (quadratic model using the number of Oxygen and metal atoms) is significantly better than the other two and, having the problem with the data as commented, then, having the more predictors, the better. Therefore, it would make sense having the XYZ files assuming the computation expense in generating them. This last statement is in case that there was no option of having a more representative input data.

Regarding to the Leave-One-Out cross-validation, the errors have been slightly higher comparing to the K-Fold cross-validation. This is because the difference in size between the training set used in each fold and the entire dataset is only a single pattern. However, it is useful for the 'other' datasets, since the sample on *NanoFingerprints* is not very big and using a k-Fold cross-validation eliminates more patterns on training.

Finally, a difference between the error metrics has to be commented. The NMSE is useful to compare the real and predicted outputs regardless the scale because it is normalized over the maximum of the two values. However, the MAE as it has been used has a problem: difference in large scale outputs is penalized, since it has more influence on the overall error than difference in low scale outputs, which can have significatively more error, but it is not reflected on the overall error because of this effect. This can be clearly seen on Table 6.23, as it has been commented as well.

5.9. References

[1] MathWorks. Fit linear regression model (fitlm). Available at: https://www.mathworks.com/help/stats/fitlm.html (Accessed: 01/04/2023)

[2] MathWorks. Wilkinson Notation. Available at: https://www.mathworks.com/help/stats/wilkinson-notation.html (Accessed: 01/04/2023)

[3] Ali, P. J. M., Faraj, R. H., Koya, E., Ali, P. J. M., & Faraj, R. H. (2014). Data normalization and standardization: a technical report. Mach Learn Tech Rep, 1(1), 1-6.

[4] MathWorks. Cross-validation. Available at: https://www.mathworks.com/discovery/cross-validation.html (Accessed: 05/04/2023)

[5] Jason Brownlee (2021, February 16). Regression Metrics for Machine Learning. Available at: https://machinelearningmastery.com/regression-metrics-for-machine-learning/ (Accessed: 15/04/2023)

CHAPTER VI
CONCLUSIONS AND FUTURE RESEARCH LINES

The aim of this last Chapter is to summarize the work and contribution of this Master Thesis, extract conclusions and propose improvements for future research lines.

The first part of the Master Thesis consisted on designing an algorithm capable of generating *Structural NanoFingerprint* given the three-dimensional structure of a nanoparticle, described in an XYZ-file. The first approach, which focused on using the VF3 library to succeed in the search of a subgraph into a graph, turned out to be a misleading approach for the problem of generating *Structural NanoFingerprints*.

Therefore, an algorithm from scratch was designed to overcome the task. The algorithm succeeded in the generation and, indeed, some improvements were made to be less-computationally expensive, reducing the time in an order of two. However, this algorithm has two main drawbacks: it needs the existence of an XYZ file to be generated, and for large structures it can take more than two hours.

Regarding to the computational expense time of the algorithm, the code was analysed by parts and the results showed that there were two parts spending most of the computational time: the first one, and most expensive, the conversion of the 3D structure into a graph and, the second one, the computation of Section 2 and 3. Being quite confident that it is difficult to improve these results, a future research line could be the improvement of these two parts, specially the first one, trying to find a way of storing the graph in a fast-computing way, complicated for large structures.

On the other hand, the need of having the three-dimensional structures to generate *Structural NanoFingerprints* is another drawback, since their generation of these is also computationally very expensive. For this reason, the second part of the Master Thesis focuses on artificially generating *NanoFingerprints* to obtain a model which can generate them without depending on the nanoparticle 3D-structure.

Two approaches have been considered: the first one, to generate a model without using the nanoparticles' XYZ files, implying that new *NanoFingerprints* could be generated using just the size and atomic number of the metal atom used (this last case depending on the dataset), and the second one, using the nanoparticle's XYZ files, meaning that the nanoparticle's 3D-structure is already created and the number of Oxygen and Metal atoms are additionally used as predictors. The first approach is the computationally less expensive, since the creation of the

nanoparticle's 3D structure is avoided, whereas the second one avoids the computation time expense that the proposed algorithm spent.

To obtain these models, a dataset of *NanoFingerprints*, generated from the 3D structure of the nanoparticles, is needed. Therefore, the computational time of generating these *NanoFingerprints* had to be assumed. Two datasets have been used: the *Atena* dataset, which includes 99 nanoparticles of TiO_2 with sizes ranging from 2 Å to 100 Å, and a set of other datasets which included 60 different metal-oxide nanoparticles of different sizes.

The first dataset was equally balanced, and good to be used to train a model, but the second one was highly unbalanced and small-sized regarding the number of different nanoparticles. Here, there is a big room for future improvement: obtaining a dataset with many different metal-oxide nanoparticles, equally and sufficiently represented by all the different metal-oxide nanoparticles of the same type. This includes generating the XYZ files for each one and then the Structural *NanoFingerprints* with the algorithm presented on this Master Thesis.

To decide which type of method to generate the models was the most approapriate one, it was observed that the different parts of the *NanoFingerprint* had an exponential growth with size. The clearer evidence were for the *Atena* dataset, whereas for the 'other' datasets the exponential growth was not such evident because of the problem commented with these datasets. Moreover, not all the elements of the NanoFingerprint presented an exponential growth. Then, the selected method was a regression. For the *Atena* dataset, three models were generated: a Linear Regression using the size as input, a Multiple-Linear Regression using the size, number of Oxygen atoms and Metal atoms as inputs and finally a Multiple-Non-Linear (quadratic) Regression using the size, Oxygen atoms and Metal atoms as inputs as well. For the other datasets, the three same models were generated but additionally using the atomic number as input, since these datasets had different metal-oxide nanoparticles and the variation of apparitions may differ depending on each one.

The results have been commented on section 5.8. Conclusions. For the *Atena* dataset, the results have been quite acceptable for *NanoFingerprint* applications, related to deducing some of the nanoparticle's properties (an approximation is enough, it is not needed the exact value). However, for *NanoFingerprint* parts not following any pattern, a regression model might not be the most suitable approach. For the 'other' datasets, the results are not acceptable, since just for some concrete *NanoFingerprint* parts the predictions were acceptable. However, these results are mainly due to the quality of the datasets, a key factor in Machine Learning.

The main objective for future lines of research is to have a model in which, by specifying the size and atomic number of a metal-oxide nanoparticle, the *NanoFingerprint* can be successfully generated, obtaining results statistically significant. Having a good, balanced and sufficiently representative dataset, a fit using the first approach proposed on this Master Thesis could be performed. Additionally, it is aimed to use the models having the XYZ file (2nd approach) to make appropriate comparisons, assuming that the best results will be obtained using the XYZ files, but analysing if the difference between both approaches is close or not to decide if it is suitable using a model without depending on the 3D structure of the nanoparticle (the best case is not to depend on it and avoid its generation).

Analogously, another approach would be generating a different model for each different metal-oxide nanoparticle, which would imply not inputting the atomic number to the model. Then, comparing the performance of each model by separate with the performance of the previously proposed unique model, comparing the errors for each different metal-oxide nanoparticle and concluding which approach is the best.

Another question to answer, in order to decide between using a unique or many different models, is: do the different metal-oxide nanoparticles significatively change depending on the atomic number? On the case of this work, it was tested the performance of the model for 'other' datasets without using the atomic number, and the results were even worse (not shown on this Master Thesis), but what would happen with a better and greater dataset? This analysis would tell if using the atomic number as predictor is really useful or adds more noise than being helpful.

It should be noted that the *NanoFingerprints* generated have the shell thickness and maximum number of bonds per atom fixed at concrete parameters, considering the whole nanoparticles. However, by only considering the shell, through which some properties of the nanocompound can be deduced, would reduce the computation generation time using the *NanoFingerprint* generation algorithm. Additionally, it should be studied whether discarding the thickness parameter, which is dependent of size on this case and non-used on this work as predictor, would affect the results. In is necessary to take into consideration that the outputs would only contain the shell appearances; thus, the thickness effect would already be indirectly considered.

Furthermore, some other improvements are proposed. The first one is designing an algorithm in which one artificially generated Section is used to generate the following one. In this way, the artificially generated *NanoFigerprints* would be more accurate to its correct structure. However, this algorithm would be more complex than a simple regression and would involve

a chain of models, one for each Section (more predictors used as the Section count advances). Moreover, it is important to highlight that the Sections used to generate the following Sections should have been artificially generated from the first model input predictors, and not obtained from the generated *NanoFingerprint* using the main algorithm presented on this Master Thesis, because this would mean practically not improving the computational expense time.

On Figure 6.1 it is shown the prediction of a certain *Local Structure* appearances with respect to size using as predictors the size, atomic number, number of Oxygen and metal atoms and Section 1. Note that the predicted values, even for the test set, fit very good to the variation on the real values. This is not a good approach, since it is being used too much information of the *NanoFingerprint* as predictors, but it serves as example to see how potentially good could be generating a model based on previous predictions, thus, having more predictors than the models presented on this Master Thesis.

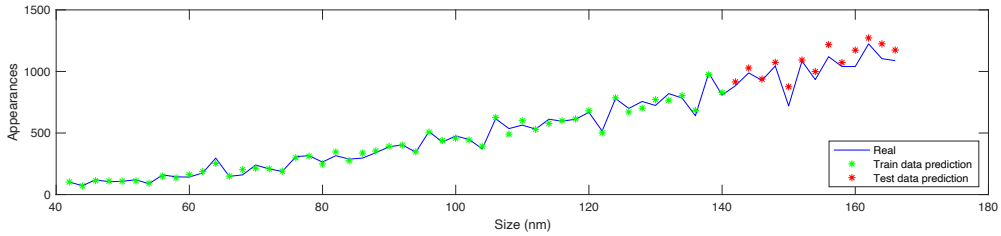


Figure 6.1. Representation of output appearances for a certain *Local Structures* in function of size using up to Section 1 as predictors.

Finally, another approach is searching for other prediction methods different than a regression. A part of the error comes from *Local Structures* which do not follow a determinate pattern with respect to size. This tended to happen with some *M(x,y)*, *M(x,y)* and *O(x,y)-M(x,y)*, for instance, which could vary from higher to lower counts as size grew, or rarely appeared, being 0 for most sizes. The main problem of regression is that uses a mathematical expression to obtain the predictions. Using other prediction techniques, such as a deep learning neural network, could help in highly improving the obtained results, since these more difficult tendencies to predict could be learned by the weights and biases. On this cases, standardization (or normalization) of the data would play a more important role than in regression, especially in the output of data, since in some activation functions, which play the role of deciding if a neuron from a neural network should be activated or not, it is needed to have the outputs among a certain range (from 0 to 1 for sigmoid, for instance).

This analysis of the strong and weak points of the thesis, as well as the proposals for improvements, are aimed for future research of *Structural NanoFingerprint* generation.

ANNEX

At the GitHub from the Research Group *ASCLEPIUS: Smart Technology for Smart Healthcare*[8], there are the following contents publicly available:

- The Subgraph search code project (*Code_sub_graph_search_MacOS*) optimized for MacOS in C, providing instructions on how to compile and execute (*Readme_substructure_search_main* and Readme_vf3, to generate the *vf3lib* executables).

- The *Local Structure* search code project (*Code_localstructuresearch_MacOS*) optimized for MacOS in C, providing instructions on how to compile and execute (*Readme_substructure_search_main* and Readme_vf3, to generate the *vf3lib* executables).

- The *NanoFingerprint generation* code project optimized for MacOS (*Code_nanofingerprint_generator_MacOS*) and Windows (*Code_nanofingerprint_generator_Windows*) in C, providing instructions on how to compile and execute (*Readme_nanofingerprint_generator_main*). Additionally, it is provided the code, in the case of MacOS, and a .bat file, in the case of Windows, to run multiple generations of *NanoFingerprints* automatically. For the case of Windows, the program also generates a .csv file with the execution time in seconds, the number of nodes and the name of the file for each generation.

- The generated NanoFingerprints and the nanoparticles XYZ files, from all datasets (*StructuresAndGenerations*).

- The code in MATLAB (main program to read and function) to execute the *NanoFingerprint* correctness checking error computing (*Code_nanofingerprint_check_MATLAB*).

- The regression codes in MATLAB and functions used (Code_regression_MATLAB), as well as the artificially generated *NanoFingerprints* with the models described in the Master Thesis, in .csv format grouped together by approaches, cases and datasets (inside the regression codes folder, *NanoFingerprints_artificial_results*).

[8] https://github.com/ASCLEPIUS-URV