United Nations
Educational, Scientific and
Cultural Organization

ICTP - East African Institute
for Fundamental Research
under the auspices of UNESCO

Superconducting Critical Temperatures of Metals from Machine Learning Techniques

Firas Elkheir Shuaib Mohammed
Supervised by: Dr. O. Akin-Ojo
ICTP-East African Institute for Fundamental Research (EAIFR)
University of Rwanda
February 1, 2021

The degree is awarded by the University of Rwanda

The degree is submitted in full fulfilment
The TURNITN anti-plagiarism check declaration

Acknowledgment

My special gratitude and thanks go to my supervisor **Dr. Omololu Akin-Ojo**, for his continuous support of my project research, immense knowledge, motivation and patience. His guidance helped me in all the time of research and writing of this thesis. The door to **Dr. Omololu Akin-Ojo** office was always open whenever I run into a trouble spot or had a question about my research or writing.

I would like to express my thanks to **Prof. Gian Marco** and **Dr. Corey Oses**, for their guidance and providing necessary informations regarding to the project.

I thank the **ICTP** and the Office of External Activities **(OEA) at ICTP** for scholarship funds for my MSc program.

My thanks and appreciations also go to the staff of **ICTP** and **ICTP-EAIFR** for their support, teaching, assistance and guidance during my studies.

My special thanks go to my family: my parents, sisters and brothers, for providing me with unfailing support and continuous encouragement throughout doing this program and my life in general. I can not forget to thank my best friends who left fingerprints of grace on my life, they shall not be forgotten.

Above all, my greatest gratitude is due to Allah for granting me the patience and knowledge to successfully pass through this stage of my academic career.

Abstract

Superconductivity is one the most sophisticated and interesting phenomena of nature in which the electrical resistance of a material completely disappears at a certain temperature T_c. Since the superconductivity discovery by the Dutch physicist Heike Kamerlingh Onnes in 1911, enormous research efforts have been focusing on predicting the critical temperature T_c, which control the range and type of industrial applications that can benefit from different superconductors.

It is a very difficult task to predict T_c for new superconductors, even for electron-phonon coupling superconductors. Recently, because of the advances in computing power and different online repositories containing huge amount of data of measured and calculated materials properties have been created over the years, researchers started using Machine Learning (ML) to find the connection between the superconducting state and properties of materials and also design materials with specify values of T_c. In this research we developed ML models to determine the superconducting transition temperature T_c arising from electron-phonon interaction for any metallic material.

Several machine learning models based only on the chemical compositions are developed to estimate the value Debye temperature θ_D for ~ 5200 materials obtained from an online database ["AFLOW"]. Also, several models are trained to predict the density of states near the Fermi level $N(0)$ for ~ 13000 metallic materials existing in the same AFLOW online repository. These models are employed to predict θ_D and $N(0)$ for ~ 4000 compounds of metallic materials obtained from the Supercond databases. We used these characteristics/descriptors and elemental properties to develop ML models for predicting the value of the transition temperature T_C for those (~ 4000) metallic materials. The best model shows strong predictive power, with learned predictors offering potential insight into the mechanism behind superconductivity in this family of materials. Eventually, the best model in predicting θ_D, $N(0)$ and T_c was employed to search for potential new superconductors in the Inorganic Crystallographic Structure Database (ICSD). The compounds from ICSD having the top twenty predicted values of T_c are listed for further studies.

Contents

1 *Introduction*

The goal of this work is to apply machine learning techniques to predict the superconducting transition temperatures T_c of materials. Thus, we will introduce superconductors, the complexity of determining T_c, after which a short description of machine learning techniques and how they can be used to predict T_c is given.

1.1 Introduction to Superconductivity

1.1.1 Short History

Superconductivity is one of the most interesting and sophisticated phenomenon in condensed matter physics. The first discovery of superconductivity was by *Kamerlingh Onnes* in 1911 by chance while studying the resistance of metals (Mercury) at low temperatures. Based on Drude's theory [1] he expected the resistivity to decrease gradually as its temperature is lowered until becomes very small due to Eq. (1) bellow, but he observed that the electrical resistance of mercury dropped suddenly at $4.15K$ and become undetectably small, which was completely unexpected [2].

$$\rho = \rho_o + aT^2 + bT^5 + ... \tag{1}$$

where:
ρ $\equiv$ Resistivity.
T $\equiv$ Temperature in kelvin.
ρ_o $\equiv$ Resistivity at zero temperature, depends on the concentration of impurities.

After that discovery, people started defining superconductivity as a physical property in certain materials where electrical resistance completely disappeared at low temperature. This temperature is called a critical or a transition temperature, see Fig. (1).

The Meissner-Ochsenfeld effect:

After the discovery of Kamerlingh Onnes, superconductivity became a focus of enormous research effort. In 1933 The German physicists Walther Meissner and Robert Ochsenfeld improved the theory of superconductivity by discovering a magnetic phenomenon that showed super-

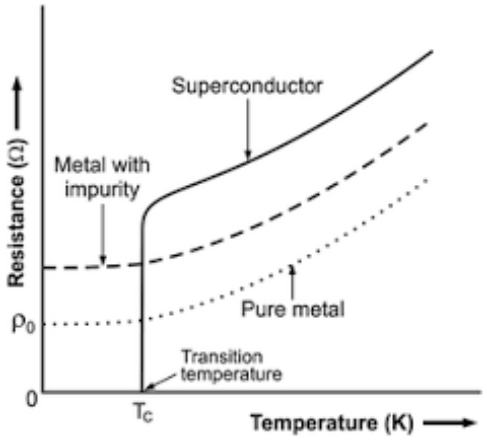

Figure 1: Electrical Resistance of Metal and superconductor as a Function of Temperature.

conductors are not simply perfect conductors. Basically, they considered two spherical materials held at temperature T (Suppose initially, $T > T_c$ and the external magnetic field, $B_{ext} = 0$) see, Fig. (2) [2].
They started by cooling down (left part of Fig. 2) one sample to a superconducting phase, then later they gradually turned on the external field. After a while the sample expelled the magnetic field from its interior (bottom of Fig. 2). This is a consequence of zero resistivity. A normal sample on the other hand was first placed in a magnetic field (right part of Fig. 2) and then cooled to a superconducting phase (bottom of Fig. 2). The observation after cooling the system to below T_c, was that the magnetic field was expelled from the interior of the sample. This fact cannot be deduced from the simple fact of zero resistivity ($\rho = 0$).

Basically, a superconductor expels the magnetic field from its interior by setting up electric currents (Persistent currents) at the surface and this current flows forever without any driving voltage. The surface current creates a magnetic field that exactly cancels the external magnetic field. However, this happens only when $T < T_c$ and the strength

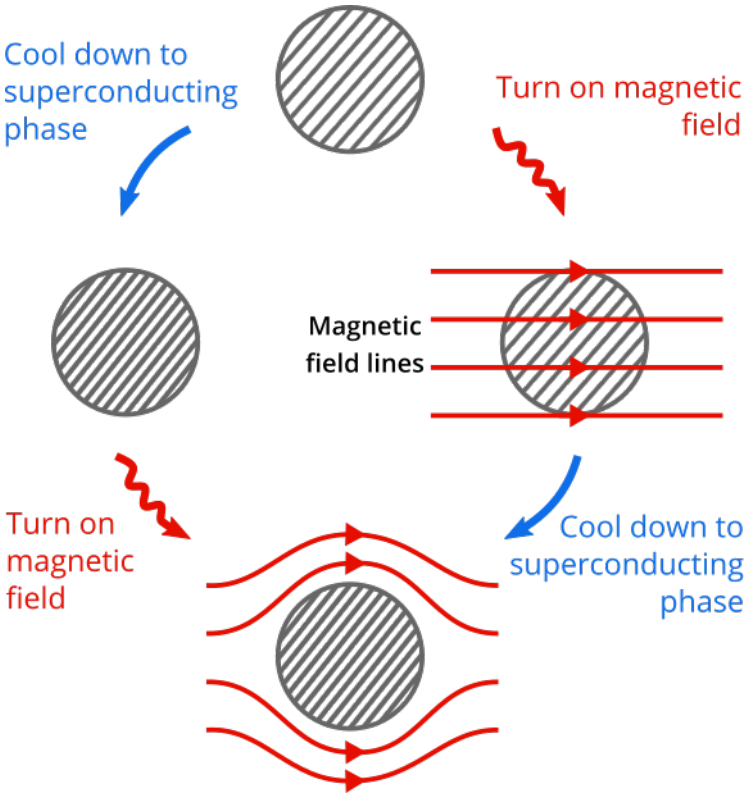

Figure 2: Meissner Effect

of the applied field is below a critical value H_c.

$$B(x) = B_{ext} \exp\left[\frac{-x}{\lambda}\right] \tag{2}$$

$$\lambda = \frac{\lambda_o}{\sqrt{1 - (\frac{T}{T_c})^4}} \tag{3}$$

Where:

λ $\equiv$ Penetration distance at temperature T.

$\lambda_o \equiv$ Penetration distance at temperature T_o and takes value between (30-130)nm, depending on the superconducting material.

Recently, people have defined a superconductor as a sample which exhibits the *Meissner-Ochsenfeld* effect. This is because the *Meissner-Ochsenfeld* effect is a thermal equilibrium property, while resistivity is a nonequilibrium transport effect. As we see in Fig. (2) the final state of the system does not depend on the history of the sample, which is a

necessary condition for thermal equilibrium.

1.1.2 Classes of superconductors

Depending on certain features we can classify superconductors into different types, for instance,
based on the response to an external magnetic induction we can classify superconducting materials into

- **Type I:** The superconductor expels all the magnetic field from the interior of it while the applied field is less than the critical field. This means that the magnetic field cannot penetrate it while it is in the superconducting state (read more [2]). This type is normally exhibited by some metallic alloys and pure metals. The first superconducting element discovered, mercury is of this type.
- **Type II:** have two different critical fields. The superconductor completely expels the external magnetic field from its interior while the applied field is smaller than the lower critical field H_{c1}. But at H_{c1} the external magnetic field begins to partially penetrate the interior of the superconductor until the superconducting state is destroyed at the upper critical field H_{c2} (read more [2]). The first superconducting type II compound discovered is an alloy of lead and bismuth. Usually this type of superconductor is made out of metal alloy (e.g. niobium-titanium and niobium-tin).

Based on Superconductors that follow the BCS Theory we can classify superconducting materials into two. The BCS theory is briefly described under the next subsection.

- **Conventional:** Superconductors that obey the BCS theory (read more [2]). This refers to electron-phonon coupled superconducting

electron-pairs described by BCS theory [3]. Most type I superconductors are conventional.

- **Unconventional:** Superconductors that do not obey the BCS Theory. Most type II superconductors are unconventional. Most high T_c materials are unconventional. Further understanding of the cause of the unconventional pairing leads to further progress in high temperature superconductivity, which is the aim of all researchers working on this field.

1.1.3 T_c and difficulty in its determination

The Prediction of the transition temperature T_c of new superconducting materials is a very difficult task, even for electron-phonon coupled superconductors, for which the theory is relatively well understood. In order to measure the properties of a superconductor, one must cool the sample through T_c and apply a magnetic field. Sample temperature is controlled by the flow of liquid nitrogen or helium, warmed as necessary by a resistance heater. Thermometers are provided to measure temperatures of the sample and heat exchanger.

The first successful microscopic theory to superconductors is from Bardeen,Cooper and Schrieffer (BCS) who postulate electron-electron pairing (to form a boson) arising from electron-phonon coupling. In the BCS [3] model the superconductivity in the low temperature domain (conventional) is a consequence of electron-phonon interaction, responsible for electron pairing and long range Bose-Einstein condensation of so called Cooper pairs. One of the most basic aims of many researchers, is to calculate and predict the superconducting transition temperatures T_c. But, this calculation most of the time is inaccurate [4]. The reason lies not in any basic failing of the BCS theory, but rather in the fact that T_c, depends sensitively on the normal state properties of materials such

as:

- Electronic band structure at the Fermi energy $N(0)$.
- Phonon spectrum which controls the Debye frequency θ_D and the effective electron-phonon interaction V.
- Electron-phonon interaction, which is dressed by the electron-phonon coupling constant λ.
- The fully dressed Coulomb interaction between electrons described by a constant, μ^*.

Based on the BCS theory the superconducting transition temperature T_c for conventional superconductors can be calculated as :

$$T_c = 1.14\,\theta_D \exp(\frac{-1}{\lambda}) \tag{4}$$

$$\lambda = 2\int_0^\infty \frac{d\omega}{\omega}\alpha^2(\omega)\,F(\omega) \tag{5}$$

Where:
$F(\omega)$ $\equiv$ Phonon density of states.
$\alpha^2(\omega)$ $\equiv$ Measures the coupling of electrons to phonons of frequency ω.

Recently, normal state properties like transition temperature T_c, the Debye temperature θ_D, the density of state at the Fermi level $N(0)$, and the electronic heat capacity coefficient γ have been measured and calculated for a number of known superconductors and they are available in several online repositories.

Unfortunately, Eq. (4) is valid only in the weak coupling limit ($\lambda << 1$) and if we neglect the coulomb interaction. For simple metals, λ can be calculated easily because the electron-ion interaction can be represented by a pseudopotiotial. But, generally is more complicated for real metals because of *Bragg reflection* at zone boundaries. **Eliashberg** came

with a theory that describes the attractive interaction between two electrons. The full evaluation of the *Eliashberg* equation is computationally expensive and currently, is not suitable for the new discovery of superconductors.

Later, based on the **Eliashberg** formula, **McMillan** [5] did a detailed study of the dependence of T_c, on the electron-phonon coupling (λ) in metals. He ended with a simple formula for T_c, in terms of a few parameters, the most important one is the electron-phonon coupling constant λ. McMillan formula for T_c is [6]:

$$T_c = \frac{\theta_D}{1.45} \exp\left[\frac{-1.04\,(1+\lambda)}{\lambda - \mu^*(1+0.62\lambda)}\right], \tag{6}$$

$$\text{where} \qquad \mu^* = \frac{\mu}{[1+\mu\ln(E_F/\theta_D)]}, \tag{7}$$

$$\text{and} \qquad \mu = N(0)\int_0^{2k_F} \frac{qdq}{(2k_F)^2} V_c(q). \tag{8}$$

Here,

E_F $\equiv$ Fermi energy.

$V_c(q)$ $\equiv$ Fully screened Coulomb interaction between electrons.

μ^* $\equiv$ Coulomb pseudopotential.

Equation (6), is known as the McMillan equation, yields values of λ when T_c, θ_D, and μ^* are all known. μ^* is not usually known in practice, but since T_c is not too sensitive to μ^*, then we can estimate it as ($\sim$ 0.1$-$0.2). For weak electron-phonon coupling superconducting materials, Eq. (6) should give quite reliable values of λ. But for strong electron-phonon coupling superconductors, T_c becomes sensitive to the exact form of the phonon spectrum, therefore Eq. (6) is not expected to work very well. Unfortunately, Eq. (6) when $\lambda \longrightarrow \infty$, T_c becomes constant.

Later Allen and Dynes improved Eq. (6), based on new computational checks available [9]. They replaced the prefactor $\theta_D/1.45$ of the McMillan equation with $< \omega > /1.20$, where $< \omega >$ is the second moment of the distribution $g(\omega) = (2/(\lambda\omega))\alpha^2 F(\omega)$.

$$T_c = \frac{f_1 f_2\, \omega_{log}}{1.20} \exp\left[\frac{-1.04\,(1+\lambda)}{\lambda - \mu^*(1+0.62\lambda)}\right] \tag{9}$$

Where:

$f_1, f_2 \equiv$ Correction factors that depend on ω_{log}, μ, and λ .

Unlike the *McMillan* equation, the *Allen-Dynes* expression obeys an asymptotic result of *Eliashberg* theory, that $T_c \longrightarrow \sqrt{\lambda}$ as $\lambda \longrightarrow \infty$, with fixed the other parameters. Allen and Dynes formula fits T_c to superconductors known by that time. However, for recently discovered materials Allen - Dynes formula does not fit well. Furthermore, the prediction for the recently discovered high T_c materials, which do not follow the BCS theory, also do not follow Allen-Dynes equation. Thus, researchers investigating the use of machine learning approaches to connect between superconductivity properties such as T_c and chemical/structural properties of materials. There are different machine learning techniques that can be applied to model the critical temperatures of superconductors.

1.2 Introduction to Machine Learning technique

1.2.1 Short History

Machine learning is a general-purpose method which uses mathematical and statistical methods to learn from datasets. This can give the ability to independently find solutions to problems by recognizing patterns in databases. There are three core concepts of machine learning: **data, a model, and learning** (read more [6]). Ideally, a "Machine learning algorithm" means a system that adapts some internal parameters of the "predictor" so that it performs well on future unseen input data.

The first successful self-learning programs were developed by the American pioneer *Arthur Lee Samuel* in 1959. He developed a Checkers-playing program. Basically, the computer has the patience of playing tens thousand of games against itself; In contrast, no human being has this patience. By doing that the computer was able to get much experiences in playing this game and at the end became a better checkers player than Samuel himself (read more [7]).
Machine learning is made up of three parts:

- Computational algorithm, which has a responsibility of making predictions.
- Features "descriptors" that make up the decision.
- Base knowledge of actual values of a property that we train the system to learn and make prediction.

Basically, we feed our dataset for which the correct answer is known to the model. Then we run and adjust the algorithm until the output (prediction) of the algorithm agrees with the exact known answer. Ide-

ally, increasing amounts of input data help the system learn and process higher computational decisions.

1.2.2 Machine Learning Technique

There are several different Machine Learning techniques based on the learning algorithms used. The main ones are:

- Supervised learning: In this case we have to teach the system how to do something. Initially, the system learns using given input and output pairs. Basically, we train the system to learn from successive calculations with different inputs and outputs and to establish connections. Supervised learning has several algorithms like:
  - **Regression algorithms**: refer to the fact that we are trying to predict continuous output values. Some of the algorithms that come under this type are as *Linear Regression, Ridge Regression, Random Forest, Artificial Neural Networks.* Regression is a fundamental problem in machine learning, and it appears in different research areas and applications (read more [6]).
  - **Classification algorithms**: refer to the fact that we are trying to predict discrete output values. For instance if we are looking at cancer tumors and trying to decide whether the tumor is malignant or benign, therefore there are only two desecrate values. 0 (for malignant) and 1 (for benign).
- Unsupervised learning: In this case we do not have to teach the system how to do something, the data is not explicitly labeled into different classes. Using unsupervised learning algorithms the data can be identified based on their structures, similar segments, densities, and other similar features. It is mainly used for learning segmentation (clustering).

1.3 Aim and Objectives of this Work

The discovery of super conducting material has generally proceeded by chance, trial and error. At the same time it is difficult to design materials to specify values of T_c. The goal of this research is to use the method of Machine Learning (ML) to determine the superconducting transition temperature T_c arising from electron-phonon interaction for any metallic material. For materials whose superconductivity arises from electron-phonon coupling, the BCS theory shows that T_c should depend on the electron density of state at the Fermi level, the electron-phonon coupling and certain details of the phonon spectrum (which depends on the ionic masses among other atomic properties). We use these characteristics (descriptors) to develop ML models for predicting the T_c for different metallic materials. The training and validation data used was obtained from the AFLOW [8] and Supercond [9] databases. We obtained descriptors using the **MAGPIE** program and "featurizer" from **MATMINER**. The T_cs predicted for metals using ML models are presented in this work.

2 *Literature Review*

In the past, scientists used the McMillan and the Allen-Dynes formulae, consistent with strong-coupling theory Eq. (9) to obtain T_c. Predicting the critical temperature T_c of newly discovered superconductors is a very difficult task and the Allen-Dynes formula does not work.

Recently, many databases include measured and calculated properties of materials have been created over the years, and machine learning (ML) algorithms can be trained/developed on the collected features in these databases. These ML models can also be used to predict critical temperatures T_c of unknown superconductors. Recently, two groups have used ML approaches to predict the critical temperature T_c and make connections between structural/chemical properties of materials and superconductivity.

An ML model based on compositions of materials was developed by "AFLOW scientist group" [10]. Basically, they extracted a list of $>$ 16000 compounds from the SuperCon database, which is a database that contains the composition of materials and their critical temperatures. This database is an online repository for superconducting materials known from experiment. In SuperCon, roughly 5700 compounds are cuprates, 1500 are iron-based and the remaining set is a mix of various materials, including those that are likely to be conventional electron-phonon driven superconductors. Unfortunately, this repository provides only the chemical formulae and T_c. The AFLOW group employed the Materials Agnostic Platform for Informatics and Exploration *(MAGPIE)* to convert the composition information into meaningful features/descriptors. MAGPIE computes a set of features for each sample [10], including elemental property statistics like the standard deviation and the mean of 22 different elemental properties (e.g., period/-

group on the periodic table, melting temperature, atomic radii, atomic number), as well as electronic structure descriptors. MAGPIE features capture significant chemical information, which play a decisive role in determining structural and physical properties of materials.

A classification model was trained by the AFLOW group to separate these materials into two portions based on their values of T_c: above $10K$ and below $10K$. The model used *coarse-grained* descriptors based only on the chemical formulae. It shows good predictive power, with accuracy of about 92%. After training a successful classification model, three separate regression models focusing on materials with $T_c > 10K$ were trained to estimate the T_c values for iron-based, cuprate, and low T_c compounds. The three models showed good performance on compounds belonging to their training set family while demonstrating no predictive power on the others. For iron-based compounds R^2 was 0.74, for cuprates it was just below 0.8, and for materials with low T_c R^2 was about 0.85. The R^2 statistics is a statistical measure that gives the percentage of variation explained by the relationship between known and prediction variables. Ideally, R^2 represents the fineness of fit of a regression model (i.e. varies from 0 to 1). The closer the value of R^2 to 1, the better fitted is the model.

Finally, new descriptors using materials data from the AFLOW Online Repositories were added to improve the accuracy of the three models. Unfortunately, they ended up with an unsatisfactory regression model. They interpreted that as a problem arising from the chemical sparsity of superconductors in the ICSD (see Methodology section), i.e., the dearth of closely related compounds, usually, created by chemical substitution (read more [10]).

In 2019 another interesting model was developed by a group of scientists from the University of Florida [11]. They used a new machine learning approach called *SISSO* to improve the Allen-Dynes formula Eq. (9), in the context of recently discovered materials. They applied the *Sure-Independence Screening and Sparsifying Operator (SISSO)* method to predict the critical temperature T_c from μ^*, λ and ω_{log} with the goal to obtain an equation of similar or enhanced performance to the one proposed by *Allen* and *Dynes*. Basically, this modern technique searches for analytical relations between a minimal set of features (for each of the input feature one can apply a series of functions to get a combination of additional features).

The $SISSO$ model provided an optimal equation [10], of similar performance to the *Allen-Dynes* expression. The equation-based machine learning used the values of ω_{log}, μ^* and λ of 29 materials. The Root-Mean-Square Error (RMSE) of this equation evaluated on the training data was $0.25K$. The $RMSE$ in a regression model is a distance measure between the correct answer and the predicted target. The smaller the value of the RMSE, the better is the predictive accuracy of the model. The SISSO result is impressive given the use of only three parameters and a single numerical coefficient compared to four parameters and seven coefficients for the Allen-Dynes expression.

$$T_c^{SISSO} = 0.09525 \frac{\lambda^4 \omega_{log}}{\lambda^3 + \sqrt{\mu^*}} \tag{10}$$

Equation (10) is a numerical equation, therefore we do not expect all the terms have physical meaning. For example the $\sqrt{\mu^*}$ term, which may be a proxy for a constant term due to the small range of data and the paucity of features at this level of learning. In Eq. (10) $T_c \to 0$ as $\lambda \to 0$ even at nonzero μ^*. This equation also, increases linearly with λ, at very strong couplings. This behavior violates the asymptotic limit

of *Eliashberg* theory, $T_c \sim \sqrt{\lambda}$, built into the $Allen - Dynes$ equation. This disagreement with physics is due to the absence of data points, either in the training or the testing set (read more [11]).

3 *Methodology*

In ML, two things are important: ① **Data and** ② **Model**. In this work datasets have been extracted from different sources. The data is fed into three different ML models under the supervised learning algorithms (see[1.2.2]): ⓘ*Linear regression,* ⓘⓘ*Ridge regression and* ⓘⓘⓘ *Random forest*. The models were trained based mainly on MAGPIE descriptors obtained from the composition, each model showing different accuracies with different deviations. In this chapter we discus how we prepared our datasets and the ML models used.

3.1 Data Sources

This is made of the target properties $[T_c, \theta_D, N(0)]$ for different compounds and the corresponding descriptors. Descriptors having meaningful physical correlations, are highly recommended to create accurate and powerful ML models with high accurate predictions.

3.1.1 Target Data (y_i)

We used data extracted from SuperCon [9] and AFLOW [12] [13], online database houses. AFLOW has nearly 600 million calculated properties that have been extensively validated with observed properties, like θ_D and files of calculated properties such as the electronic density of states (see [10]). It also contains information for the vast majority of compounds in the ICSD (the inorganic crystal structure databases) [14] and about 1200 superconducting materials. We predicted T_c, θ_D and $N(0)$. The target data for these predictions were obtained and filtered as follows:

3.1.1.1 The Critical Temperature T_c

We have used a list of $\sim$ 16400 compounds, that were extracted from SuperCon database by AFLOW scientist group [10]. This list https://github.com/vstanev1/Supercon provides only the chemical composition and T_c, for different materials. Many of these materials have no reported T_c, and many are materials for which the mechanism of superconductivity is still under debate. However, the presence of these materials, in our dataset for the purpose of creating ML models can be problematic when applied to new metallic materials because predictions of ML models are limited to the patterns encountered in the training set. To circumvent this problem, the data was cleaned and processed using a some-made python script to keep only materials with $2.00K < T_c < 40.00K$ (metallic materials with stable superconducting state). Only about 4000 compounds in SuperCon fall into this category. These compounds are used to develop ML models to predict T_c for metallic superconductors.

3.1.1.2 Debye Temperature θ_D

The online database AFLOW is a framework having a high throughput ab initio calculations of the density functional theory (DFT) with standard ensures that the derived properties and calculations are reasonably well-converged, empirical (reproducible), and above all, consistent (fixed set of parameters), a particularly attractive feature for machine learning modeling. Many important materials properties for superconductivity are calculated within the *AFLOW* framework, and are easy to access through the *AFLOW* online database (see [10]).

From the AFLOW online repositories http://aflow.org/search/advanced.php, we were able to access calculated thermodynamic properties and searched and obtained the Debye temperature θ_D for about

$\sim$ 5500 materials in an Excel spreadsheet which we converted to a **.csv** file. A python script was developed to filter our dataset and only non duplicated elemental compositions are considered in this work ($\sim$ 5200).

3.1.1.3 Density of State at the Fermi Level $N(0)$

AFLOW also contains information for the vast majority of compounds in the ICSD. Files with information about the electronic density of states for $\sim$ 60000 of these materials http://aflowlib.duke.edu/AFLOWDATA/ICSD_WEB/ were extracted via a python script. Another python script was developed to pick only elemental compositions with their corresponding non zero density of state near to the Fermi level $N(0)$ (metallic materials), and also saved the data in **.csv file** (comma-separated value). CSV files are a common file format for transferring and storing data. Typically, the first row in a CSV file contains the names of the columns for the data. Terminal techniques commands are used to save only the chemical formula of theses compositions and remove all things not part of the chemical formula.

Finally, we filtered our dataset and kept only non duplicated elemental compositions with $N(0) \neq 0$ ($\sim$ 13600) using a some-made python script. Data used in this work is available at:
https://github.com/Firaskigali/Critical-Temperature-Tc

3.2 Loading The Data to Python Pandas

Python pandas is also called Pandas DataFrames, it is a free software libraries written for the *Python programming language* for manipulating and analyzing data. It was developed by Wes McKinney to store data, and offers data structures and operations for manipulating numerical tables.

First one needs to load the pandas libraries: **import pandas as pd**,

then read and store the data from **.csv file** using *data = pd.read_csv ("filename.csv")*. When one specifies a filename to pandas without changing his working directory to the directory where the file exists, pandas looks in his "current working directory" and searches for the filename. Pandas also can delete duplicated rows in all columns or in specific column by using **drop_duplicates()** function. By default, **drop_duplicates()** function removes completely duplicated rows, which is quite useful in cleaning your dataset (see Appendix section [6]).

3.3 Descriptors

The first step is to turn a string composition into a pymatgen Composition. One way to do this is to use the conversions featurizers in MATMINER by loading the library: ***from matminer.featurizers.conversions import StrToComposition***. Then for each compound whose target properties we have from [3.1.1], we used one of the featurizers in MATMINER [15] (**from matminer.featurizers.composition import ElementProperty**) to call the MAGPIE program to add a suite of descriptors to the DataFrame, (large sampling, but features limited to compositional data). These descriptors for each compound includes *mean electronegativity, average deviation of electronegativity, mean mass and other features* obtained from the chemical formula of the compound. A complete list of all the descriptors can be found in [10].

MATMINER is a Python library for data mining the materials properties. The advantages of using MATMINER is that it contains routines that transform and featurize complex materials features (e.g. band structure, crystal structure, and composition), into numerical descriptors for data mining. It also allows us to work with the pandas data format to make several ML tools and libraries available to materials science applications. Finally **MATMINER** generates interactive

plots through an interface to the **plotly** visualization package.

3.4 ML Models Used

Several ML models are available in the python **Scikit-learn** libraries, which is a simple and efficient tool for predictive data analysis [16]. These models take as input a number of descriptors of each material we have from [3.3]. These models are:

3.4.1 Linear Regression LR

From Scikit python libraries, first one needs to load **Linear Regression** library: *from sklearn.linear_model import LinearRegression* as **lr** and the regression is solved by **Y=lr.fit(x,y)** (see Appendix section [6]). Basically, Linear Regression outputs a function $h_\theta(x)$, which by convention is called the *hypothesis function*. Its job is to take the input x and tries to output the estimated value of y by assuming a linear relationship between the input variables (x) and the single output variable (y).

$$y = h_\theta(x) + \Delta \tag{11}$$

$$h_\theta(x) = \theta_o x_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n = \theta^T \cdot X \tag{12}$$

where:

Δ $\equiv$ **RMSE** and represents the errors are residuals.

$\theta_o, \theta_1, \ldots, \theta_n$ $\equiv$ Parameters of the model.

Different choices of these parameters correspond to different functions $h_\theta(x)$ or to a different straight line fit to our data. This means *Linear Regression* is going to solve a minimization problem that makes the average distance between the prediction value $h_\theta(x)$ and the target value y to be small.

3.4.2 Ridge Regression RR

When we have more parameters than the observations, or even when we have near-linear relationships among the independent variables (a condition called *Multicollinearity*), this can create an inaccurate estimate of our regression parameters, increasing the error of our model parameters and degrading the predictability of the model. A simple way to avoid and correct this multicollinearity is to use **Ridge Regression**.

Firstly, from Scikit python libraries, we need to load Ridge Regression: **from sklearn import linear_model** as **rr** and the regression is solved by **Y= rr.fit(X,y))** (see Appendix section [6]). Ideally, Ridge Regression adjust the parameters back into their original scale, when they are displayed (read more [17]).

3.4.3 Random Forest RF

Random Forest Regression is a quite powerful algorithm, with a similar idea as linear regression and we can train this model following the same process for liner regression. The only difference is, the *Random Forest* hypothesis function seems like a black box that can't be represented easily as a function. In general, *Random Forest* models work quite well on large datasets, produce better results, and are able to work with missing data by estimating values of them.

Easily, one can import the random forest library: from Scikit python libraries by **from sklearn.ensemble import RandomForestRegressor** as **rf**. The **RF** model tries to predict a property based on a series of questions you ask on different features, and combine those questions in order to end up with a given value at a given end point. Decision Trees are generated to obtain non-linear relationships between input attributes and our interest values.

The Decision Trees work as a bunch of **if-else** conditions. Start on the top with one node (by default). Then the node splits into right and left nodes, these nodes then split into their respective left and right nodes. At the end of the leaf node, the average of the observation that occurs within that area is computed. Ideally, the maximum depth of the trees (n_estimators) must be specified to prevent the tree from going too deep (see Appendix section [6]).

3.5 R-Squared (R^2)

R^2 is the default metric for scikit-learn regression problems. It is used to compare between different models developed on the same dataset and that helps us to get a rough feeling about the performance of each model. We used it explicitly like: ***r2_score(y_true, y_pred)***, the higher scores are better. Ideally, a model that explains with a low value R^2 would show a low level of correlation. A model with an R^2 is 1 means there is no variance at all and the predicted and observable values are perfectly correlated which mean the model fits the data perfectly. Variance is a measure of how far predicted values differ from the average of predicted values. Basically,

$$R^2 = 1 - (\text{SSE} \ / \text{TSS}) \tag{13}$$

Where:
SSE $\equiv$ Is the sum of the squares of the differences between the target values [3.1.1] and the ML predicted values.
TSS $\equiv$ Is the sum of the squares of the differences between the target values and the average of the target values.

3.6 Root Mean Squared Error (RMSE) Δ

From Scikits-learn one can import: ***from sklearn.metrics import mean_squared_error*** library, which going to predict the cost I want my learning algorithm to pay, if it outputs the prediction value, and mathematically represented as.

$$\Delta = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2} \tag{14}$$

Where:
$m \equiv$ Total number of training examples.
$y_i \equiv$ The exact value of the target.
$x_i \equiv$ Set of descriptors corresponding to the given data y_i.

This means that given a regression line through the data we calculate the distance from each data point to the regression line, square it such that points below the mean don't cancel out with points above the mean, and sum all of the squared errors together. We then take the average and finally, the square root. We get the RMSE with a function in *scikit-learn* as: **np.sqrt(mean_squared_error(**$y_{true} = y$ **,** $y_{pred} = lr.predict(X)$**))**. RMSE is les intuitive to understand, but extremely common, (see Result section [4]).

3.7 The predictive power of the model

When the model fits to a dataset, that does not mean it is necessarily a good model because it may fail to generalize to new data that are not in the training set. The standard way to evaluate a learned hypothesis is the following:

3.7.1 Cross Validation

We Split each target dataset [3.1.1] into two portions. The first portion is going to be our usual *training set*, and sent a random 90% of our data to it. The second portion is going to be our test data and we call it the *cross validation set*, and sent a random 10% of our data. The cross validation set is used to estimate a generalization error of the selected model. If the dataset is not randomly ordered, may be better to randomly reorder it before splitting the data.

There are several cross validation techniques, In this work we used the most popular method i.e the *K-Fold* Cross Validation. From Scikit python libraries, first we load the cross validation and KFold libraries: ***from sklearn.model_selection import KFold, cross_val_score***. Initially, the main training dataset is broken up into 10 equal parts. The first part is kept as the testing set and the remaining 9 parts are used to train the model. Then the trained model is tested on the test set. This process is repeated 10 times, in each case we keep on changing the test set. Thus, every data point gets an equal opportunity to be included in the test set. As a final result we get R^2 and $RMSE$ on average. The cross validation error is Computed by the same objective function as we always use, except that, now, it is defined using the test dataset (see Appendix section [6]).

3.8 Capturing the most important features in ML models

Fortunately, Random forest regression has a function that shows the most important descriptors in our dataset, which can reflect the nature and the mechanism that govern the training data. Nevertheless, some important predictors have no straightforward interpretation. We used this function explicitly like this: **rf.feature_importances**.

3.9 Making Predictions

The goal of a ML project is to reach a final model that performs the best, which is a model that we need to use for making predictions on new data. We can make a regression predictions for new data instances using our finalized regression model in scikit-learn using the predict() function: ***from sklearn.model_selection import cross_val_predict*** (see Appendix section [6]).

4 *Results*

In the McMillan formula T_c depends on the Debye temperatures θ_D. Thus, it may be useful, to predict θ_D for materials for which we need to predict T_c.

4.1 Predicting the Debye Temperature θ_D

The Debye temperature is one of the important property of materials. It refers to the temperature of crystals at the highest normal mode of vibration (where some or all atoms vibrate together with the same frequency), and it correlates the elastic properties with the thermodynamic properties such as phonon spectra, specific heat capacity, thermal conductivity, coefficient of thermal expansion and lattice enthalpy. We suggest including it in our model as a descriptor which can improve the predictive power of a ML model, especially since it features in the McMillan formula for T_c. For a simple material we have [15].

$$\theta_D = \frac{3nhN_A\rho^{1/3}}{4\pi kM} v_m \tag{15}$$

where:

h $\equiv$ Planck's constant.

k $\equiv$ Boltzmann's constant.

N_A $\equiv$ Avogadro's number.

ρ $\equiv$ Number density.

M $\equiv$ Molecular weight.

n $\equiv$ Number of atoms per primitive unit cell.

v_m $\equiv$ Averaged wave velocity integrated over several crystal directions.

Based on the dataset [3.1.1.2] that extracted from AFLOW database ($\sim$ 5200 compounds) and descriptors that we added to the dataset from MAGPIE [3.3], several ML models [3.4] are applied to the dataset.

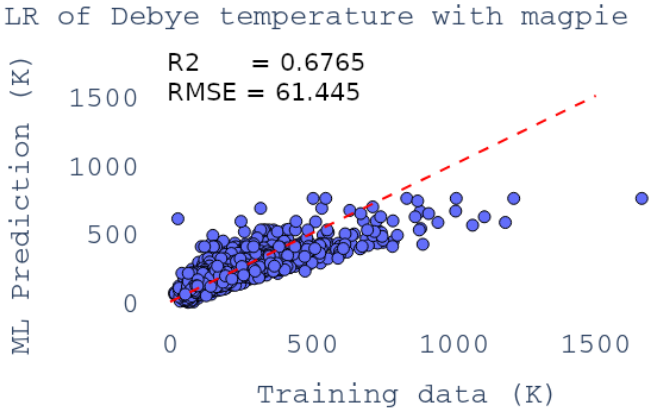

Figure 3: How Linear Regression fits to our dataset

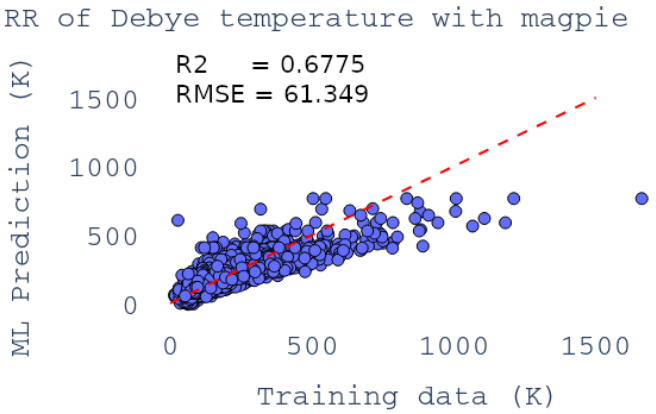

Figure 4: How Ridge Regression fits to our dataset

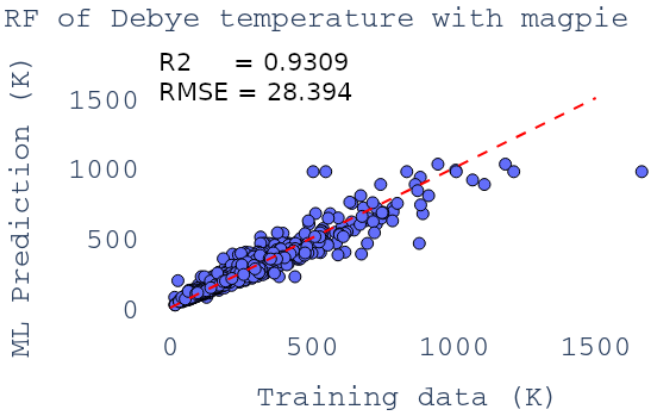

Figure 5: How Random Forest Regression fits to out dataset

These models are trained using *Scikit-learn* which is powerful and efficient ML Python libraries. Unsurprisingly, **The Linear-Regression and Ridge Regression** do not fit well to our training dataset, as one can see in Figs. [3] and [4], respectively. This looks reasonable since Linear and Ridge regressions are simple (high bias) models. The two models achieve $R^2 \approx 0.68$, with RMSE $\sim 61K$.

As we expected, the Random Forest model does reasonably well on our training dataset, the model achieved $R^2 \approx 0.93$ with deviation $(RMSE) = 28K$ for predicting θ_D. We suggest that the Random Forest regression [3.4.3] is more powerful and it is able to make good predictions. All the figures above illustrate how the models fit to our training dataset but, this does not tell how well the model is able to make predictions for new datasets. To generalize our models the dataset is parsed randomly (90%/10%) into training and testing subsets (cross validation).
Fig. (6) and Fig. (7) show how Linear and Ridge regressions are generalized to a new data that we do not have in our training dataset. The models do not perform better than the models trained on the whole data points because, not all the data are included in the training set. The Random Forest regression model Fig. (8) is better for Debye temperature predictions.

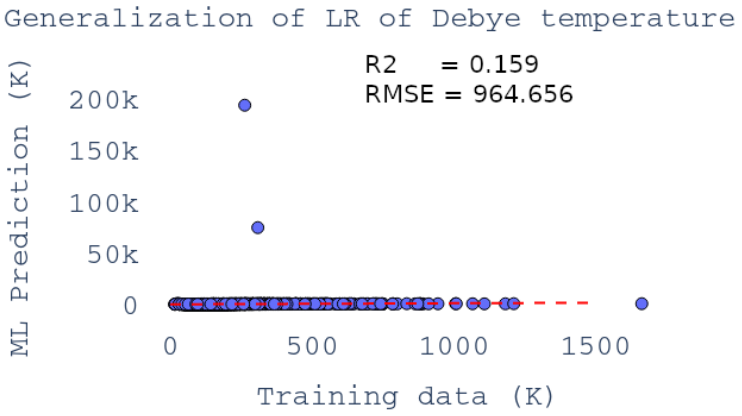

Figure 6: How Linear Regression fits to a new dataset

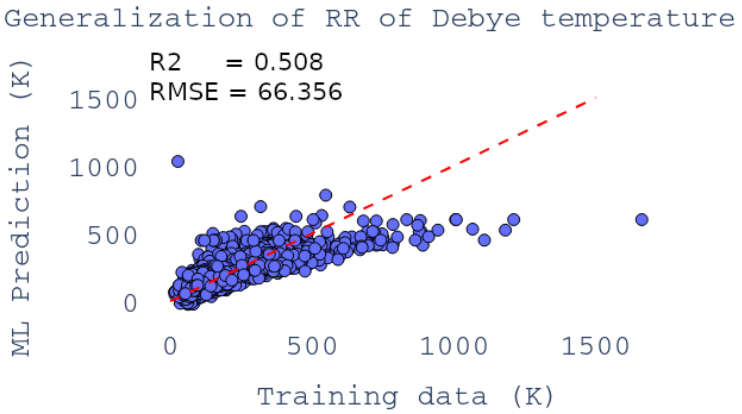

Figure 7: How Ridge Regression fits to a new dataset

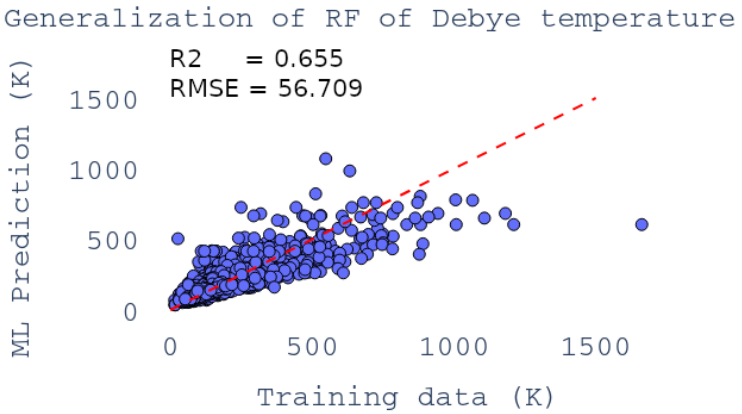

Figure 8: How Random Forest Regression fits to a new dataset

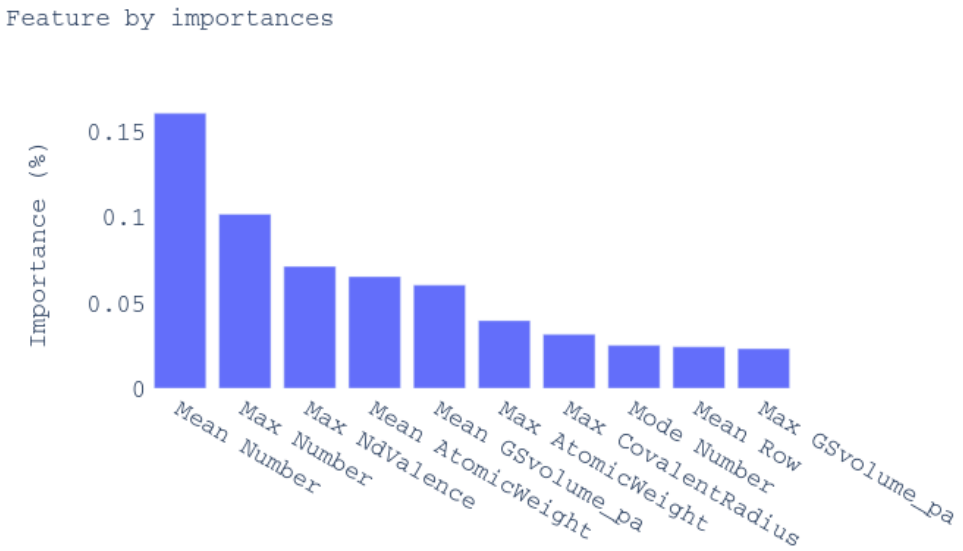

Figure 9: The most important features used by the random forest model

.

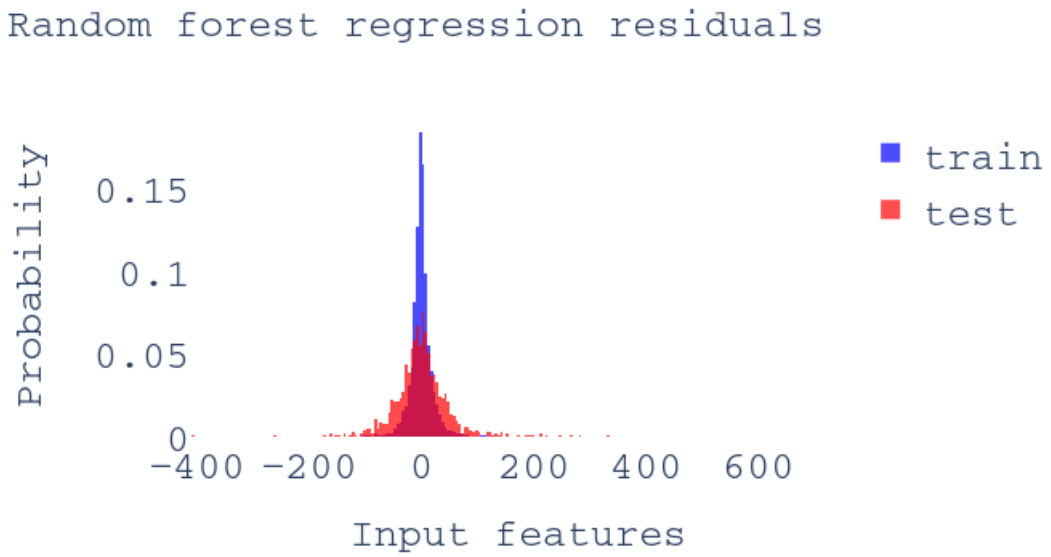

Figure 10: Visualization the training error

The most important descriptors/features in the Random Forest regression for predicting the Debye Temperature θ_D is **mean Number** see Fig. (9), which is the mean value for the all the features that we have in our dataset and it is not clear why it is the dominant descriptor. This is followed by: Max Number, Max NdValence and mean atomic weight. The correlation of θ_D with the mean atomic weight is obvious from Eq. (15). Also, Fig. (10) illustrates how well our Random Forest model estimates the value of θ_D from the training and test datasets.

4.2 Predicting the Density of States at the Fermi level N(0)

Conventional superconductivity is a macroscopic effect which results from the condensation of Cooper pairs into the same state when the temperature drops below a critical temperature T_c [1.1.2]. This condensation signals an abrupt change in the properties of the metal. To break a Cooper pair, one needs to add a minimum amount of energy to the pair, unlike in normal metals, where the state of the electron can be changed by any small amount of energy. Therefore, there is an energy gap for single-electron excitation and this gap vanishes at the critical temperature T_c when the superconducting state exists. The BCS theory gives the superconducting critical temperature T_c in terms of the density of states at the Fermi level $N(0)$ and an effective electron-phonon coupling V (see Eq. [4] with $\lambda \simeq N(0)\,V$). Therefore, we include it in our model as a feature which may improve the predictive power of our ML model.

Once we have a list of relevant predictors [3.1.1.3] with suitable descriptors, various ML models are applied to the dataset ($\sim$ 13600 compounds)
However, the models do not work well as can be seen in Figs. 11, 12 and 13, which clearly shows that we require more powerful and interpretable models, and/or better descriptors in order to obtain a good

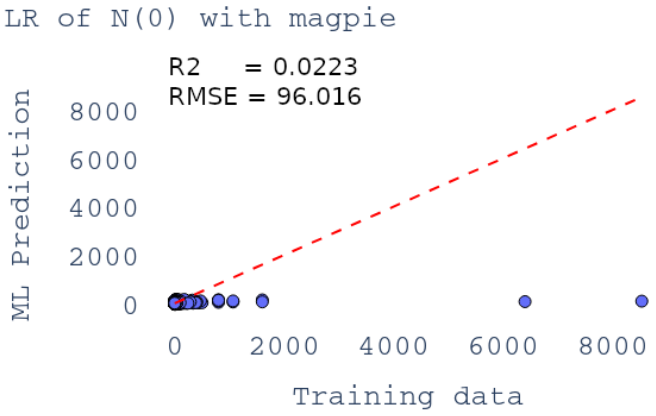

Figure 11: How Linear Regression fits to our dataset

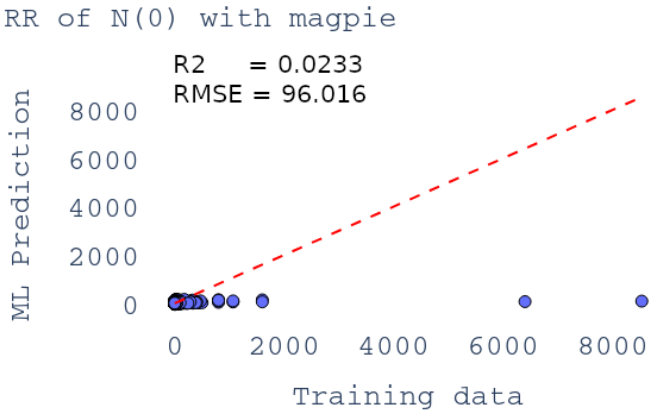

Figure 12: How Ridge Regression fits to our dataset

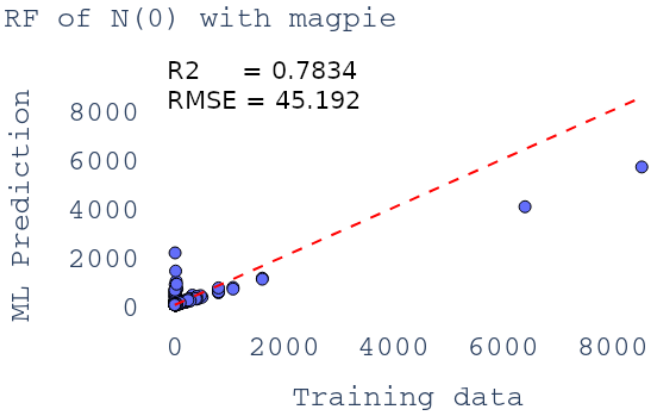

Figure 13: How Random Forest Regression fits to our dataset

estimation.

4.3 Predicting the Critical Temperature T_c

Unfortunately, only $\sim$ 1200 compounds of our SuperCon dataset have calculated properties within the AFLOW database. We employed our Random Forest models [4.1] and [4.2] to predict θ_D and $N(0)$ respectively for the dataset [3.1.1.1] ($\sim$ 4000 compounds) and added them as descriptors in our training dataset. Based on these informations and descriptors that we created through $MAGPIE$ [3.3], three ML models [3.4] were developed to predict the critical temperature T_c. Each model shows different predictive power with different deviation see figures below. Figs. [14] and [15] look reasonable for Linear Regression and Ridge Regression, since they are not powerful models. But the more advanced machine learning model Random Forest Fig. [16], does better on the training data, giving very low RMSE ($\simeq$ 1.7K) and very high R^2 ($\simeq$ 0.98).

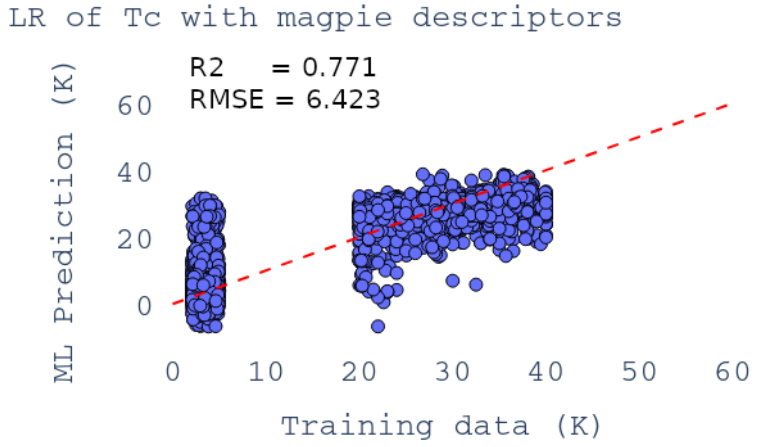

Figure 14: How Linear Regression fits to our dataset

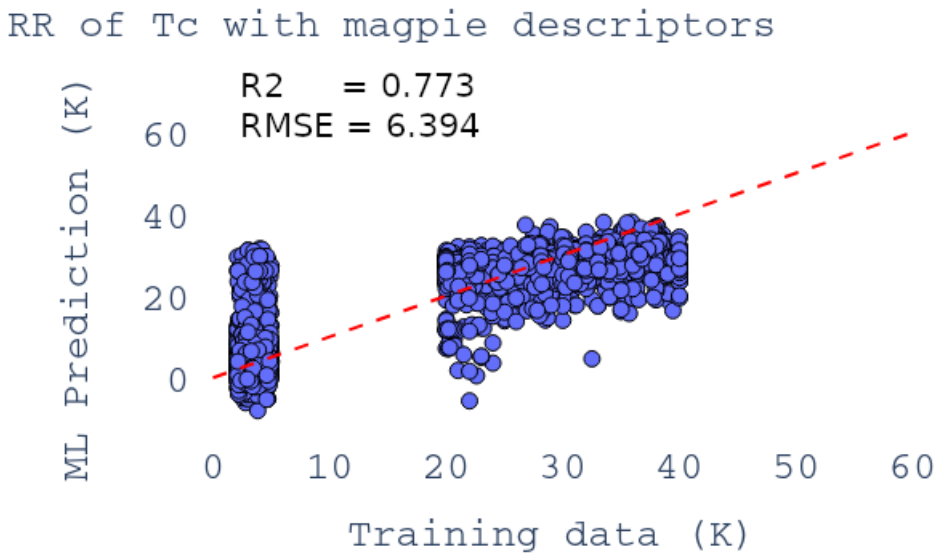

Figure 15: How Ridge Regression fits to our dataset

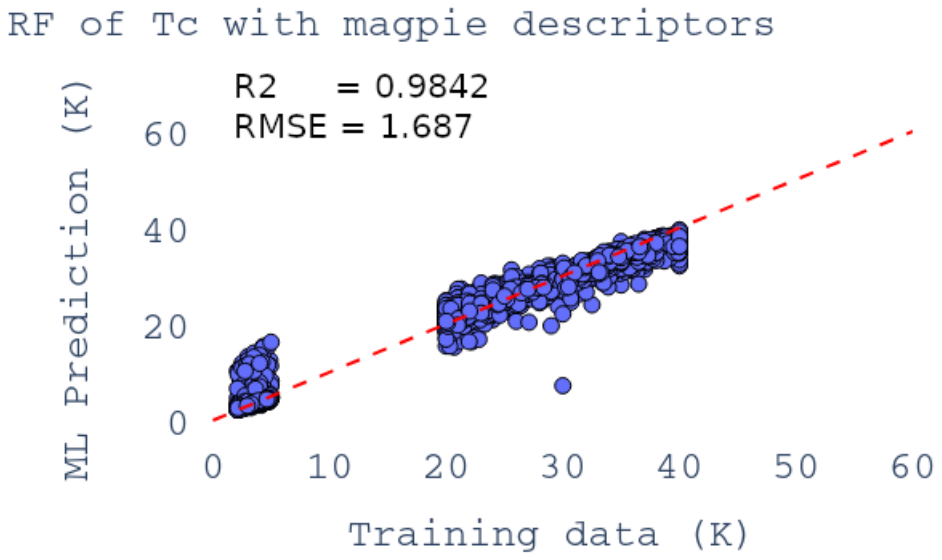

Figure 16: How Random Forest Regression fits to our dataset

As usual, in order to really validate that we are not over-fitting and to generalize our ML models for application to a new materials, we need to check the cross-validation rather than just fit the data. The dataset was split randomly (90%/10%) into two portions: training and test subsets and we trained our models on these subsets.

Surprisingly, the RMSE for linear regression (LR) and ridge regression (RR) models are not bad, see Fig. (17) and Fig. (18). However, there are definitely some outliers. As expected, the Random Forest regression model does much better than the linear regression and ridge regression approaches, see Fig. (19).

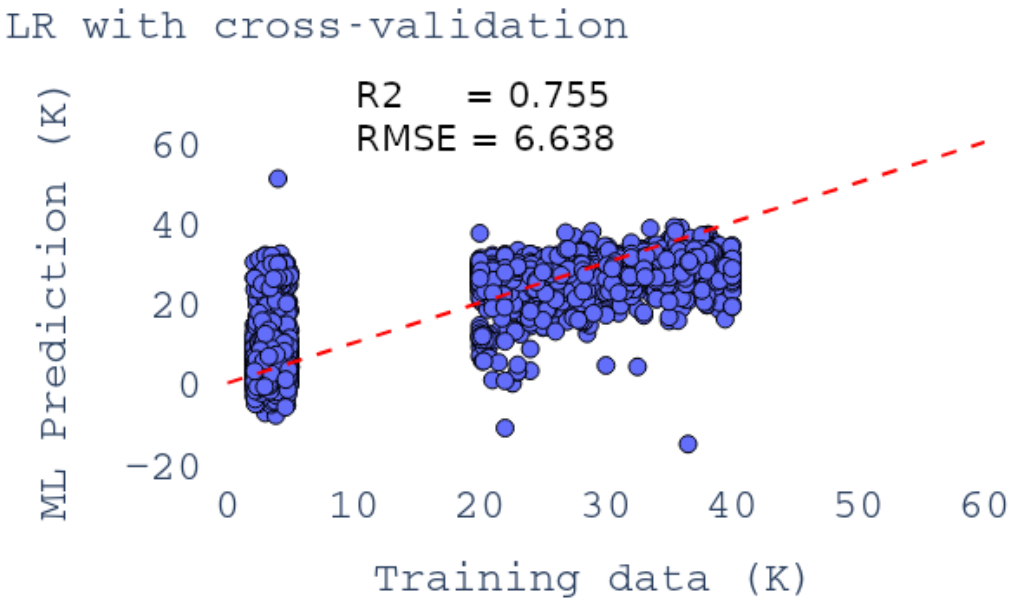

Figure 17: How Linear Regression fits to a new dataset

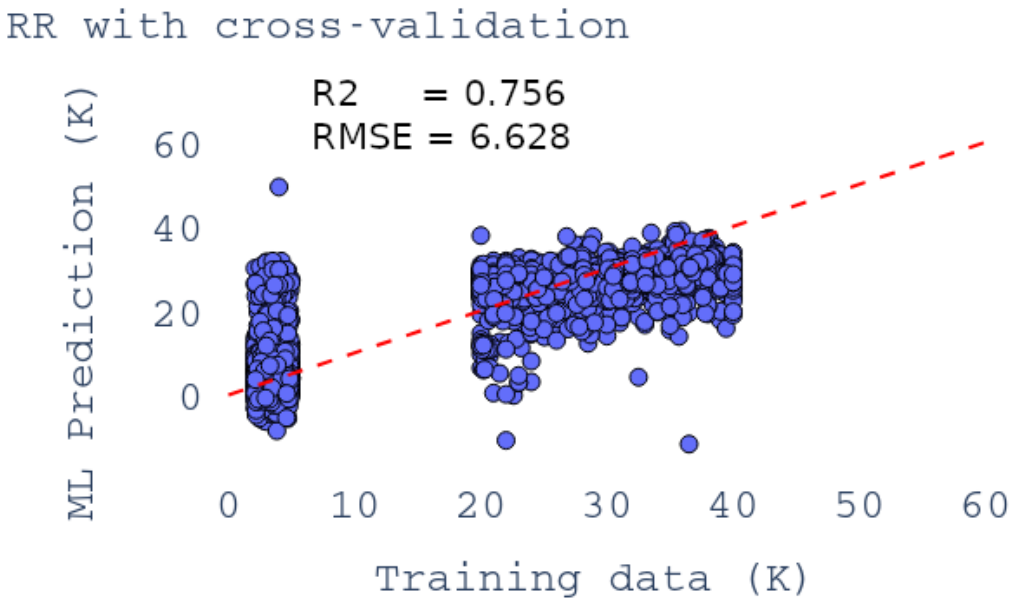

Figure 18: How Ridge Regression fits to a new dataset

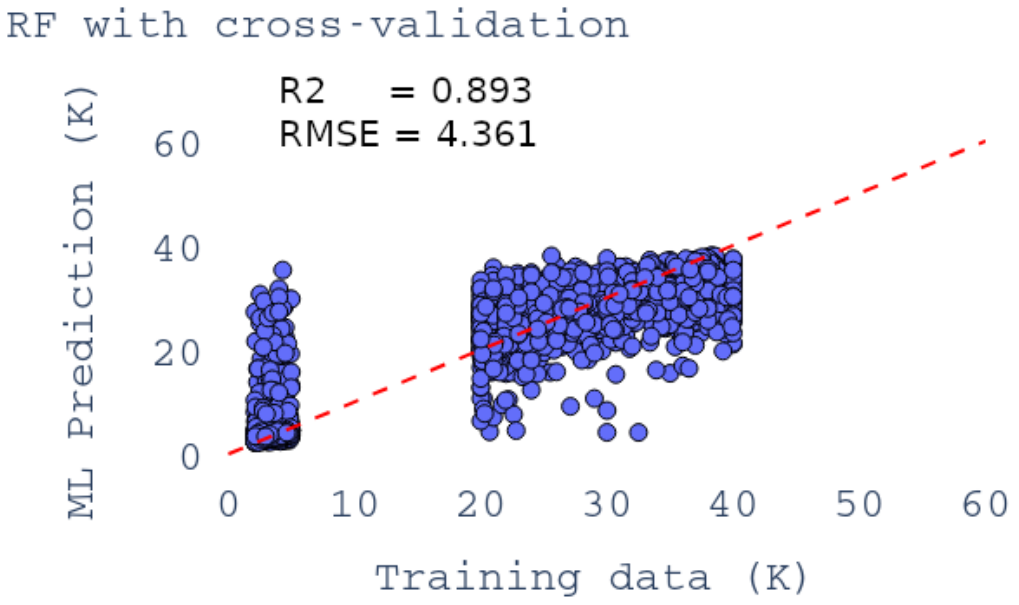

Figure 19: How Random Forest Regression fits to a new dataset

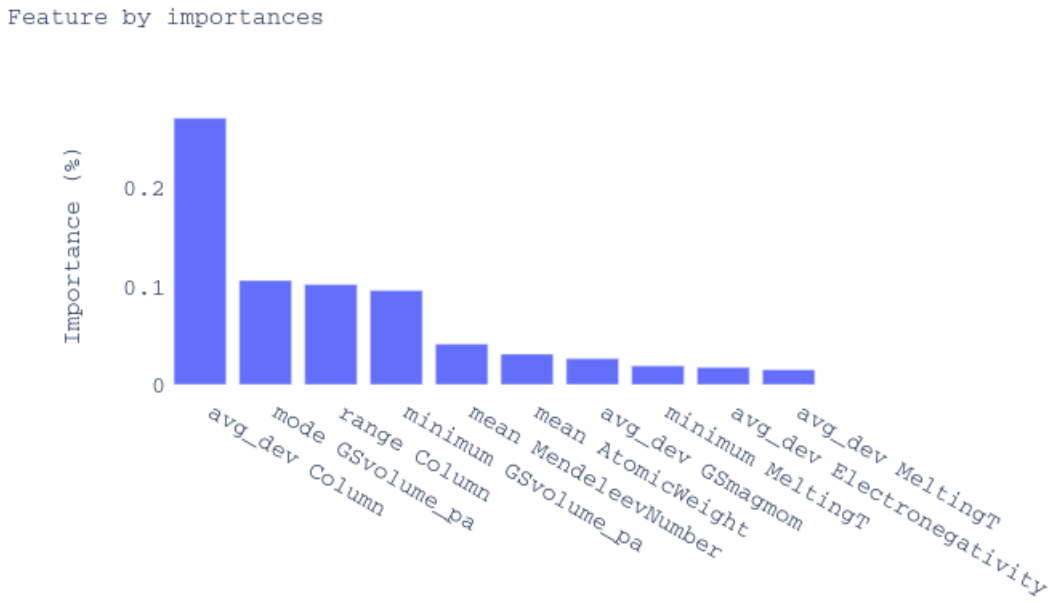

Figure 20: The most important features used by the random forest model

.

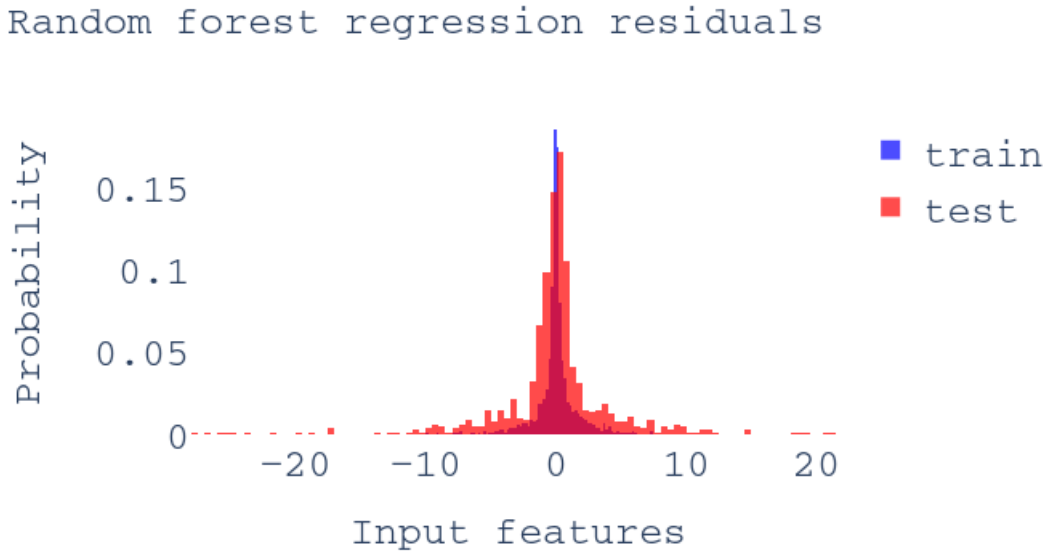

Figure 21: Visualization the training error

The most important descriptor/feature in the Random Forest regression for predicting T_c is the **average deviation of column of all the features**, see Fig. (20). The connection between T_c and this feature may offer new insight into the mechanism for superconductivity in this family. This shows predicting the transition temperature T_c of metallic materials affected by all descriptors we were used to train our Random Forest model. Such findings validate the ability of machine learning approaches to discover meaningful patterns that encode true physical phenomenon.

ICSD has only a small subset of materials in the *SuperCon* database. Therefore we applied our RF model to predict T_c for metallic materials included in our list of compounds [3.1.1.3] ($\sim$13000) that we extracted from the AFLOW database. The list was fed into the Random Forest regression model to predict the Debye temperature θ_D, density of states at the Fermi level $N(0)$ and the transition temperature T_c. Most of the materials are predicted to have T_c below $38K$. This is a good prediction and is not surprising, since the training dataset are metals and alloys. Some of the predicted materials in this list is compared to their known T_c and the model shows a good estimation of T_c with a small deviation. But, at the moment it is unknown whether some of these compounds are really superconducting materials. The list of compounds with the top twenty T_c predicted from our RF model is given below.

ICSD top twenty T_c List			
Compound Name	θ_D	$N(0)$	T_c
B2Mg1	623.464	1.759	38.072
Ba2Cu4O8Y1	193.339	2.053	34.953
Cu1O2Sr1	203.814	1.353	34.527
Ba2Cu3Eu1O7	175.297	3.339	34.514
Cu2Gd1O8Ru1Sr2	182.628	5.058	34.328
Ba6Ca6Cu9O29Tl5	165.822	8.015	34.107
Ba2Cu4Ho1O8	179.893	11.999	33.869
Ba2Cu4Er1O8	181.338	3.493	33.505
Ba2Cu1O5Tl1	144.462	5.354	32.753
Ca1Cu1O2	231.596	1.225	32.629
Ba2Cu3La1O7	175.729	4.326	32.458
Ba2Ca1Cu2O8Tl2	158.473	3.574	32.399
Ba2Ca1Cu2O7Tl1	160.586	3.717	31.859
Ca2Cu1O3	244.661	1.260	31.779
Ca3Cu1Ir1O6	199.898	4.084	31.741
Ba2Cu3Gd1O7	177.533	13.818	31.602
Cu2O7Sr2Tl1Y1	177.007	1.997	31.449
Ba2Cu3O7Y1	190.763	2.544	31.114
Ba2Cu3Dy1O7	172.531	3.849	30.979
Ca3Cl2Cu2O4	221.880	2.769	30.918
Fe2K1Se3	129.353	3.761	30.746

5 *Discussion and Conclusion*

Hither, three Machine Learning models are developed to estimate the Debye temperature θ_D and density of states near the Fermi level $N(0)$ using datasets from the AFLOW online repository [3.1.1.2] and [3.1.1.3]. Elemental properties are generated using the Matminer featurizer. Models like Linear Regression, Ridge Regression and Random Forest regression are developed to predict the value of θ_D and $N(0)$. The Random Forest regression model shows a good accuracy in fitting the training dataset compare to the others.

Next, using the dataset [3.1.1.1], elemental properties are generated from MAGPIE using a Matminer featurizer and we also employed our Random Forest models to estimate θ_D and $N(0)$ for these materials in the list. Then three successful models Linear Regression, Ridge Regression and Random Forest regression models are developed to predict the critical temperature T_c. The Random Forest regression model shows excellent performance with the training dataset having $R^2 \sim 0.98$. and a very good estimation with cross validation having $R^2 \sim 0.89$. By studying the importance feature, insight about a possible mechanism driving superconductivity is obtained.

Finally, we employed the Random Forest model on a dataset extracted from the ICSD database [3.1.1.3] of materials with non zero density of state at the Fermi level, to search for new inorganic superconductors. Most of these have no provided information during the training of the model. The model shows a good estimation for known superconductors with small deviation. The data used to generate these results and the prediction of T_c for ICSD materials can be downloaded from https://github.com/Firaskigali/Critical-Temperature-Tc

As a conclusion, this research demonstrates the advantages of us-

ing ML in superconductivity research and shows the ability of ML to make connection between material's properties and superconductivity. Although, all models in this research take as input a number of predictors generated from the elemental composition of each material, surprisingly, the Random Forest model is accurate. We suggest future research improve this predictive power by adding information about: space group, geometric structure, and phonon energies as well as using more sophisticated ML algorithms such as neural networks if possible.

6 Appendix

```
# Import necessary modules
import numpy as np
import pandas as pd

from matminer.featurizers.conversions import StrToComposition
from matminer.featurizers.composition import ElementProperty
from matminer.figrecipes.plot import PlotlyFig

from sklearn.linear_model import LinearRegression
from sklearn import model_selection
from sklearn.ensemble import RandomForestRegressor
from sklearn import linear_model

from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold, cross_val_score
from sklearn.model_selection import cross_val_predict

from sklearn import datasets
from sklearn.model_selection import train_test_split

from sklearn.datasets import make_regression

# Knowing your pandas version
print(pd.__version__)

# Here i give the path & import the data into pandas
dat = pd.read_csv("/home/firas/project/Dos1/Meta_Tc_pred_debye.csv") #
    Create a Dataframe from CSV
dat1 = pd.read_csv("/home/firas/project/Dos1/Doscar_pred_debye.csv")

# Drop a duplicate row, based on ENTRY name
dat = dat.drop_duplicates(subset='ENTRY', keep="first")
dat1 = dat1.drop_duplicates(subset='ENTRY', keep="first")

# Add some composition based features to our DataFrame.
data = StrToComposition().featurize_dataframe(dat, "ENTRY")
data1 = StrToComposition().featurize_dataframe(dat1, "ENTRY")

# Add some ElementProperties from featurizer
prop = ElementProperty.from_preset(preset_name="magpie")
data = prop.featurize_dataframe(data, col_id="composition",
   ignore_errors=True)
```

```
data1 = prop.featurize_dataframe(data1, col_id="composition")

# Get columns whose data type is object i.e. string
filteredColumns = data.dtypes[data.dtypes == np.object]
filteredColumns1 = data1.dtypes[data1.dtypes == np.object]
# list of columns whose data type is object i.e. string
listOfColumnNames = list(filteredColumns.index)
listOfColumnNames1 = list(filteredColumns1.index)
print(listOfColumnNames)
print(listOfColumnNames1)

# Try some different machine learning models to relate input features
    to the bulk modulus
# 1\ Try a Linear &Ridge regression model using scikit-learn
# Define the target and input features
y = data[' Tc'].values
excluded1 = ['ENTRY', ' Fermi', ' Dos', 'composition']
excluded = ['ENTRY', ' Tc', 'composition']
X = data.drop(excluded, axis=1)
Xnew1 = data1.drop(excluded1, axis=1)
X.fillna(X.mean(), inplace=True) #mean normalization
Xnew1.fillna(Xnew1.mean(), inplace=True) #mean normalization

# ML models
rr = linear_model.Ridge(alpha=.3)
lr = LinearRegression()
rr.fit(X, y),lr.fit(X, y)

# Get fit statistics
print('Linear training R2 = ' + str(round(lr.score(X, y), 4)))
print('Linear training RMSE = %.3f' % np.sqrt(mean_squared_error(
    y_true=y, y_pred=lr.predict(X))))
print('Ridge training R2 = ' + str(round(rr.score(X, y), 4)))
print('Ridge training RMSE = %.3f' % np.sqrt(mean_squared_error(y_true
    =y, y_pred=rr.predict(X))))

# How the data looks like on a plot
pf = PlotlyFig(x_title='Tc from experiment (K)',
               y_title='ML Prediction (K)',
               title='LR of T_c with magpie',
               mode='notebook',
               filename="lr_regression1.html")

pf.xy(xy_pairs=[(y,lr.predict(X)), ([0, 200], [0, 200])],
      labels=data['ENTRY'],
```

```
      modes=['markers', 'lines'],
      lines=[{}, {'color': 'red', 'dash': 'dash'}],
      showlegends=False

     )

pf = PlotlyFig(x_title='Tc from experiment (K)',
               y_title='ML Prediction (K)',
               title='RR of T_c with magpie',
               mode='notebook',
               filename="rr_regression1.html")

pf.xy(xy_pairs=[(y,rr.predict(X)), ([0, 200], [0, 200])],
      labels=data['ENTRY'],
      modes=['markers', 'lines'],
      lines=[{}, {'color': 'red', 'dash': 'dash'}],
      showlegends=False

     )

# Use 10-fold cross validation (90% training, 10% test)
crossvalidation = KFold(n_splits=10, shuffle=False, random_state=1)

scores_LR = cross_val_score(lr, X, y, scoring='neg_mean_squared_error'
    , cv=crossvalidation, n_jobs=1)
rmse_scores_LR = [np.sqrt(abs(s)) for s in scores_LR]
r2_scores_LR = cross_val_score(lr, X, y, scoring='r2', cv=
    crossvalidation, n_jobs=1)

scores_RR = cross_val_score(rr, X, y, scoring='neg_mean_squared_error'
    , cv=crossvalidation, n_jobs=1)
rmse_scores_RR = [np.sqrt(abs(s)) for s in scores_RR]
r2_scores_RR = cross_val_score(rr, X, y, scoring='r2', cv=
    crossvalidation, n_jobs=1)
print('Cross-validation results of LR:')
print('Folds: %i, mean R2: %.3f' % (len(scores_LR), np.mean(np.abs(
    r2_scores_LR))))
print('Folds: %i, mean RMSE: %.3f' % (len(scores_LR), np.mean(np.abs(
    rmse_scores_LR))))

print('Cross-validation results of RR:')
print('Folds: %i, mean R2: %.3f' % (len(scores_RR), np.mean(np.abs(
    r2_scores_RR))))
print('Folds: %i, mean RMSE: %.3f' % (len(scores_RR), np.mean(np.abs(
    rmse_scores_RR))))
```

```
120
121 # How the data looks like on a plot with cross validation
122 pf = PlotlyFig(x_title='Tc from experiment (K)',
123                y_title='ML Prediction (K)',
124                title='Generalization of LR of T_c with magpie ',
125                mode='notebook',
126                filename="lr_regression1.html")
127
128 pf.xy(xy_pairs=[(y, cross_val_predict(lr, X, y, cv=crossvalidation)),
       ([0, 200], [0, 200])],
129       labels=data['ENTRY'],
130       modes=['markers', 'lines'],
131       lines=[{}, {'color': 'red', 'dash': 'dash'}],
132       showlegends=False
133      )
134
135 pf = PlotlyFig(x_title='Tc from experiment (K)',
136                y_title='ML Prediction (K)',
137                title='Generalization of RR of T_c with magpie ',
138                mode='notebook',
139                filename="rr_regression1.html")
140
141 pf.xy(xy_pairs=[(y, cross_val_predict(rr, X, y, cv=crossvalidation)),
       ([0, 200], [0, 200])],
142       labels=data['ENTRY'],
143       modes=['markers', 'lines'],
144       lines=[{}, {'color': 'red', 'dash': 'dash'}],
145       showlegends=False
146      )
147
148 # 2\ Try a random forest model
149 rf = RandomForestRegressor(n_estimators=100, random_state=5)
150 rf.fit(X, y)
151 print('training R2 = ' + str(round(rf.score(X, y), 4)))
152 print('training RMSE = %.3f' % np.sqrt(mean_squared_error(y_true=y,
       y_pred=rf.predict(X))))
153
154
155 # How the data looks like on a plot with random forest
156 pf_rf = PlotlyFig(x_title='Tc from experiment (K)',
157                   y_title='ML Prediction (K)',
158                   title='RF of T_c with magpie',
159                   mode='notebook',
160                   filename="rf_regression1.html")
161
```

```
pf_rf.xy([(y, rf.predict(X)), ([0, 200], [0, 200])],
      labels=data['ENTRY'], modes=['markers', 'lines'],
      lines=[{}, {'color': 'red', 'dash': 'dash'}], showlegends=False)

# Compute cross validation scores for random forest model
r2_scores = cross_val_score(rf, X, y, scoring='r2', cv=crossvalidation
    , n_jobs=-1)
scores = cross_val_score(rf, X, y, scoring='neg_mean_squared_error',
    cv=crossvalidation, n_jobs=-1)
rmse_scores = [np.sqrt(abs(s)) for s in scores]

print('Cross-validation results:')
print('Folds: %i, mean R2: %.3f' % (len(scores), np.mean(np.abs(
    r2_scores))))
print('Folds: %i, mean RMSE: %.3f' % (len(scores), np.mean(np.abs(
    rmse_scores))))

final_model = rf.fit (X, y)

# How the data looks like on a plot with random forest and cross
    validation
pf_rf = PlotlyFig(x_title='Tc from experiment (K)',
                  y_title='ML Prediction (K)',
                  title='Generalization of RF for T_c with magpie',
                  mode='notebook',
                  filename="rf_regression1.html")

pf_rf.xy([(y, cross_val_predict(rf, X, y, cv=crossvalidation)), ([0,
    200], [0, 200])],
      labels=data['ENTRY'], modes=['markers', 'lines'],
      lines=[{}, {'color': 'red', 'dash': 'dash'}], showlegends=False)

# 1.3 Make a prediction with Random forest
df = pd.DataFrame(final_model.predict(X))
df1 = pd.DataFrame(final_model.predict(Xnew1))

# Add the predictions to our data
dat['pred_Tc']= df
dat1['pred_Tc']= df1

# Print the data to csv file

dat.to_csv('Meta_Tc_pred_Tc.csv')
dat1.to_csv('Doscar_pred_Tc.csv')

```

```
201
202 # Visualize the Probability of the prediction
203
204 X['ENTRY'] = data['ENTRY']
205 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.2, random_state=1)
206 train_formula = X_train['ENTRY']
207 X_train = X_train.drop('ENTRY', axis=1)
208 test_formula = X_test['ENTRY']
209 X_test = X_test.drop('ENTRY', axis=1)
210
211 rf_reg = RandomForestRegressor(n_estimators=50, random_state=1)
212 rf_reg.fit(X_train, y_train)
213
214 # Get fit statistics
215 print('training R2 = ' + str(round(rf_reg.score(X_train, y_train), 4))
    )
216 print('training RMSE = %.3f' % np.sqrt(mean_squared_error(y_true=
    y_train, y_pred=rf_reg.predict(X_train))))
217 print('test R2 = ' + str(round(rf_reg.score(X_test, y_test), 4)))
218 print('test RMSE = %.3f' % np.sqrt(mean_squared_error(y_true=y_test,
    y_pred=rf_reg.predict(X_test))))
219
220 pf_rf = PlotlyFig(x_title='Input features',
221                   y_title='Probability',
222                   title='Random forest regression residuals',
223                   mode="notebook",
224                   filename="rf_regression_residuals.html")
225
226 hist_plot = pf_rf.histogram(data=[y_train-rf_reg.predict(X_train),
227                                   y_test-rf_reg.predict(X_test)],
228                             histnorm='probability', colors=['blue', '
    red'],
229                             return_plot=True
230                            )
231 hist_plot["data"][0]['name'] = 'train'
232 hist_plot["data"][1]['name'] = 'test'
233 pf_rf.create_plot(hist_plot)
234
235
236 # Get the most important features used by the random forest model.
237 importances = rf.feature_importances_
238 # Included = np.asarray(included)
239 included = X.columns.values
240 indices = np.argsort(importances)[::-1]
```

```
241
242 pf = PlotlyFig(y_title='Importance (%)',
243                 title='Feature by importances',
244                 mode='notebook',
245                 fontsize=20,
246                 ticksize=15)
247
248 pf.bar(x=included[indices][0:10], y=importances[indices][0:10])
```

References

[1] Paul Drude. Zur elektronentheorie der metalle. *Annalen der physik*, 306(3):566–613, 1900.

[2] James F Annett et al. *Superconductivity, superfluids and condensates*, volume 5. Oxford University Press, 2004.

[3] John Bardeen. Ln cooper, and jr schrieffer. *Phys. Rev*, 108(1175):5, 1957.

[4] Philip B Allen and Marvin L Cohen. Pseudopotential calculation of the mass enhancement and superconducting transition temperature of simple metals. *Physical Review*, 187(2):525, 1969.

[5] WL McMillan. Transition temperature of strong-coupled superconductors. *Physical Review*, 167(2):331, 1968.

[6] Marc Peter Deisenroth, A Aldo Faisal, and Cheng Soon Ong. *Mathematics for machine learning*. Cambridge University Press, 2020.

[7] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[8] AFLOW. a globally available database of many materials compounds many calculated properties, available: $http://aflow.org/$, 2019.

[9] National Institute of Materials Science. Materials information station, supercon,. https://supercon.nims.go.jp/en/, 2020.

[10] Valentin Stanev, Corey Oses, A Gilad Kusne, Efrain Rodriguez, Johnpierre Paglione, Stefano Curtarolo, and Ichiro Takeuchi. Machine learning modeling of superconducting critical temperature. *npj Computational Materials*, 4(1):1–14, 2018.

[11] SR Xie, GR Stewart, JJ Hamlin, PJ Hirschfeld, and RG Hennig. Functional form of the superconducting critical temperature from machine learning. *Physical Review B*, 100(17):174513, 2019.

[12] Camilo E Calderon, Jose J Plata, Cormac Toher, Corey Oses, Ohad Levy, Marco Fornari, Amir Natan, Michael J Mehl, Gus Hart, Marco Buongiorno Nardelli, et al. The aflow standard for high-throughput materials science calculations. *Computational Materials Science*, 108:233–238, 2015.

[13] Frisco Rose, Cormac Toher, Eric Gossett, Corey Oses, Marco Buongiorno Nardelli, Marco Fornari, and Stefano Curtarolo. Aflux: The lux materials search api for the aflow data repositories. *Computational Materials Science*, 137:362–370, 2017.

[14] Guenter Bergerhoff, R Hundt, R Sievers, and ID Brown. The inorganic crystal structure data base. *Journal of chemical information and computer sciences*, 23(2):66–69, 1983.

[15] matminer. matminer is a python library for data mining the properties of materials. https://hackingmaterials.lbl.gov/matminer/, 2020.

[16] scikit learn. Simple and efficient tools for predictive data analysis. https://scikit-learn.org/stable/, 2020.

[17] NCSS Statistical Software Documentation. Chapter 335: Ridge regression. available:. https://ncss-wpengine.netdna-ssl.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Ridge_Regression.pdf, 2020.