Personal Programming Project 2023-24

Predicting Debye temperature in Metals using Machine Learning

TECHNISCHE UNIVERSITÄT
BERGAKADEMIE FREIBERG
Die Ressourcenuniversität. Seit 1765.

Harish Somaghatta

Computational Material Science

66018

Supervised by
Dr.-Ing. Arun Prakash
Course supervisors
Dr.-Ing. Arun Prakash and Dr.-Ing. Stefan Prüger

May 22, 2024

Contents

1 Abstract

The main goal of this project is to predict the Debye temperature of metals, a crucial component of the phonon spectrum of materials, using machine learning (ML) models. The raw dataset consists of various material properties of chemical compound's and elemental properties of atoms. Chemical compound's data taken from the open AFLOWlib repository and elemental properties of atoms obtained from the WebElements resource. In this approach, two different descriptors are created: one contains only information about a compound's chemical composition, while the other includes weighted elemental properties of atoms. Before building the machine learning models, the dataset is preprocessed by using techniques like encoding, filling the missing values of the dataset with the mean values, and feature selection techniques like principal component analysis. Initially, the material data is classified into metals and non-metals using classification algorithms like Artificial Neural Networks (ANN) with different optimization methods, Random Forest, and K-Nearest neighbours. Later, the Debye temperature for each metal is predicted using regression models like Artificial Neural Network (ANN) with various optimization algorithms, Gradient Boosting model, and Multiple Linear Regression in Python from scratch(using only numpy, matplotlib, os, sys libraries). After predicting the Debye temperature, the predicting accuracy of the models is compared using different metrics.

2 Introduction

At various temperatures and pressures, some of the thermal properties, like the coefficient of thermal expansion, specific heat capacity, etc., are to be known when using a material in any application. These properties can be calculated using two approaches: Model calculations and ab initio methods. Some of the model parameters like debye temperature, grüneisen parameter, etc., are required to perform model calculations. However, the debye temperature values of all the materials are not known. Ab initio methods depend only on crystal structure and atomic information. If the unit cell is large, ab initio requires intensive computational

power to calculate the lattice properties. In order to overcome the above drawbacks, machine learning models are used to determine the thermal properties, but it is not a universally applicable approach to predict the property value directly since it affects the prediction accuracy because it is based on predicting only at a specific temperature and pressure. To avoid this, machine learning models are used to predict model parameters like Debye temperature, Grüneisen parameter, etc., After finding the values of Debye temperature, thermal properties can be calculated using the Debye model at various ranges of temperatures and pressures. Details about the Debye temperature of numerous chemical compound's can be found in the AFLOWlib open database (https://aflowlib.org/). In this project, initially, the raw dataset is preprocessed, and then, the principal components that retain most of the information are considered using principal component analysis, following that, the metals data is classified from the materials data (metals and non-metals) using classification algorithms like Artificial Neural Network (ANN), Random Forest, and K-Nearest neighbours. The Debye temperature of metals is more significant due to electrons contribution. The thermal energy of the electrons increases with temperature, and they contribute more significantly to the thermal properties of metals. Thereafter, regression models like the Artificial Neural Network (ANN), Gradient Boosting Model, and Multiple Linear Regression are applied to the classified metals data, taken from the better-performing classification model, to predict the Debye temperature of metals. Based on the target prediction, the different algorithms are analyzed.

3 Theory

3.1 Specific Heat of Solids

3.1.1 Dulong-Petit law

According to Dulong-Petit law, a large number of solids have a constant heat capacity, which is unaffected by temperature.[1]

$$C = 3R^{[1]} \tag{3.1}$$

where C is heat capacity and R is ideal gas constant. But experimentally, the specific heat decreases and touches zero with a decrease in temperature up to zero absolute temperature. The Dulong-Petit law holds good for the materials at high temperatures, but it was unable to explain the drop in specific heat with the drop in temperature.

3.1.2 Einstein's model

Based on Planck's quantization assumption, Einstein's model successfully explained the behaviour of heat capacity with temperature by assuming that all the atoms are vibrating harmonically and oscillating at the same frequency called "Einstein's frequency". It was able to explain how the heat capacity depends on temperature. Einstein's model worked as expected at high temperatures same as Dulong-Petit law, but when the temperature reduces, the specific heat becomes zero very quickly. Despite the enormous success of Einstein's theory of specific heat, there were still deviations from the expected behaviour at low temperatures.[1]

3.1.3 Debye model

Debye model explained the specific heat behavior with temperature better than Einstein's model as Debye came to know that the atoms are oscillating same as the sound waves. Debye model assumes that the velocity of sound is same for the transverse and longitudinal modes[1], solids as continuum and also phonons with a range of frequencies unlike single frequency in Einstein's model. This model behaves approximately same as experimental observation at low and high temperatures in which at low temperatures, the specific heat proportional to T^3.[2]

The specific heat of a material in the Debye model can be determined using debye temperature as an important parameter. The debye temperature is the temperature at which the energy of the phonons reaches its maximum. From Fig. [1] it can be clearly seen that, In determining the specific heat, the Debye model

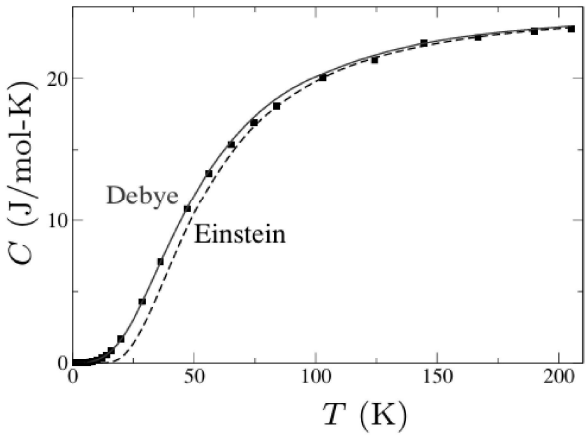


Figure 1: Silver's heat capacity in relation to the Debye and Einstein models. [1]

works better than Einstein's model especially at low temperatures and works same as the Dulong-Petit law at high temperatures.

3.2 Machine Learning Approach

Predicting thermal properties directly using a machine learning algorithm is not a universally acceptable approach since the prediction accuracy will be lower due to the consideration of single values of temperature and pressure. To overcome this, instead of predicting the thermal properties like specific heat, thermal conductivity etc., directly, predict the model parameters like Debye temperature, Grüneisen parameter. Once these parameters are found, thermal properties can be easily calculated at various ranges of temperatures and pressures. for example, By using the Debye temperature values, the specific heat can be calculated at different temperatures and pressures using debye model.[3]

At the debye temperature, the phonon modes in the crystal lattice are fully excited . In metals, phonons and electrons contribute to the lattice vibrations. Due to their dense lattice structure, metals have a large range of vibrational modes. The number of free electrons is huge in metals, which makes them contribute more

to the specific heat. Based on the previously mentioned considerations, the Debye temperature of metals is more significant than that of non-metals.

In this project, the raw dataset as shown in Fig. [2] comprises compound's data, including their corresponding parameters and properties such as space group, Grüneisen parameter, bulk modulus, Poisson ratio, heat capacity, thermal conductivity, thermal expansion, and material type, sourced from the AFLOWlib open database.[4] The compound's raw dataset consists of 5553 compound's with, 3950 as metals and 1603 as non-metals.

compound	spacegroup	grueneisen_parameter	bulk_modulus	poisson_ratio	heat_capacity	thermal_conductivity	thermal_expansion	material_type	debye_temp
Mn2Sc4Se8	227	1.90302	56.8447	0.345736	41.8433	1.46419	5.15E-05	Non-Metal	248.911
Ge1Ni2Zn1	225	2.45813	132.159	0.372802	11.9407	2.85205	6.21E-05	Metal	312.156
La6Ru8Sn26	223	2.27079	78.6194	0.329089	120.456	0.689631	4.69E-05	Metal	230.033
Ga1Pd2Sc1	225	2.28361	110.022	0.383655	12.0123	2.43747	4.98E-05	Metal	248.895
Ag2I2	129	2.49393	19.9855	0.327095	13.2897	0.562013	0.000149012	Non-Metal	133.151
As4Co2	58	2.31979	114.486	0.210609	16.7608	8.52199	4.85E-05	Metal	444.905
Cu4Ni1Yb1	216	2.26263	76.9393	0.335969	18.4007	1.8199	8.07E-05	Metal	250.133
Pt10Se8	14	2.4657	129.727	0.341654	53.8611	1.33708	4.00E-05	Metal	245.998
Cr4La4S12	62	2.03712	80.8486	0.265666	57.1027	2.92377	4.68E-05	Non-Metal	383.129
La8N4S6	58	1.96606	79.9316	0.272408	52.05	3.15013	3.94E-05	Non-Metal	335.046
Cs3In9	119	2.57809	15.0015	0.249797	41.4413	0.36912	0.000194347	Metal	148.99
Re4S4Te4	216	2.14379	121.89	0.247987	34.4004	5.67341	3.07E-05	Non-Metal	347.976
Cu4Sb4Se8	62	2.16354	52.6009	0.280119	48.4292	1.39627	6.64E-05	Non-Metal	255.048
Ag1Zr2	139	1.9271	90.8162	0.337855	8.88198	6.13645	3.89E-05	Metal	258.016
Nb2P2	141	1.96896	159.958	0.286052	10.8413	16.883	2.77E-05	Metal	469.053
Ca1S1	225	1.93521	52.2683	0.222083	5.6097	14.7572	5.85E-05	Non-Metal	436.099
Ca4Ge4Ni6	51	1.87681	73.2013	0.259377	40.4149	3.49629	5.30E-05	Metal	358.008
Cu1Hf1Hg2	216	2.50532	92.7529	0.426147	12.4233	0.638006	5.85E-05	Metal	134.009
Ga4Ho2Pd2	63	2.27119	88.0087	0.293613	23.9287	2.78277	5.52E-05	Metal	277.089
Ca2Ga4	194	2.2556	40.8836	0.165724	18.1104	3.93579	9.84E-05	Metal	341.195
Cd2Er4Ge4	127	2.05652	70.9255	0.239841	29.8026	3.36541	5.21E-05	Metal	269.963
S6Zn6	186	1.99227	63.2638	0.285296	35.0248	2.72643	6.21E-05	Non-Metal	348.987
Ge8	194	2.26676	52.905	0.199588	23.7605	3.63824	6.96E-05	Metal	329.987
Ho12Ru4	62	1.72595	67.5491	0.307572	47.9238	2.00312	3.97E-05	Metal	201.985

Figure 2: Raw dataset containing compound's data

Additionally, elemental properties such as Atomic Number, Symbol, Atomic Mass, Atomic Radius, and Electronegativity, obtained from WebElements, which can be seen in fig[3] are also included. Elemental properties raw dataset consists of 118 elements.

In this approach, as shown in Fig. [4], the project mainly consists of several key phases. Initially, the compound data containing Debye temperature information is collected from an extensive AFLOWlib online repository, and additionally, the necessary properties of all elements are obtained from the WebElements database.[5]

Following data collection, Data preprocessing includes addressing the missing values within the dataset, generating descriptors based on the compound chemical composition, and employing encoding techniques to make the data compatible with

AtomicNumber	Symbol	AtomicMass	AtomicRadius	Electronegativity
1	H	1.008	53	2.2
2	He	4.002	31	
3	Li	6.941	167	0.98
4	Be	9.012	112	1.57
5	B	10.81	87	2.04
6	C	12.011	67	2.55
7	N	14.007	56	3.04
8	O	15.999	48	3.44
9	F	18.998	42	3.98
10	Ne	20.1797	38	4.5
11	Na	22.99	190	0.93
12	Mg	24.305	145	1.31

Figure 3: Raw dataset containing elemental properties data

machine learning models. Subsequently, the dimensionality reduction technique (principal component analysis) transforms high-dimensional, correlated features into low-dimensional, uncorrelated features, retaining the most information.

In the modeling phase, various machine learning algorithms are used for classifying metals from materials and followed by, predicting the debye temperature of metals.

Finally, the performance of the different machine learning models are evaluated using different metrics based on their prediction accuracy.

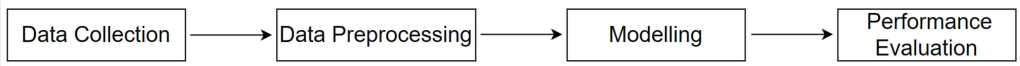


Figure 4: Steps involved in machine learning approach

4 Implementation of Data preprocessing

4.1 Project structure

The project is implemented as shown in the Fig [5].

where PCA means Principal Component Analysis. The primary objective of the project is to categorize the materials data into groups: metals and non-metals,

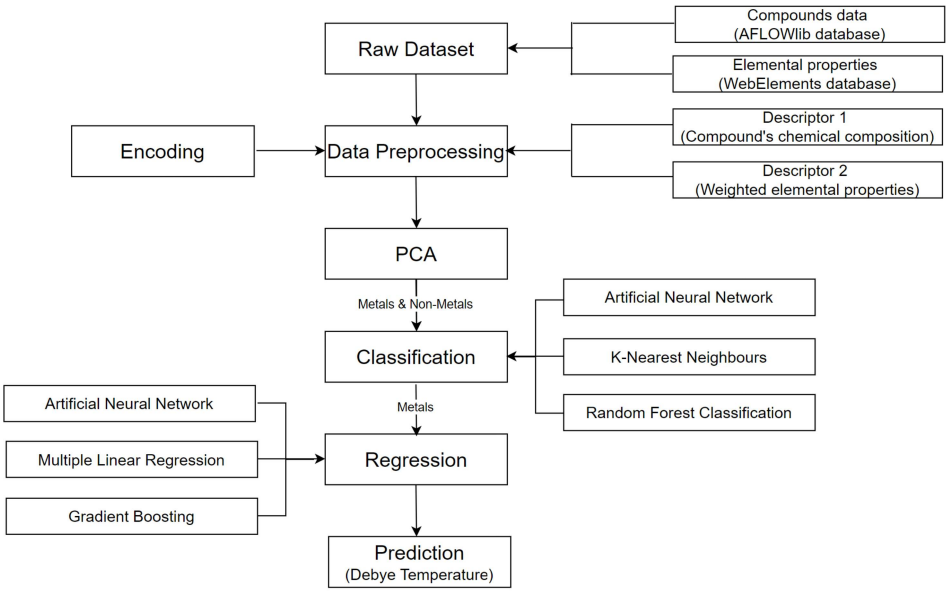


Figure 5: Structure of the project implemented

followed by, predict the Debye temperature specifically for metals. The individual steps outlined in the Fig [5] will be thoroughly addressed in the upcoming sections.

4.2 Data collection

Building a machine learning model requires data which can be sourced from various online open databases like AFLOW, NoMaD, Materials Project, OQMD, AiiDA etc., In this project, the data has been sourced from AFLOW open database which contains vast amount of compound's data. The necessary parameters and properties of the compound's for which the data is available are obtained through queries made to the LUX materials search API[6]. The elemental properties data for all the 118 elements are obtained from WebElements database[5].

4.3 Data pre-processing

The raw data that is obtained from databases is not pre-processed, contains missing values, and is noisy. The data preprocessing involves cleaning and transforming the data. Minimizing the GIGO (Garbage In, Garbage Out) is essential, i.e., reducing the poor quality data that gets into the model and minimizing the inaccurate results from the model. The variables in the raw dataset can be in different ranges, which could influence the model predictions. The variables with higher values will be given more importance than those with lower values, and the data has to be transformed so that all the variables are in the same range[7].

4.3.1 Descriptors generation

The trends and patterns in the data can be found using the descriptors. In this project, two descriptors are generated[5].

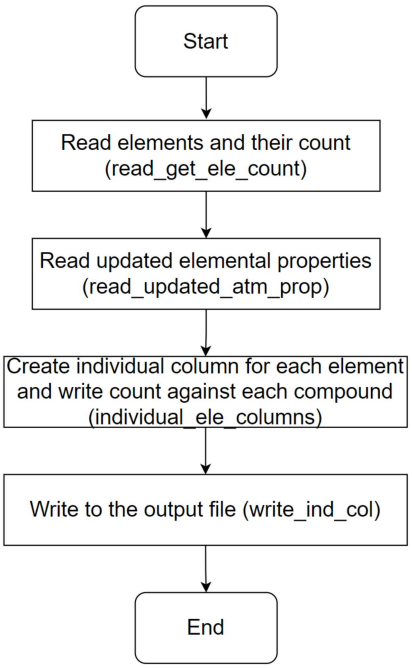
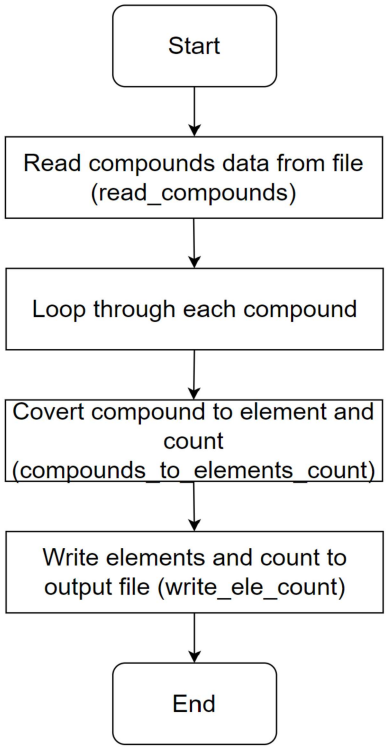
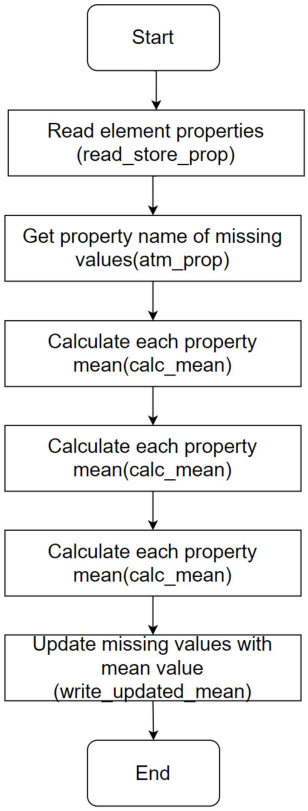


Figure 6: Flow chart of descriptor-1 coding approach

The first descriptor gives information about the chemical composition of the compound by creating individual columns for each element in the periodic system, and the numerical value of each compound for a particular column is equal to the number of atoms that the chemical formula of the compound has.



Compound chemical formula to element and their count



Missing values filled with corresponding property mean value

Figure 7: Supporting libraries for creating two descriptors

Code approach: No external libraries are used in this code(without numpy). Written a code to separate element and count from chemical formula of compound as in Fig. [7], which can be used as input file to create individual column for each element and store corresponding count of atoms against the specific compound formula.

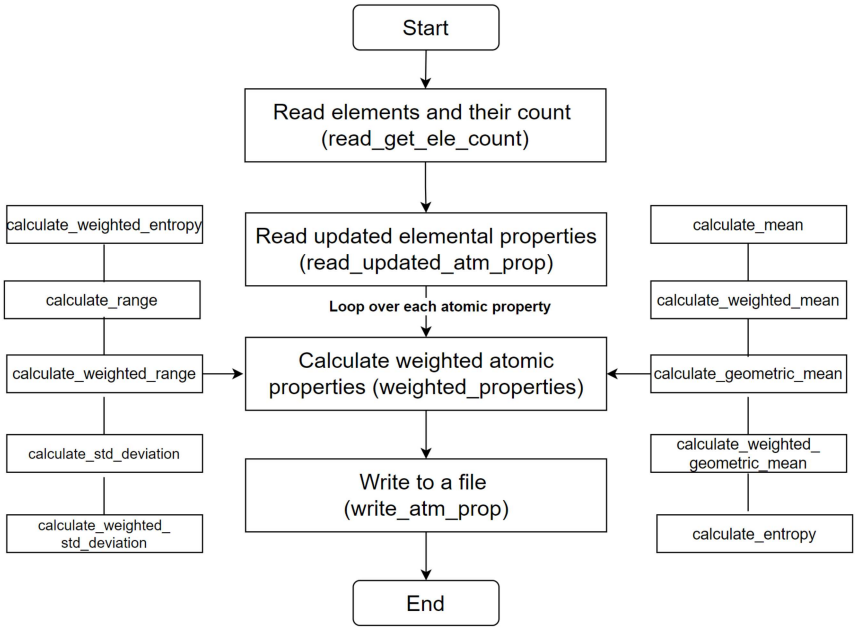


Figure 8: Flow chart of descriptor-2 coding approach

The second descriptor includes the information corresponding to the properties of the elements constituting the compound[8]. It can be explained clearly through an example. Consider a chemical formula of a compound $Mn_2Sc_4Se_8$. The atomic numbers of Mn, Sc and Se are taken as $t_1 = 25$, $t_2 = 21$ and $t_3 = 34$ respectively. The element count ratios in the compound formula are taken as $p_1 = 2/(2+4+8)$, $p_2 = 4/(2+4+8)$ and $3_3 = 8/(2+4+8)$ respectively. The atomic number ratios in the material are taken as $w_1 = t_1/(t_1+t_2+t_3)$, $w_2 = t_2/(t_1+t_2+t_3)$ and $w_3 = t_3/(t_1+t_2+t_3)$ respectively. From element count ratios and atomic number ratios, $A_1 = p_1w_1/(p_1w_1+p_2w_2+p_3w_3)$, $A_2 = p_2w_2/(p_1w_1+p_2w_2+p_3w_3)$, $A_3 = p_3w_3/(p_1w_1+p_2w_2+p_3w_3)$ respectively. The above example is explained for only atomic number property, the same has to be repeated for atomic mass, atomic radius, and electronegativity. The 10 features from the Fig. [9] will be used in this descriptor. Based on the described 10 features and 4 atomic properties a total of 40 features are generated.

code approach: No external libraries are used in this code (without numpy).

Feature	Formula
Mean(μ)	$(t_1 + t_2 + t_3)/3$
Weighted mean(κ)	$(p_1t_1 + p_2t_2 + p_3t_3)/(p_1+p_2+p_3)$
Geometric mean	$(t_1t_2t_3)^{(1/3)}$
Weighted geometric mean	$((t_1)^{p_1} . (t_2)^{p_2} . t_3^{p_3})^{(1/(p_1+p_2+p_3))}$
Entropy	-(w1ln(w1)+w2ln(w2)+w3ln(w3))
Weighted entropy	-(A1ln(A1)+A2ln(A2)+A3ln(A3))
Range	$\max(t_1, t_2, t_3) - \min(t_1, t_2, t_3)$
Weighted Range	$\max(p_1t_1, p_2t_2, p_3t_3) - \min(p_1t_1, p_2t_2, p_3t_3)$
Standard Deviation	$\sqrt{(((t_1-\mu)^2+(t_2-\mu)^2+(t_3-\mu)^2)/3)}$
Weighted Standard Deviation	$\sqrt{(((p_1t_1-\kappa)^2 + (p_2t_2-\kappa)^2 + (p_3t_3-\kappa)^2)/(p_1+p_2+p_3))}$

Figure 9: Features and formulas implemented in descriptor 2[8]

The missing values in the elemental properties data have been filled with mean of the available values in the corresponding property variable as in Fig. [7]. This file can be used as input for the weighted atomic properties descriptor.

4.3.2 Encoding

Encoding is mainly used to transform categorical variables into numeric format, which is compatible with machine learning models. Encoding technique gives more information about data within a variable to the machine learning algorithm.

Label encoding: The target feature for the classification which is material type(Metal or Non-metal) is encoded such that a unique integer 1 is assigned to metals and 0 is assigned to non-metals. The label encoding technique is used in this scenario since there is no order or hierarchy between the two categories. The space group numbers are also encoded with the label encoding technique so that it provides information to the machine learning model that there is no relation among the space groups.

One-hot encoding: Based on the number of elements in the compound, the compound chemical formula is one-hot encoded. A binary vector is created for each

compound, and each column in the vector corresponds to the number of elements in the compound's formula. If the compound's formula contains two elements, then the second column of the binary vector is active, i.e., equal to 1; otherwise, 0. This encoding technique provides chemical compound formula information to the machine learning model, which could improve its performance.

Ordinal encoding: The thermal conductivity variable has various ranges of values, and those values within the low range, specifically 0 to 10, are encoded as 0. Values within the medium range, spanning 10 to 100, are encoded as 1, and values in the high range, i.e., more than 100, are encoded as 2. Since the thermal conductivity ranges are in order, the ordinal encoding technique is used to make the model understand the patterns within this variable.

4.3.3 Element composition vector

Element composition vector provides insights into the contribution of individual element to the compound. This information enriches the model performance giving a deeper understanding about the compound formulas.

4.4 Principal Component Analysis(PCA)

Principal component analysis is a dimensionality reduction technique that transforms a high-dimensional correlated dataset into a low-dimensional uncorrelated dataset, retaining most of the data information or variance. It is challenging to conduct exploratory data analysis on a dataset with m samples and d variables; only two or three dimensions of data may be effectively seen. To depict n variables by taking into account 2 variables at a time, n (n-1)/2 scatter plots are needed, and the information is not understandable. Therefore, we would like to reduce the dimensionality of the dataset while preserving the maximum amount of information[9]. If we consider a raw dataset, not all the variables are equally significant. Variance gives information about how significant the variable is. In high-dimensional data, PCA finds the directions of maximum variance and projects the data onto a smaller dimensional space while preserving the essential informa-

tion. The new features that replace the original variables are called principal components, which are nothing more than a linear combination of the original d variables. Redundancy is eliminated by the orthogonality and uncorrelatedness between these principal components[9]. Initially, the number of principal components is equal to the number of variables in the raw dataset. However, the principal components that contain most of the variance will be retained, which in turn reduces the dimensionality. The first principal component holds the most information, followed by the next principal components.

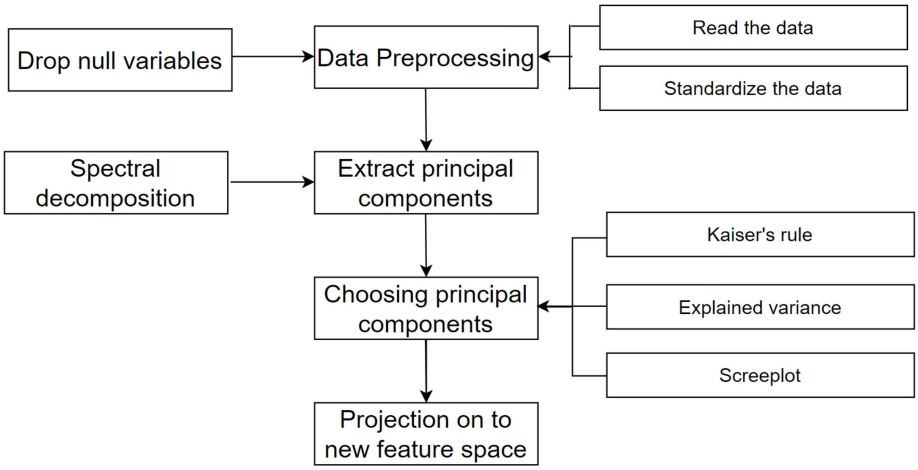


Figure 10: Implementation of Principal Component Analysis[9]

4.4.1 Data preprocessing in PCA

The data has to be preprocessed before performing principal component analysis. The first step in this preprocessing is to drop the variables that contain only null values. Fully null valued variables do not contribute any information, which would make the dimensionality reduction less effective. Once the non-null variables are imported, we need to normalize the variables as their range differ significantly from one another. The greater-range variables dominate the lower-range variables, which influence the outcome. Therefore, the variables should be normalized. In this project, we are using the Z-score standardization technique. It is given as

the difference between the variable value and the variable mean value and scaled by the standard deviation of the variable values[7]. By using standard deviation technique, the independent features are scaled.

$$Z - score = \frac{X - mean(X)}{SD(X)} \tag{4.1}$$

where X is variable value, SD refers standard deviation.

4.4.2 Principal components extraction

The eigenvectors of the covariance matrix are principal components. The new feature space's directions are given by the eigenvectors or principle components, while the variances of each principal component can be determined by the eigenvalues. The percentage of the total variance in the raw data set that can be attributed to the first k principal components is represented by the ratio of the sum of the first k eigenvalues to the sum of the variances of all d original variables. The eigenvectors can be computed by using either spectral decomposition or singular value decomposition (SVD) method. SVD is used in the majority of PCA software applications to increase computational efficiency[9]. The extraction of the principal components can be explained in detail as follows:

- Compute the covariance matrix of the scaled independent variables.
- Find eigenvectors and eigenvalues from the covariance matrix.
- Sort the eigenvalues in descending order and arrange the corresponding eigenvector accordingly.

Since the eigenvalue corresponds to the variance of that particular principal component, the eigenvector with the highest eigenvalue contains the maximum variance and is the first principal component. Eigenvalues play a key role in deciding the number of eigenvectors. The eigenvectors, which do not contain much information, can be neglected. The lower the eigenvalue, the less information is lost[9].

4.4.3 Choosing principal components

The dataset that is preprocessed consists of 172 independent variables; after performing PCA data preprocessing step, the count is reduced to 138. Now, the available non-zero-scaled variables are to be reduced to the k number of eigenvectors that have most of the variance. There is no universally accepted method to determine the optimal number of principal components. There are three methods to determine the number of optimal principal components[9].

- Kaiser's rule
- Variance explained criteria
- Visual examination of a scree plot

Kaiser's rule: One of the popular approaches is to determine how many principal components can be extracted from the dataset. This rule states that only principal components with a variance greater than one are retained. The principal component, which has an eigenvalue less than one, is considered to be less informative[9]. In this project, based on the kaiser's rule, the number of principal components chosen is 52, which have eigenvalues greater than one. This can be seen in the Fig. [11]

Variance explained criteria: Based on the threshold, the number of first principal components will be chosen. The principal components those variance explained is greater than threshold, will be selected. Proportion of the variance explained is given as the ratio of variance explained by each principal component to the total variance of all the principal components. As shown in the Fig. [12] If I consider the threshold value as 0.7%, then the principal components that are chosen are 60.

Visual examination of a scree plot: The number of principal components can be chosen by observing the scree plot. Look for the point at which the proportion of variance explained by each principal component that drop off. In the scree plot, this is frequently referred to as an elbow. In this project, from Fig. [13], the

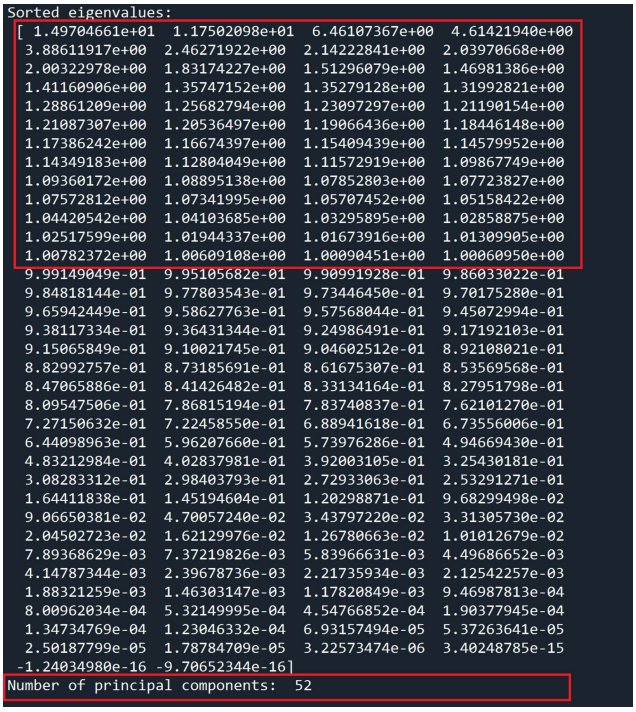


Figure 11: Number of principal components(Keiser's rule)

elbow point is around 10. Though it is hard to observe the elbow point out of 138 variables, from this method, 10 principal components are to be selected.

5 Implementation of Machine learning algorithms

The principal components from the principal component analysis contain information about both metals and non-metals, but this project aims to predict the debye temperature of metals specifically. So, the material data has to be classified into metals and non-metals using classification models (K-Nearest Neighbours, Random Forest, Deep Neural Network).

```
variance_explained_proportion:
[ 1.08481638e-01  8.51464476e-02  4.68193744e-02  3.34363724e-02
  2.81602839e-02  1.78457914e-02  1.55233943e-02  1.47804832e-02
  1.45161579e-02  1.32734947e-02  1.09634840e-02  1.06508251e-02
  1.02290512e-02  9.83675012e-03  9.80283538e-03  9.56469714e-03
  9.33776875e-03  9.10744884e-03  8.92009400e-03  8.78189524e-03
  8.77444250e-03  8.73452877e-03  8.62800262e-03  8.58305423e-03
  8.50624942e-03  8.45466647e-03  8.36300282e-03  8.30289509e-03
  8.28617267e-03  8.17420642e-03  8.08499411e-03  7.96143109e-03
  7.92465012e-03  7.89095202e-03  7.81542049e-03  7.80607444e-03
  7.79513132e-03  7.77840542e-03  7.65996028e-03  7.62017553e-03
  7.56670597e-03  7.54374533e-03  7.48520982e-03  7.45354169e-03
  7.42881155e-03  7.38727078e-03  7.36767507e-03  7.34129746e-03
  7.30307042e-03  7.29051508e-03  7.25293122e-03  7.25079345e-03
  7.24021050e-03  7.21091074e-03  7.18110093e-03  7.14516683e-03
  7.13636336e-03  7.08553292e-03  7.05395978e-03  7.03025565e-03
  6.99958296e-03  6.94657800e-03  6.93889887e-03  6.84835503e-03
  6.79795169e-03  6.78573438e-03  6.70280066e-03  6.64631958e-03
  6.63091195e-03  6.59436047e-03  6.55509067e-03  6.46455087e-03
  6.39849824e-03  6.32743255e-03  6.24402396e-03  6.18528673e-03
  6.13815859e-03  6.09729335e-03  6.03720409e-03  5.99965071e-03
  5.86628628e-03  5.70155937e-03  5.67928143e-03  5.52247297e-03
  5.26920748e-03  5.23520689e-03  4.99233057e-03  4.88084062e-03
  4.66738379e-03  4.32034536e-03  4.15924845e-03  3.58456109e-03
  3.50154336e-03  2.91911580e-03  2.84060221e-03  2.35818972e-03
  2.23393704e-03  2.16234633e-03  1.97777582e-03  1.83544399e-03
  1.19139013e-03  1.05213481e-03  8.71730949e-04  7.01666303e-04
  6.56993030e-04  3.40621188e-04  2.49128420e-04  2.40076616e-04
  1.48190379e-04  1.17485490e-04  9.18700459e-05  7.31975932e-05
  5.72006253e-05  5.34217265e-05  4.23164225e-05  3.25859893e-05
  3.00570539e-05  1.73680244e-05  1.60678213e-05  1.54016128e-05
  1.36464681e-05  1.06016773e-05  8.53774268e-06  6.86223053e-06
  5.80407271e-06  3.85615939e-06  3.29541197e-06  1.37955032e-06
  9.76338902e-07  8.91640089e-07  5.02288039e-07  3.89321479e-07
  1.81295506e-07  1.29554137e-07  2.33748894e-08  2.46557091e-17
 -8.98804202e-19 -7.03371264e-18]
Number of principal components:  60
```

Figure 12: Proportion of variance explained

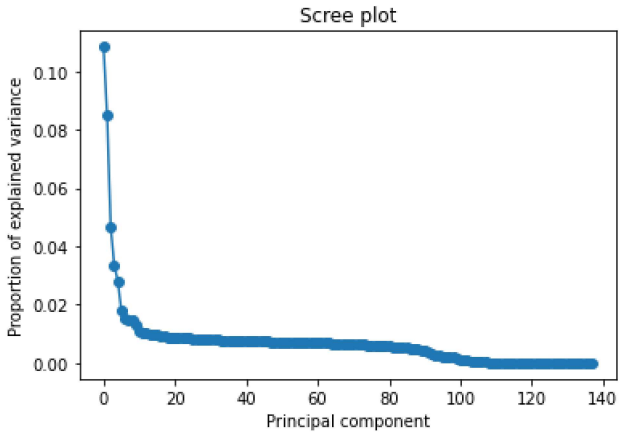


Figure 13: Visual examination of scree plot

5.1 K-Nearest Neighbours

In this project, the supervised machine learning technique K-Nearest neighbours is used for classification. K-Nearest Neighbours is an example of instance based learning, where the training data is stored and the test data can be classified by comparing it to all instances in the training dataset. The feature values of the new test data point are used as input during the classification, and the distance between the new test data point and every data point in the training dataset is calculated by the algorithm. The most often used distance function is Euclidean distance, which reflects how people typically think distance in the actual world[7]. The Euclidean distance (d) is given as,

$$d = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{5.1}$$

where x_i denotes feature values of test data point, y_i denotes feature values of training data point, and n is the number of features. Distances are arranged in ascending order once they have been calculated. The training dataset's 'k' data points, those with the shortest distances to the new test data points are chosen by the algorithm. Based on the majority voting, the class that is most common among the 'k' closest neighbours is assigned to the new test data point during the classification.

The disadvantages of this algorithm are that it can be computationally expensive to calculate the distances to every training data point for every prediction, particularly when working with large datasets, and a large amount of memory is needed to store all of the training data.

5.1.1 Hyperparameter tuning

To optimize the K-nearest neighbours classification model, the value of k has to be selected significantly. while making the prediction for a test data point, the k value denotes number of nearest neighbour to be considered. Therefore, Hyperparameter tuning is used to determine the relationship between the k value and the F1 score.

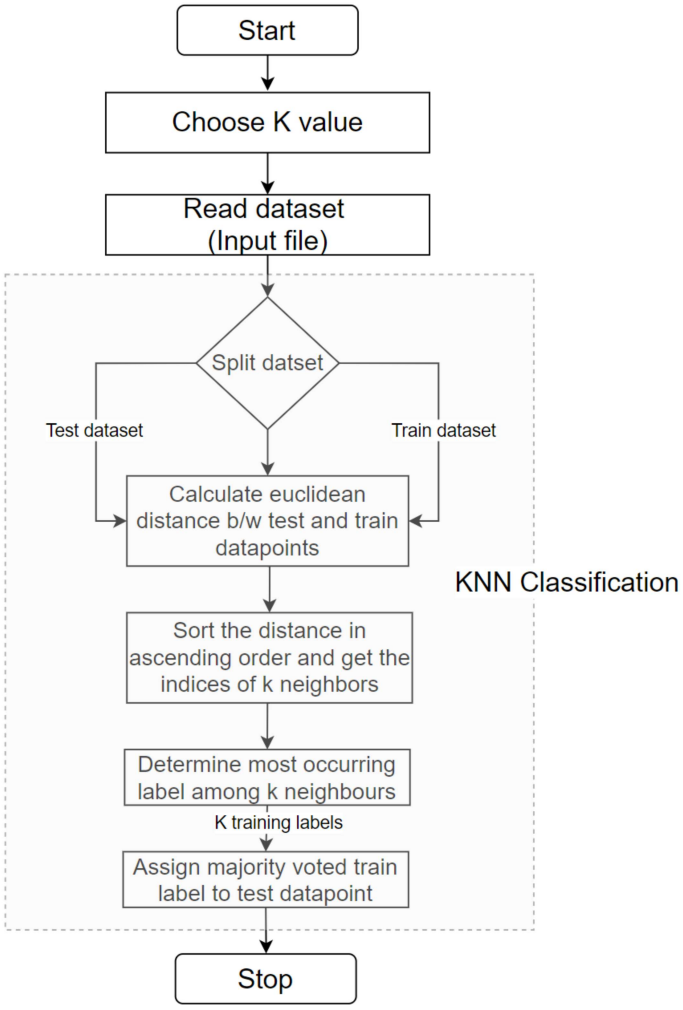


Figure 14: Flowchart of KNN classification algorithm

Using different values of k within the selected range, train the KNN model and evaluate the performance of the model using F1 score metric. The k value which performs better will be considered as the optimal value. Now, retrain the model on the entire dataset with the optimal k value. In this project, after tuning the k-value against F1 scores, the optimal value for k is 3.

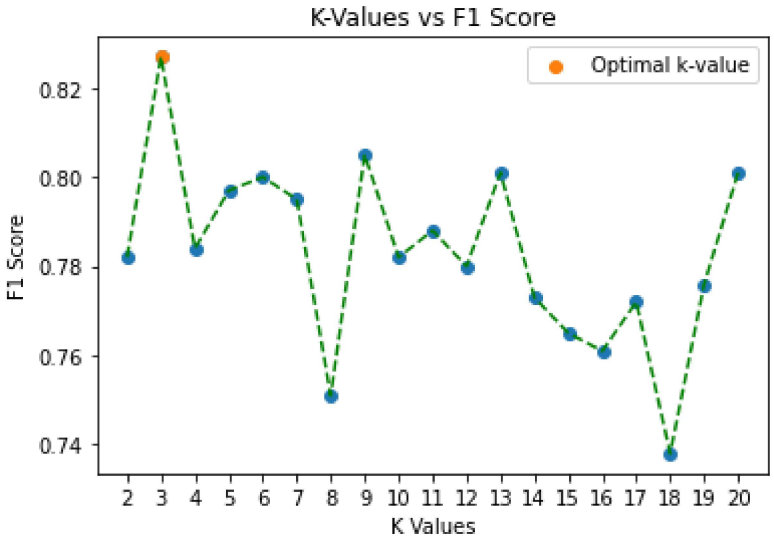


Figure 15: Hyperparameter Tuning of k neighbours

5.2 Classification and Regression Trees(CART)

The decision tree algorithm CART(Classification and Regression Trees) is used for both regression and classification tasks. It serves as the framework for the Random Forest classification algorithm and the Gradient Boosting Regression algorithm. It creates a tree where each node indicates a decision based on a feature by splitting the dataset into subsets depending on feature values. A splitting procedure that is recursive is used to construct the tree. The algorithm chooses the feature index and threshold(split value) at each node that produces the best split, which is determined by Gini impurity for classification and Mean Squared Error (MSE) for regression. The Gini impurity for classification is given by the equation (5.2),

$$\text{Gini}(S_q) = 1 - \sum_{k=1}^{K} (p_{k,q})^2 \tag{5.2}$$

where S_q represents the subset of dataset at node q, $P_{k,q}$ represents proportion of samples that correspond to class k and K is total number of classes[10]. When all instances fall into a single target class, the Gini index becomes zero, and when the cases are evenly distributed across all classes, it reaches its maximum. Once

the impurity is calculated, then Iterate through every feature in the dataset, taking consideration of possible split values according to the unique values of each feature. It splits the target feature into left and right nodes for each possible split, then computes the probability and impurity (Gini index or MSE) of each node. Using these probabilities and impurity, It calculates the Gini gain(g) given by the equation (5.3), where g(S_q) denotes Gini gain for the split of the subset, Gini(S_q) is gini impurity of the subset, $p_{L,q}$ denotes proportion of the samples at the left node, Gini(L_q) represents Gini impurity at left node and Gini(R_q) denotes Gini impurity at the right node. When a better split with a higher Gini gain is found, the function then modifies the best split parameters. In order to build decision trees, it finally returns the best split value, feature index, and Gini gain.

$$g(Sq) = \text{Gini}(Sq) - p_{L,q} \times \text{Gini}(L_q) - (1 - p_{L,q}) \times \text{Gini}(R_q) \tag{5.3}$$

The decision tree constructs a decision tree recursively. Initially, the algorithm checks whether the target features are all in the same class, whether the maximum depth has been reached, or whether there are fewer samples than the minimum required. The model returns a leaf node with the classification label or prediction value if any of these requirements are fulfilled. If not, find the best split value, feature index, and impurity of the best split. After splitting the dataset into left and right nodes, it calls a decision tree on each subset in a recursive manner to create the left and right subtrees. In the end, it returns a tuple that contains the split value, feature index, and left and right subtrees, which denotes a decision tree structure. Once the tree is built, in order to classify the target label during the classification task or predict the value during the regression of a test data point, initially the feature index and feature values of the test data point are extracted. Considering the feature index and split value of the tree, it determines whether to go to the left or right subtree based on the feature value of the data point. The algorithm continues until it reaches a leaf node, at which point it returns the class label or prediction value related to that leaf node.

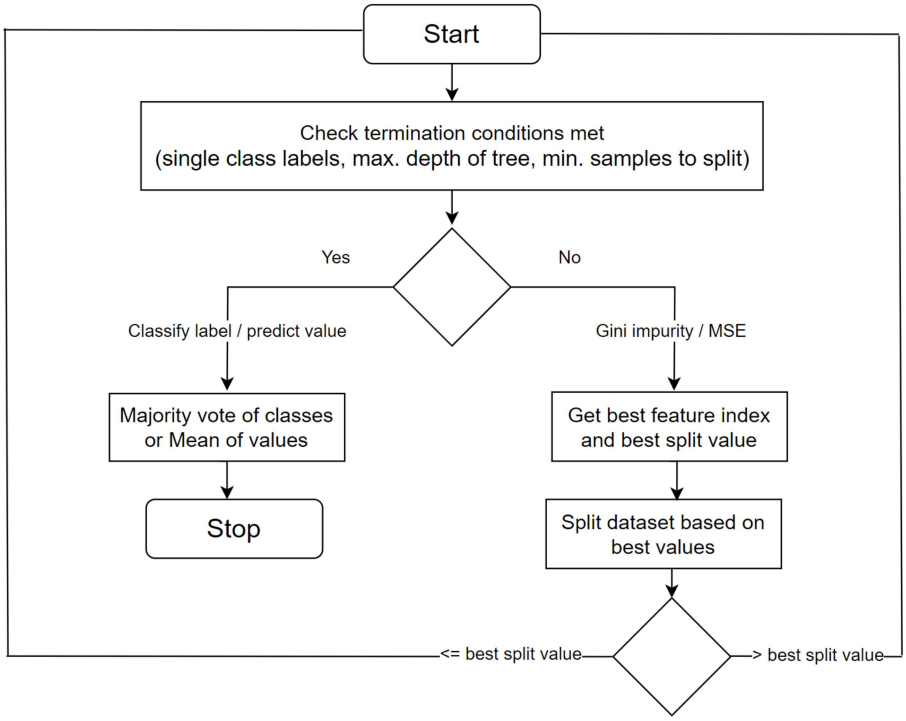


Figure 16: Process flow of classification and regression trees algorithm

5.2.1 Random Forest Classification

Random forest is a supervised machine learning algorithm. It is an ensemble learning technique for classification and regression tasks. In this project, it is used for the classification task. It constructs a number of decision trees during the training the model. The class that the majority of the trees predict is the random forest's predicted class during the classification task[11].

Initially, the algorithm creates bootstrapped samples by randomly selecting a subset of all samples from the training dataset with replacement. Because of this process, every decision tree in the forest has a different training dataset. The algorithm chooses a random subset of features, with replacement, from the initial set of features for each decision tree. This helps to avoid overfitting since every tree is trained on a different group of features. By using bootstrapping and random

feature subspace methods, every decision tree within the random forest is trained using distinct features and samples.

The algorithm creates bootstrapped samples from the training dataset for every tree in the forest. From these bootstrapped samples, a random subset of features is then chosen. The algorithm then constructs a decision tree for classification using the previously constructed cart(classification and regression trees) algorithm with these bootstrapped samples and random feature subspaces. The decision trees are constructed repeatedly and make up a random forest. To determine the final class label for each test data point, the model considers the most occurring predicted label(mode) from all the trees predictions.

5.2.2 Gradient Boosting for Regression

Gradient boosting is a machine learning approach based on boosting, used for regression tasks. It involves adding weak learners, like decision trees, to the ensemble sequentially in order to correct the errors made by the previous trees. Based on the specified number of trees, the iteration process continues. This approach of integrating several weak learners (trees) into a single model may result in an effective predictive model with gradient boosting.

The first step in this model is to initialize with a constant value, which is given by the equation (5.4), where $L(y_i, \alpha)$ refers to the loss function, y_i denotes true values, and α refers to predicted values over N samples. The predicted values that minimize the overall sum must be determined. From the equation (5.4), the prediction value that minimizes the overall sum is the mean of the true values. Therefore, we initialized the model with the mean of the true values.

$$F_0 = \arg\min_{\alpha} \sum_{i=1}^{N} L(y_i, \alpha) \tag{5.4}$$

In the next step, regression decision trees are constructed iteratively to create a gradient-boosting model. The derivative of the loss function w.r.t. the predicted value, which is called the residual, is to be computed and is given by the equation

(5.5), where Fx_i is the predicted value. The residual is to be calculated for the i-th sample at the m-th decision tree(iteration). Mean squared error (MSE) is considered as a loss function in this project.

$$r_{mi} = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\bigg|_{F(x_i)=F_{m-1}(x_i)} \tag{5.5}$$

The algorithm computes the residuals, i.e., the differences between the actual target values and the current predictions, for each iteration up to the specified number of trees. The following decision tree considers this residual as the target values and train on the independent features and these residuals. The result of any leaf in the tree is given as the mean of the residuals since we considered mean squared error (MSE) loss function. The prediction of the new sample is based on the previous prediction. If the model h_m fits the residuals exactly for a certain loss function (such as quadratic loss), the residuals become zero in the following iteration, making the process to end prematurely [12]. Using shrinkage to decrease each gradient decent step (5.6) is an appropriate method to regularize gradient boost with a learning rate ranging between 0 and 1. Here, F_m(x) is the prediction at current iteration, $F_m - 1$(x) refers to prediction of previous iteration, ρ denotes learning rate and h_m(x) denotes prediction from m-th decision tree. Over the time, accuracy is improved by reducing the impact of each tree on the final prediction with a small learning rate usually set to 0.1. This process is repeated until the specified number of trees is generated.

$$F_m(x) = F_{m-1}(x) + \rho_m h_m(x) \tag{5.6}$$

The trained gradient boosting regression model is used to predict values for the test dataset. Initially, the prediction is set to the mean of the target values from the training dataset. For each test data point, predictions are made using each decision tree in the ensemble. The prediction from each tree is scaled by the learning rate and added to the cumulative prediction from the previous trees. This iterative process, where each tree's prediction refines the overall prediction, continues until all trees in the ensemble have been used. The final prediction for

each test data point is the sum of the initial prediction and the scaled predictions from all the trees in the ensemble.

5.3 Artificial Neural Network

Artificial Neural Networks (ANN) draw inspiration from the architecture and concepts of the nervous system. ANN consists of an input layer, hidden layers, and an output layer. Hidden layers consist of neurons that transform the input into a form useful for output. The basic structure of ANN can be observed in Fig. [17].

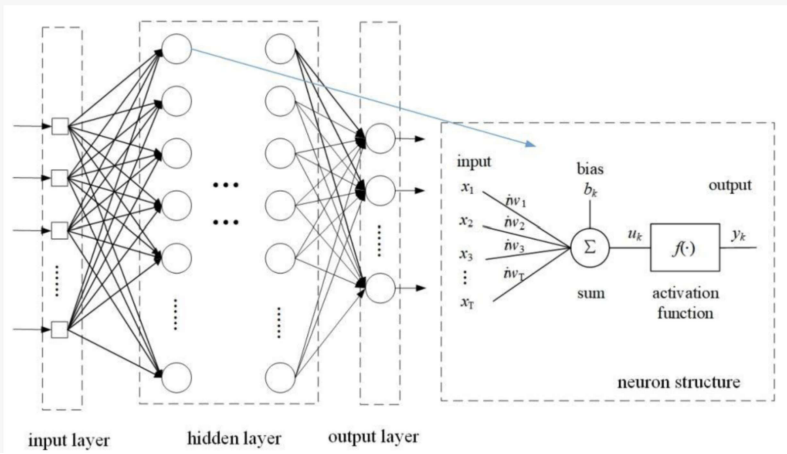


Figure 17: Structure of neural network[13]

In this project, a deep neural network with three layers is used to classify the materials into metals and non-metals, while a shallow neural network with only one hidden layer is used to predict the Debye temperature since it is performing better than the two hidden layered network. Neurons are the building blocks of the artificial neural network, which make up each layer. ANN mainly consists of multiplication, summation, and activation. Every neuron takes in input (x_1, x_2,..,x_i), processes it using an activation or transfer function to add non-linearity to the network, and then produces an output. Weights are to be adjusted in such a way as to minimize the error, i.e., the difference between predicted output and actual output.

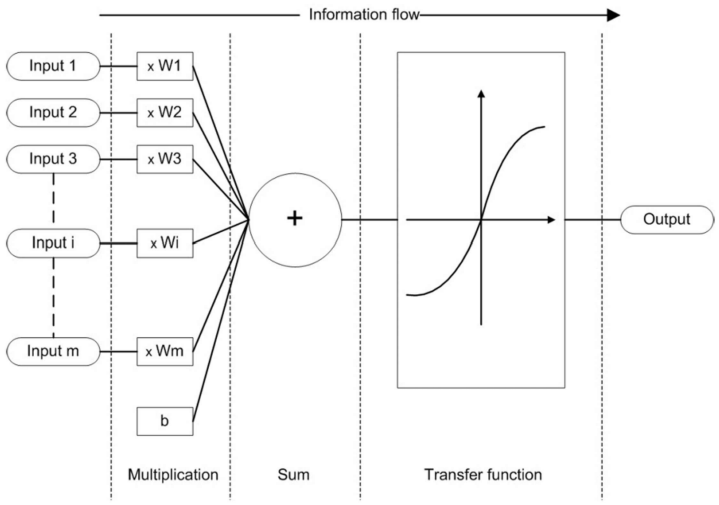


Figure 18: Working principle of Neuron [14]

Each input value is multiplied by the corresponding weight. Then, the multiplied value is added to the bias, and later, the weighted input and bias are given to the activation function. This is given by the equation (5.7) where $w_i(k)$ is weight, $x_i(k)$ is input value, b is bias, F is activation function, $y_i(k)$ is output value and i goes from 0 to m samples[14].

$$y_i(k) = F\left(\sum_{i=0}^{m} w_i \cdot x_i + b\right) [14] \tag{5.7}$$

5.3.1 Activation Functions

Activation functions in DNN(Deep Neural Network) determine the mapping between inputs and a hidden layer. Different activation functions are used at each layer in DNN. These activation functions are non-linear, which introduces non-linearity to the network. The different activation functions that are implemented in this project are ReLU(Rectified Linear unit), Tanh(Hyperbolic Tangent) function, Sigmoid function and Linear function.

1. ReLU (Rectified Linear unit): It is used as non-linear activation function in

the hidden layers which outputs maximum(0, input). When the input is negative value, then this function outputs zero, which can be observed in the figure [19]. The gradient of ReLU can be zero or some constant value. This function is mostly used in hidden layers[15].

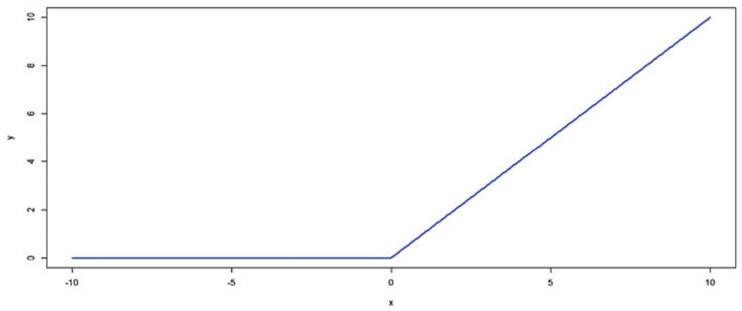


Figure 19: ReLU activation function [15]

2. Tanh (Hyperbolic Tangent): Tanh activation function is preferred to be in hidden layers since the output ranges between -1 and 1. Tanh activation function can be seen in the figure [20].It is given as $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$, where z is linear output. Tanh function has the advantage of handling negative numbers with greater ease[15].

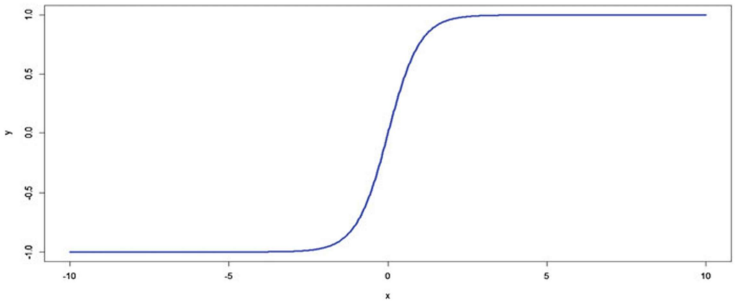


Figure 20: Tanh activation function [15]

3. Sigmoid function: Sigmoid function is mostly used in the final layer of the network. This function can be used for binary classification since the output of this function ranges between 0 and 1 and is given as $\sigma(z) = \frac{1}{1+e^{-z}}$, where z is linear output. This sigmoid function can be observed in the figure [21].

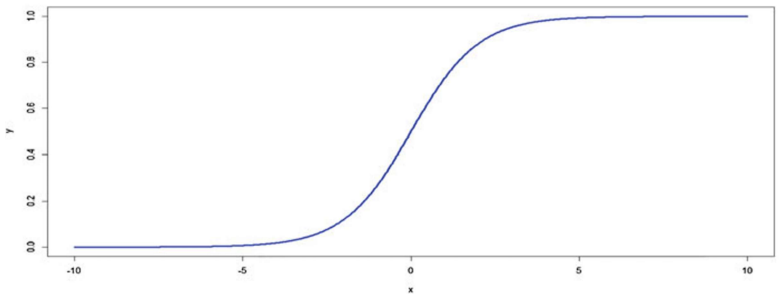


Figure 21: Sigmoid activation function [15]

4. Linear function: This function is mainly used for regression tasks. The output of the function is directly proportional to its input, without any transformation applied. The disadvantage of this function is that they do not introduce any non-linearity into the network. Neural networks lose their ability to discover complex patterns in the data if there is no non-linearity. Linear function is a straight line when plotted in graph and can be seen in the figure [22].

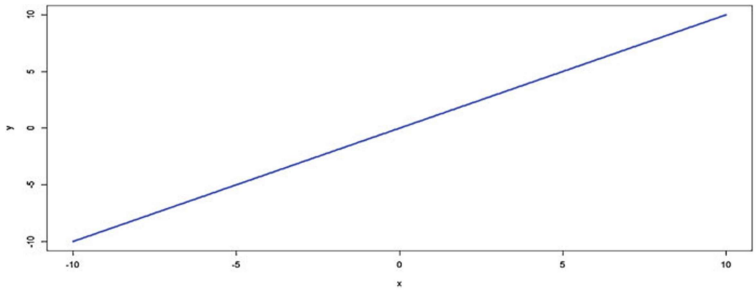


Figure 22: Linear activation function [15]

5.3.2 Forward Propagation

During forward propagation, the weights are initialized randomly while the bias is set to zero for the each layer in the neural network at starting. The input is fed to the input layer of the network and then it is given to the hidden neurons. As shown in the figure [18], In the linear forward module, each input value is multiplied by

its corresponding weight and then, the multiplied value is added to the bias term and is given by the following equation (5.8):

$$Z^{[i]} = W^{[i]} A^{[i-1]} + b^{[i]} \tag{5.8}$$

where $Z^{[i]}$ refers to the output of the i-th layer, $W^{[i]}$ denotes the weight of the i-th layer, $b^{[i]}$ represents the bias of the i-th layer and $A^{[i-1]}$ denotes previous layer activation. For the first layer, the $A^{[0]}$ would be input data. The activation function is applied to this output and is given by the equation (5.9):

$$A^{[i]} = g(Z^{[i]}) \tag{5.9}$$

The activation functions ReLU and Tanh are applied to the hidden layers for the classification and regression tasks. Sigmoid function is applied to the outer layer for classification while linear function is applied to the outer layer of the regression task.

5.3.3 Cost Function

The aim of the neural network is to learn the optimal parameters(weights and biases) that minimize the difference between the actual and predicted outputs. Different cost functions are used based on the task. Since DNN is used for both classification and regression tasks, cross-entropy cost function is used for classification task and mean square error is used regression task. This cost function is applied on the last layer to get the predicted values. The cross-entropy cost function[16] for the classification is given as:

$$\text{Cross-Entropy} = -\frac{1}{m} \sum_{i=1}^{m} \left[y_i \log(A^{[L](i)}) + (1 - y_i) \log(1 - A^{[L](i)}) \right] \tag{5.10}$$

where m is the number of examples, y_i is the true label for the i-th example, and $A^{[L](i)}$ is the predicted label for the i-th sample of the final layer L. The Mean

square error[16] for the regression is given as:

$$\text{Mean Squared Error} = \frac{1}{m}\sum_{i=1}^{m}(y_i - A^{[L](i)})^2 \tag{5.11}$$

where m is the number of samples, y_i is the actual value for the i-th example, $A^{[L](i)}$ is the predicted value for the i-th sample of the final layer L.

L2 regularization is used this model during computation of cost which prevent the overfitting in the model by adding a penalty term to the loss function. The penalty term is given as the sum of the squares of the parameters. In this project, only weight parameter has been considered during regularization. when the weights in the model are too large, this term will add penalty to the loss function. The L2 regularization or Ridge Regularization is given as:

$$Ł_2 = \lambda \sum_{i=1}^{n} w_i^2 \tag{5.12}$$

where L_2 refers to L2 regularization term, λ is the regularization parameter, which controls the degree of regularization, w_i is the i-th weight of the model, n denotes the total number of weights.

5.3.4 Backward propagation

To minimize the cost function during the training of a neural network, the parameters (weights and biases) need to be updated iteratively. For this, the partial derivative of this cost function with respect to the parameters is to be computed. The predicted values ($A^{[L]}$) from the last layer and true label/value (y) are used as an input to compute the partial derivative of cost function w.r.t activation function in the outer layer and is given as[17]:

$$\text{dAL} = -\frac{y}{A^{[L]}} + \frac{1-y}{1-A^{[L]}} \quad \text{(Classification)} \tag{5.13}$$

$$\text{dAL} = -2 \times (y - A^{[L]}) \quad \text{(Regression)} \tag{5.14}$$

The derivative of the cost function w.r.t linear output is to be determined to compute the partial derivative of cost function w.r.t parameters(weights and bias). The equations for the above discussed are given as:

$$dZ = g'(AL) \tag{5.15}$$

$$dA^{[l-1]} = W^{[l]T} dZ^{[l]} \tag{5.16}$$

$$dW^{[l]} = \frac{1}{m} dZ^{[l]} A^{[l-1]T} \tag{5.17}$$

$$db^{[l]} = \frac{1}{m} \sum_{i} dZ^{[l](i)} \tag{5.18}$$

where g'(AL) denotes derivatives of the activation function, sigmoid and linear functions as outer layers for classification and regression, respectively, and ReLU as hidder layers for both of the tasks.

5.3.5 Optimization algorithms

From the back propagation step, the gradients of weight and bias are computed and are used in this step to update the weights and bias in order to minimize the cost function. These parameters can be updated using various algorithms.

1. Stochastic Gradient Descent(SGD): This algorithm minimizes the cost function by updating the parameters iteratively. A mini-batch of m samples from the training set and the corresponding target is randomly chosen at each iteration[18]. The update equation is given as

$$\theta = \theta - \epsilon_k \hat{g} \tag{5.19}$$

where θ refers to parameters (weights and bias), ϵ_k refers to the learning rate, and $\hat{g}$ is the gradient of the loss function with respect to the parameters. The

drawbacks of this algorithm are that it consumes longer time to train a neural network on large dataset and the learning rate has to be tuned manually. To overcome this, advanced optimization algorithms are introduced, which will be explored further.

2. Stochastic Gradient Descent with Momentum: The main goal of the SGD with momentum method is to solve the oscillation and slower convergence issues that the SGD experienced. A variable called velocity is introduced to check how speed the parameters are moving through the parameter space[18]. A new hyper parameter which ranges between 0 and 1 called momentum parameter(α) is introduced by this algorithm. The update equation for Stochastic gradient descent with momentum is given as:

$$w^{[j]} = w^{[j]} - \alpha \cdot \hat{v}_{\mathrm{dw}}^{[j]} \tag{5.20}$$

$$b^{[j]} = b^{[j]} - \alpha \cdot \hat{v}_{\mathrm{db}}^{[j]} \tag{5.21}$$

where $dw^{[j]}$ and $db^{[j]}$ represents gradients of weights and biases for jth layer respectively. $\hat{v}_{\mathrm{dw}}^{[j]}$ and $\hat{v}_{\mathrm{db}}^{[j]}$ are velocities of weights and biases for j-th layer, respectively.

3. Root Mean Square Propagation(RMSProp): An adaptive learning rate optimization algorithm which modifies the learning rate during training. An exponentially weighted moving average of previous squared gradients are used to adjust the learning rate. The goal of RMSProp is to gradually decay the contribution of extreme previous gradients in order to reduce their significance[18]. RMSProp achieves an optimal deep neural network by combining adaptability and efficiency by giving priority to recent gradient information. The square gradients is given as:

$$s_{dw}[k] = \rho \cdot s_{dw}[k] + (1-\rho) \cdot (dw[k])^2 \tag{5.22}$$

$$s_{db}[k] = \rho \cdot s_{db}[k] + (1-\rho) \cdot (db[k])^2 \tag{5.23}$$

The parameters update is given as:

$$w[k] = w[k] - \alpha \cdot \frac{dw[k]}{\sqrt{s_{\mathrm{dw}}[k] + \epsilon}} \tag{5.24}$$

$$b[k] = b[k] - \alpha \cdot \frac{db[k]}{\sqrt{s_{\mathrm{db}}[k] + \epsilon}} \tag{5.25}$$

where ρ is RMSprop parameter, $s_{dw}[k]$ and $s_{db}[k]$ refer to squared gradients for weights and biases, respectively. ϵ is a small constant value to avoid division by zero error.

4. Adam optimization: Adam is a widely used deep learning algorithm for adaptive optimization and the word Adam is derived from 'adaptive moment estimation'. By using the estimates of the gradients first and second moments for different parameters, the individual adaptive learning rates are calculated. It considers exponentially weighted average of previous gradients and also corrected ones of the same. In the same manner, it includes squares of the past gradients and it's corrected ones. The weights and biases are updated as:

$$\hat{v}_{\mathrm{dw}}[i] = (\alpha \cdot \hat{v}_{\mathrm{dw}}[i]) + ((1-\alpha) \cdot \mathrm{dw}[i]) \tag{5.26}$$

$$\hat{v}_{\mathrm{dw}}^{\mathrm{corrected}} = \frac{\hat{v}_{\mathrm{dw}}[i]}{1-(\alpha^t)} \tag{5.27}$$

$$\hat{v}_{\mathrm{db}}[i] = (\alpha \cdot \hat{v}_{\mathrm{db}}[i]) + ((1-\alpha) \cdot \mathrm{db}[i]) \tag{5.28}$$

$$\hat{v}_{\mathrm{db}}^{\mathrm{corrected}} = \frac{\hat{v}_{\mathrm{db}}[i]}{1-(\alpha^t)} \tag{5.29}$$

$$\hat{s}_{\mathrm{dw}}[i] = (\rho \cdot \hat{s}_{\mathrm{dw}}[i]) + ((1-\rho) \cdot (\mathrm{dw}[i]^2)) \tag{5.30}$$

$$\hat{s}_{\mathrm{dw}}^{\mathrm{corrected}} = \frac{\hat{s}_{\mathrm{dw}}[i]}{1-(\rho^t)} \tag{5.31}$$

$$\hat{s}_{\mathrm{db}}[i] = (\rho \cdot \hat{s}_{\mathrm{db}}[i]) + ((1-\rho) \cdot (\mathrm{db}[i]^2)) \tag{5.32}$$

$$\hat{s}_{\mathrm{db}}^{\mathrm{corrected}} = \frac{\hat{s}_{\mathrm{db}}[i]}{1-(\rho^t)} \tag{5.33}$$

$$w[i] = w[i] - \alpha \cdot \left(\frac{\hat{v}_{\mathrm{dw}}^{\mathrm{corrected}}}{\sqrt{\hat{s}_{\mathrm{dw}}^{\mathrm{corrected}} + \epsilon}} \right) \tag{5.34}$$

$$b[i] = b[i] - \alpha \cdot \left(\frac{\hat{v}_{\mathrm{db}}^{\mathrm{corrected}}}{\sqrt{\hat{s}_{\mathrm{db}}^{\mathrm{corrected}} + \epsilon}} \right) \tag{5.35}$$

where t is time steps taken by Adam, ρ and α are hyper parameters, ϵ is a small number to avoid divide by zero. As per [19], ρ is taken as 0.9 and α value is 0.99.

Learning rate decay is implemented for any optimization algorithms during training in order to improve the convergence and performance of the model. The learning rate decreases over training epochs and it is given as:

$$\text{learning rate decay} = \frac{\alpha_0}{1 + (\text{decay rate} \times \text{epoch})} \tag{5.36}$$

The different optimization algorithms and the corresponding learning rates are considered by conducting hyperparameter tuning which results in minimum cost and highest accuracy which can be observed in the table 1.

Optimization Algorithms	Learning Rate	Decay Rate
SGD	0.2008810707215905	0.3
SGDM	0.2008810707215905	0.3
RMSProp	0.00354728214309162	0.3
Adam	0.000354728214309162	0.1

Table 1: Optimization Algorithms with Corresponding Learning Rates and Decay Rates

5.4 Multiple Linear Regression

Multiple linear regression is a linear regression with multiple variables. It gives the relationship between two or more independent features and a dependent feature(Target variable)[20]. The equation for the multiple linear regression is:

$$\hat{y}_i = \sum_{j=1}^{p} w_j x_{ij} + b \tag{5.37}$$

where $\hat{y}$ is the predicted value for i-th sample, b is the intercept (bias term), w_j are the coefficients associated with each independent variable x_{ij}, p is total number of independent features. The primary objective of this model is to minimize the cost function, the cost function, which brings the predictions closer to the true values. The cost function used in this model is given as:

$$\text{Cost} = \frac{1}{2n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{5.38}$$

where n is total number of sample, y_i is actual value, $\hat{y}$ is the predicted value. The model first initializes the weights to a zero vector and sets bias term to zero. In order to minimize the cost function, gradient of loss with respect to parameters need to computed. These gradients are used to update the parameters(weights and biases) iteratively using gradient descent method. The gradient of loss with respect to parameters are given as:

$$\nabla w = \frac{1}{n} \sum_{i=1}^{n} x_i^T \cdot (\hat{y}_i - y_i) \tag{5.39}$$

$$\nabla b = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i) \tag{5.40}$$

where ∇w and ∇b represent the gradients of the loss function with respect to the weights and bias, respectively. n is the total number of samples in the dataset, x_i^T denotes the transpose of the feature variable for the i-th sample, $\hat{y}_i$ is the predicted value, and y_i is the actual value. These gradients are used to update the parameters (weights and biases) using the gradient descent optimization method. The parameter update is given as:

$$w^{(i+1)} = w^{(i)} - \alpha \cdot \nabla w^{(i)} \tag{5.41}$$

$$b^{(i+1)} = b^{(i)} - \alpha \cdot \nabla b^{(i)} \tag{5.42}$$

where $w^{(i+1)}$, $b^{(i+1)}$ represent weight and bias respectively at iteration i+1, $w^{(i)}$,$b^{(i)}$ denote weight and bias respectively at iteration i, α is learning rate, $\nabla w^{(i)}$, $\nabla b^{(i)}$ denote gradient of loss w.r.t parameters. From this gradient descent method, the parameters which minimize the cost function are considered as the optimal parameters. These optimal parameters (weights and biases) can be used to predict the value for the test dataset.

5.5 Evaluation Metrics

Evaluation metrics are used to evaluate the performance of the machine learning models. Different evaluation metrics are used for classification and regression.

The combinations of predicted and actual(true) values are given by the confusion matrix[21] and is given as:

	Actual Positive	**Actual Negative**
Predicted Positive	True positive (TP)	False positive (FP)
Predicted Negative	False Negative (FN)	True Negative (TN)

Table 2: Confusion Matrix

5.5.1 Accuracy

Accuracy measures the proportion of instances that are correctly classified out of all the instances. It is given as the ratio of the number of correct predictions to the total number of predictions. It is a useful metric when the dataset is well balanced, there might be a chance of being misleading if the dataset is biased or imbalanced. Therefore, other metrics are used to measure the performance of the model.

5.5.2 Precision

Precision measures the proportion of actual positives to the number of predicted positives.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{5.43}$$

5.5.3 Recall

Recall measures the proportion of number of true positives to the total number of actual positives.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{5.44}$$

5.5.4 F1 score

F1 Score is the harmonic mean of precision and recall. It is an effective evaluation metric for classification.

$$\text{F1 Score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.45}$$

5.5.5 Mean Absolute Error (MAE)

Mean Absolute Error (MAE) is a regression metric, measures the average of the absolute differences between actual values (y_i) and predicted values ($\hat{y}_i$) of n samples. Lower value of MAE indicates better performing model[22].

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{5.46}$$

5.5.6 Root Mean Squared Error (RMSE)

Root Mean Squared Error (RMSE) is used regression metric that measures the square root of the average squared differences between actual ($\hat{y}_i$) and predicted values ($\hat{y}_i$) of n number of examples. Lower the value of RMSE, better the model performance.

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2} \tag{5.47}$$

5.5.7 R squared(Coefficient of Determination)

R^2 score is a metric that evaluates the performance of a regression model by comparing it to the mean line, indicating how well the model predicts the dependent variable. Higher R^2 indicates better performing model.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{5.48}$$

where y_i are the actual values, $\hat{y}_i$ are the predicted values. $\bar{y}$ is the mean of the actual values. n is the total number of examples.

5.6 Proposed and Implemented tasks

Tasks	**Proposed**	**Implemented**
Descriptors generation	Yes	Yes
Encodings	Yes	Yes
Principal Component Analysis	Yes	Yes
Implementation of classification and regression models using only numpy, matplotlib, os, sys libraries	Yes	Yes
Optimization algorithms for ANN	Yes	Yes
Unit and functionality tests for implemented functions	Yes	Yes
Hyperparameter tuning	Yes	Yes
Gradient check for backpropagation of ANN	No	Yes

Table 3: Proposed and Implemented Tasks

5.7 Software and Libraries versions

In this project, python programming language is used for data preprocessing and implenting the models. The version of the python used is 3.10.7. The libraries used in this project and the versions are given in the following table:

6 Tests performed

Unit tests and functional tests are performed on all the functions which can be seen the Fig. [23]

Library	Version	Usage
numpy	1.24.3	For creating and working with multidimensional arrays
sys	3.9.13	Used while creating the relative path to access the files
os	3.9.13	Navigate through directory structures
pytest	7.1.2	Implemented functions are tested
matplotlib	3.5.2	To plot the graphs
pdoc3	3.10.7	To generate string comments of all functions

Table 4: Versions and Usage of Python Libraries

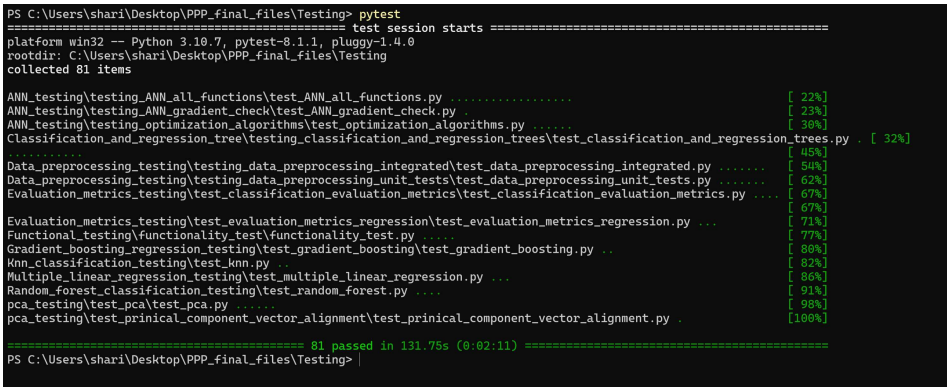


Figure 23: Functional and unit tests

6.1 Test the data preprocessing and feature engineering functions

Unit tests and integrated tests are performed to verify various functions used in data preprocessing and feature engineering for machine learning models. By comparing the obtained outputs against the expected output, each test aims to verify that a certain function works as expected given the inputs. While performing integrated testing, the test reads the data from the input file and verifies whether the expected output matches the obtained outcome. Pytest framework to assert the truthfulness of various data preprocessing and feature engineering functions.

6.2 Test Principal Component Analysis

One of the methods to test the principal component analysis is that the sum of the eigenvalues obtained from the covariance matrix should be equal to the total variance of the data given for analysis[9]. It makes sure that the total variance of the original data is retained after principal component analysis (PCA). The implementation of PCA has been successfully verified with this test.

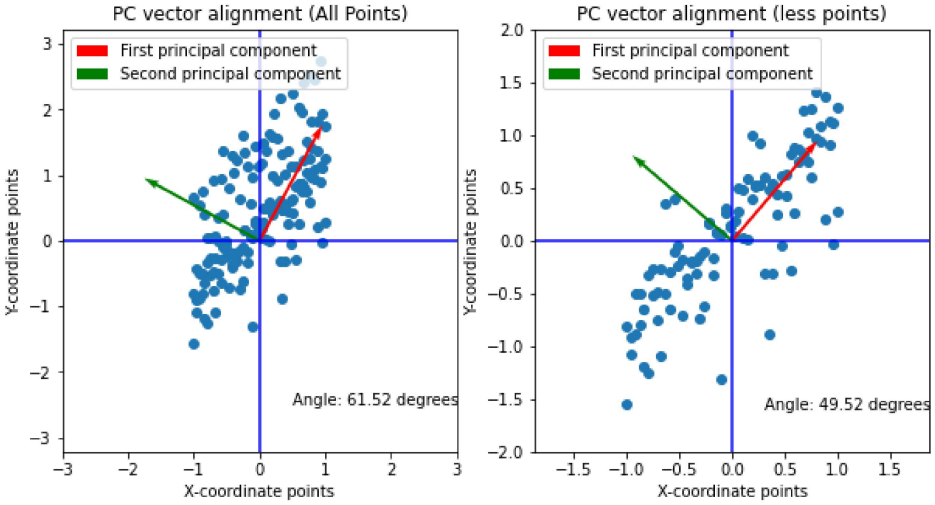


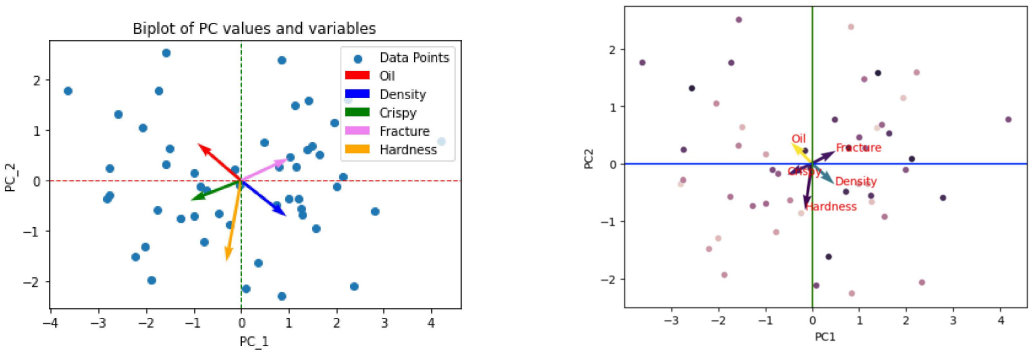
Figure 24: Principal Component Vector alignment with the data points

Another test was performed to verify if the principal component analysis (PCA) accurately captures the direction of maximum variance (information) in a given dataset. The principal component vector should point in the direction where the data spread is greatest. Initially, data points were generated with different offsets, and the eigenvectors of the covariance matrix were calculated. The angle made by the principal component vector with the x-axis was determined. Later, the data points with the largest offsets were removed, and the eigenvectors and their angles were recalculated. This process is shown in Fig. [24].

The first subplot shows the alignment of the principal component vector when all data points are included, while the second subplot shows the alignment after removing the data points with higher offsets. When these higher offset points are

removed, the principal component vector changes its direction to align with the direction of maximum variance in the remaining data. This change in the angle of the principal component vector is clearly seen in the figure [24].

In addition, a sample dataset has been taken from the SOGA-Py project[9]. The results obtained from the PCA of this project are compared to the results obtained from the source project, as shown in figure [25]. The direction of the variables matches, illustrating that the principal component analysis is implemented correctly.



Biplot of PCA results

Biplot from SOGA-Py project[9]

Figure 25: Comparison of Biplots

6.3 Test KNN

The functional test of KNN is to verify the implementation of the K-Nearest neighbours (KNN) classification algorithm. The aim of this test is to verify if the model correctly predicts the test dataset labels. A sample dataset with independent and dependent features is taken, with a parameter value k = 3 (number of nearest neighbours). The dataset is split into train and test datasets. The distance between each test data point and all training data points is calculated manually, and they are sorted in ascending order. Based on the selected k value, the labels of the k closest training data points are considered. The class with the majority voting among the k neighbours is assigned to the test data point. The predicted

labels from the knn main function are compared to the expected labels computed manually for the verification. The expected values match the obtained predictions called from the main knn function.

6.3.1 Decision boundary plotting

The functionality test of KNN can be verified by plotting a decision boundary [26] for a sample dataset taken from kaggle[23].

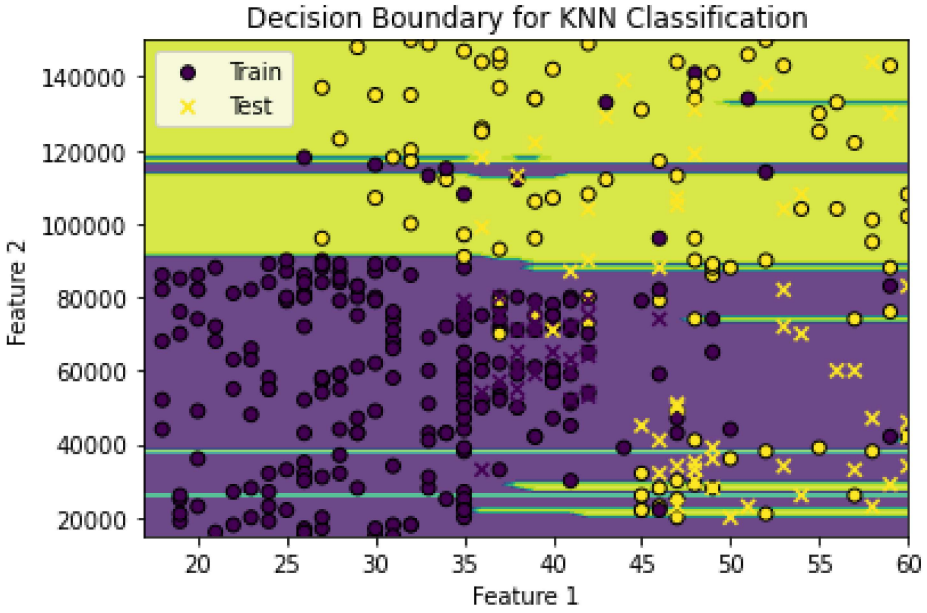


Figure 26: Plotting Decision boundary for KNN

6.4 Test ANN

The forward propagation of the neural network is verified by considering a sample numpy array as input, and the known pre-calculated output. The output from the model matches the expected output. The same method is followed to verify the activation functions.

Gradient check: The backward propagation is verified using the gradient check method[24] to check the gradients computed by the model. While implementing backward propagation, there might be many chances to commit mistakes in the derivation part. In such cases, the gradient descent method can be used to verify the correctness of this step. A small mistake in computing gradients may impact the model's performance.

The gradient of the cost function J(θ) w.r.t parameter θ is given as:

$$\frac{dJ(\theta)}{d\theta} = \lim_{\epsilon \to 0} \frac{J(\theta + \epsilon) - J(\theta - \epsilon)}{2\epsilon} \tag{6.1}$$

The main objective of this test is to make sure that the analytically computed gradients (from backpropagation) match the numerically approximated gradients. Choose a small epsilon value (10^{-4}). In order to validate these gradients, each parameter (weights and biases) is significantly perturbed by a small value ϵ using the function (check bwd prop), which then computes the forward propagation cost for these perturbed parameters. By dividing the cost difference by 2ϵ, the numerical gradient can be determined. The norm of the difference between this gradient and the analytical gradient is used to calculate the error measure. A small error indicates that the backpropagation implementation is correct. If the error value is near zero, the function confirms that the gradient computed is correct.

6.4.1 Decision boundary plotting

The functionality test of DNN can be verified by plotting a decision boundary[27] for a sample dataset taken from kaggle[23].

6.5 Test Random Forest

The functionality test random forest can be verified by plotting a decision boundary[28] for a sample dataset taken from kaggle[23].

From the plotting of decision boundaries, we can infer that the classification algorithms are functioning as expected.

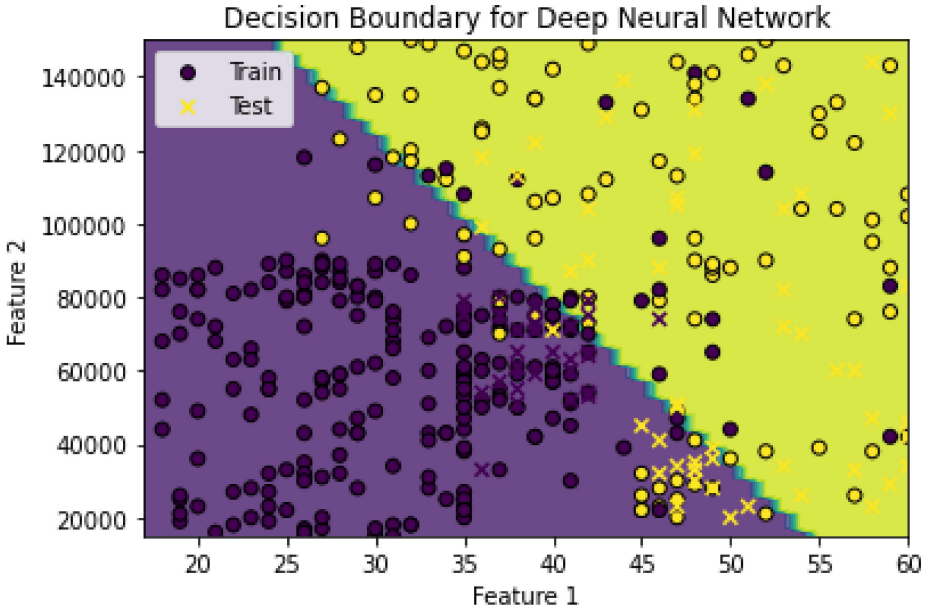


Figure 27: Plotting Decision boundary for Deep Neural Network

6.6 Functional test for Regression models

The main goal of this test is to make sure that the values predicted by the regression models match the expected pre-calculated values based on the given inputs and hyperparameters. The functional test for the regression models (ANN, Gradient Boosting, and Multiple Linear Regression) can be implemented by creating a sample dataset with known independent features (x train) and dependent features (y train). The independent features have two features, one with a different integer values (n) and the other with a value 2 in all samples. The target feature would be the multiplication of n different features by 2 (n * 2). The regression models are trained on this train dataset using the discussed independent features (x train) and the corresponding target feature (y train). By using the trained model, the values for the test data have to be predicted. The difference between the expected prediction values and the obtained prediction values from the model should be within a small tolerance value. Then, the model is said to be functioning as expected.

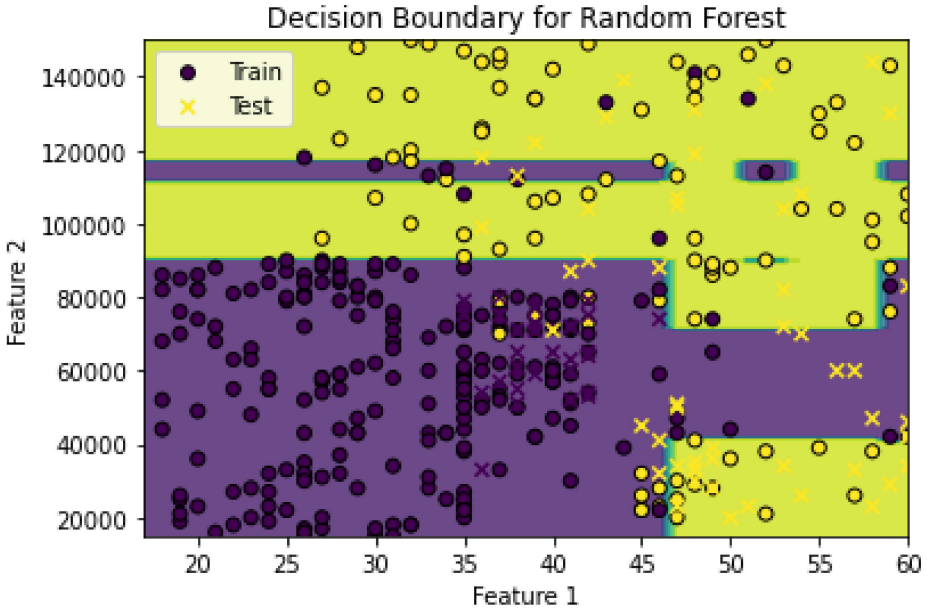


Figure 28: Plotting Decision boundary for Random Forest Algorithm

7 Results Analysis

7.1 KNN Classification

The performance of the K-Nearest neighbours classification model is determined by choosing the optimal value of k nearest neighbours. Hyperparameter tuning of the k values against F1 scores has been performed to get the optimal value, as shown in Fig. [15]. The optimal value of k is the point at which the F1 score reaches its maximum. This k value plays an important role in defining the KNN model's accuracy. Once the k value is selected, predictions on the test dataset can be performed using these k nearest neighbours. The model can be evaluated by comparing the predicted labels to the true labels.

An accuracy of 89.29% is achieved by this model, which means that 89.29% of the overall samples are correctly classified. With a precision score of 0.922, the KNN model correctly predicted a positive outcome in almost 92.2% of all the positively predicted cases. With a recall score of 0.926, the KNN model was

able to accurately identify 92.6% of the dataset's true positive cases. The KNN model appears to have found a good compromise between precision and recall, as indicated by its F1-score of 0.924.

7.2 Random Forest Classification

Hyperparameters such as the number of trees (n estimators), the minimum number of samples to split, and the maximum depth of each tree play a crucial role in improving the performance of the Random Forest classification model.The number of trees parameter gives information about the number of trees in a forest, seven trees are considered in this project after conducting hyperparameter tuning. The performance of the model can be enhanced if there are more trees, but if the data is imbalanced, then there might be an issue of wrongly predicted. Minimum number of samples describe the minimum number of instances needed to split the node. After performing hyperparameter tuning, no minimum samples are needed to split into child nodes. This makes sure that splits are made only when there is sufficient data, which helps prevent overfitting. The maximum depth of a tree determines how many levels it can have, i.e., the tree can split up to this level from root note to leaf node. In this project, based on the hyperparameter tuning, each tree can have 20 levels of depth in a forest. The optimal values of these hyperparameters can be observed in the Fig. [29]

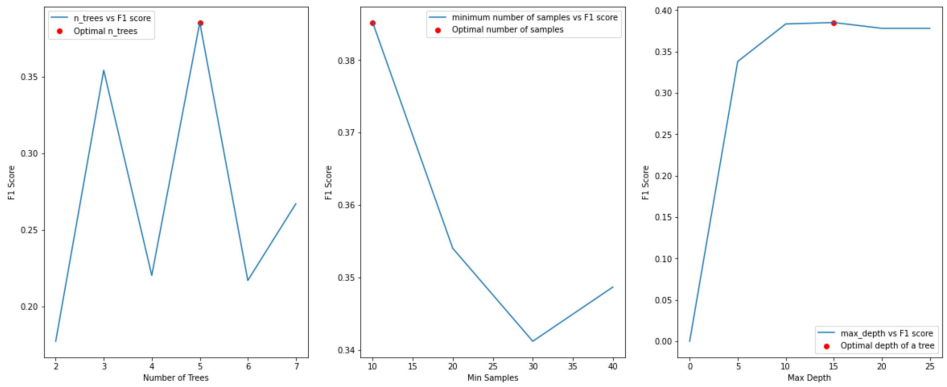


Figure 29: Hyperparameters tuning of Random forest model

This model has an accuracy of 62.74%, which means that 62.74% of the samples are correctly predicted out of all the examples. The precision of about 0.706 implies that 70.57% of samples are true positive predictions out of all positive predictions. The ratio of true positives to the total actual positives, i.e., the recall value, is 0.808. This means the model correctly predicts 80.84% of true positives out of all the actual positives. The model achieves a good balance between precision and recall, as indicated by its F1-Score of 0.754.

7.3 Artificial Neural Network

In this project, a DNN with 3 layers (2 hidden and 1 output) is used for the classification task, and a shallow neural network (1 hidden layer) is used for the regression task. Based on the hyperparameter tuning of the number of layers for the regression task, the model is performing better with 1 hidden layer than with 2 hidden layers. In the case of the classification task, the model is performing better with 3 layers (30-50-1), i.e., 30 units in the first hidden layer, 50 units in the second hidden layer, and 1 unit in the final layer, which classifies the material into metal and non-metal.

In order to improve the performance of the model, four different optimization algorithms are used in this project. The one that performing better is used to train the model and perform prediction on the test dataset. The better performing optimization algorithm can be determined by hyperparameter tuning. The optimization algorithm which has more accuracy and minimum cost is preferred. These optimization algorithms are Stochastic gradient descent, Stochastic gradient descent with momentum, Root Mean Square Propagation (RMSProp) and Adaptive optimization (Adam). These algorithms are clearly explained in the section 5.3.5. The model is trained with the mentioned optimization algorithms with the parameters considered as per the table 1. Among the different optimization algorithms, Adam optimizer is the optimal optimizer with minimum cost value and highest accuracy which can be observed clearly in the Fig. [30]

In DNN (Deep Neural Network), hidden layers play an important role in learning the complex patterns in the data. The number of neurons in each hidden

Neurons in 1st layer	Neurons in 2nd layer	Mini-Batch Size	Cost
30	50	256	0.167
50	30	512	0.185
30	30	256	0.192
50	50	512	0.192
50	50	2050	0.196
30	15	256	0.198
30	50	1020	0.201

Table 5: Neurons configuration and corresponding cost

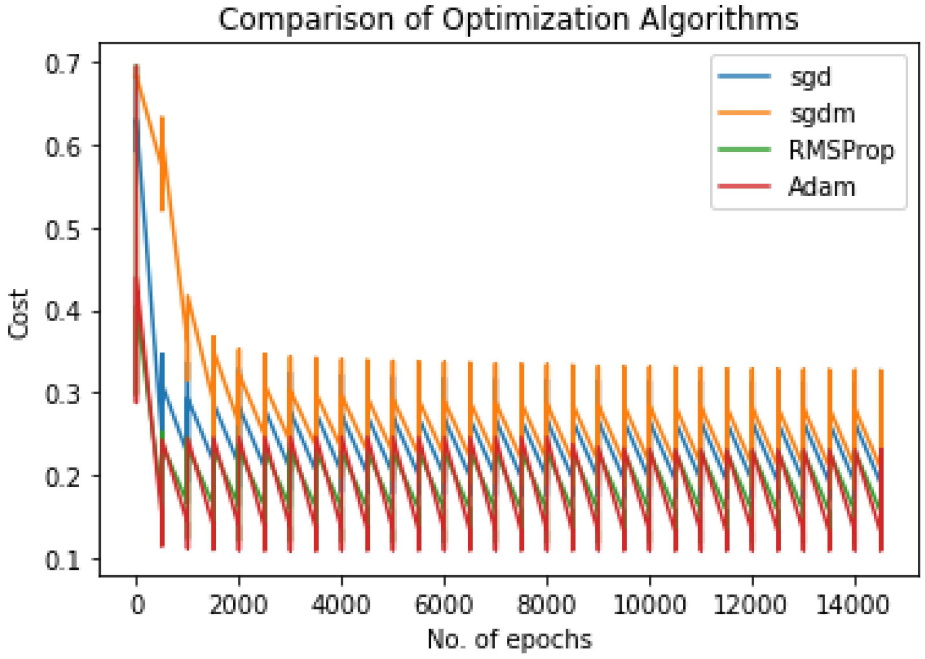


Figure 30: Comparison of optimization algorithms

layer is also tuned to improve the performance of the model. The neuron number combination that contributes more accuracy with minimum loss is considered the optimal set of neuron numbers in the hidden layers of DNN (Deep Neural Network). Different ranges of the number of neurons in each hidden layer are

considered for tuning. Among those ranges, the first hidden layer with 30 neurons and the second hidden layer with 50 neurons are the most effective combinations for minimizing the cost function. This combination is considered to be the optimal set of neurons for the first and second hidden layers. One layer of 4 neurons are used for the regression task which performs better than the two layered network.

The learning rate is an essential hyperparameter that must be tuned in order to train an ANN model. It has an impact on the convergence speed and performance of the model. An optimal learning rate value speeds up convergence with the minimum loss and efficiently trains up the model. A low learning rate makes the model learn slowly (it needs more epochs to converge) and makes it hard to reach the global minimum. A large value of the learning rate might diverge the model, and the model performance will be poor. The random values of learning rate are initialized within a range of 0.01 to 1 for stochastic gradient descent and stochastic gradient descent with momentum, whereas the RMSProp and Adam learning rate values are initialized within a range of 0.0001 to 1. Since RMSProp and Adam are advanced optimization algorithms that adapt the learning rate during training, small learning rate values are initialized to avoid divergence. The optimal learning rate of stochastic gradient descent and stochastic gradient descent with momentum is 0.2008810707215905, and RMSProp and Adam have an optimal learning rate value of 0.000354728214309162 with the minimum cost value and highest F1 score.

This model has a prediction accuracy of 90.01%, which means that 90.01% of the samples are correctly predicted out of all the examples. The precision of about 0.930 implies that 92.97% of samples are true positive predictions out of all positive predictions. The ratio of true positives to the total actual positives, i.e., the recall value, is 0.928. This means the model correctly predicts 92.85% of true positives out of all the actual positives. The model performs quite well when finding a good balance between precision and recall, as indicated by its F1-Score of 0.929. Therefore, the model performs well for the test instances by attaining high F1-Score and maintaining a good ratio of precision to recall.

For the regression model, the root mean squared error (RMSE) value is 0.532. The lower the RMSE value, the better the performance of the model. The prediction mean absolute error (MAE) value is 0.304, the lower the value, the better the

performance of the model. The R-squared (R^2) value is 0.784, the closer the value to 1, the better the model fits the data.

7.4 Multiple Linear Regression

Multiple linear regression predicts the value by fitting a linear relationship between the independent features and dependent features. Gradient descent optimization method is used to minimize the mean squared error cost function. The parameters are updated iteratively in order to minimize the cost function. The performance of the model can be evaluated by using root mean squared error (RMSE), mean absolute error (MAE), and R-squared error (RSE) metrics for both training and test datasets.

The RMSE value for the training dataset is 0.363, whereas for the test dataset, it is 0.441. The lower the RMSE value, the better the performance of the model. The absolute difference between the predicted and true value is given by MAE. The MAE value for the training dataset is 0.228 and the test dataset is 0.257. The model's performance is better when the MAE value is lower. The relative squared error (RSE) for both training and test datasets is relatively high, which indicates that the model is performing better, and the values for train and test datasets are 0.856 and 0.852, respectively.

7.5 Gradient Boosting for Regression

The minimum number of samples to split the node, the maximum depth of each tree, and the number of trees (n estimators) are the hyperparameters used in this Gradient Boosting for Regression model. In order to improve the learning process of the model, a new tree aims to learn from the errors of the previous ones. By tuning these hyperparameters, the performance of the model can be improved. The performance of the model gets better if we use a large number of trees, where the model can correct the errors from the previous ones. By performing the hyperparameter tuning, the optimal number of trees is 20, the maximum depth of each tree is 20, and the minimum number of samples to required split is 4 which

can be observed in the Fig. [31].

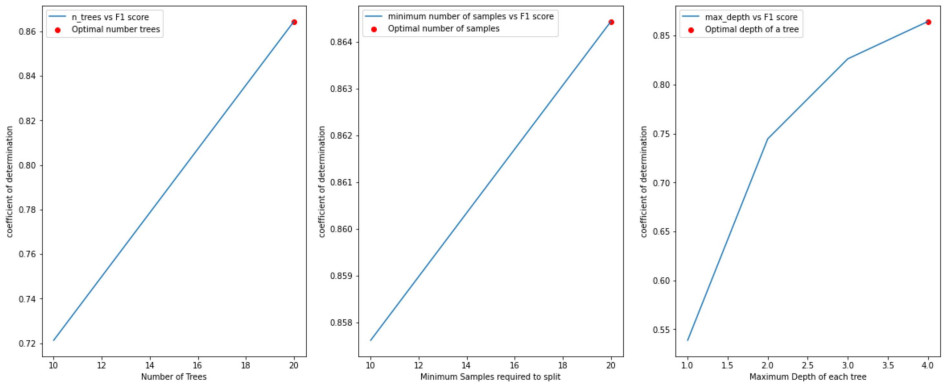


Figure 31: Tuning the hyperparameters of Gradient Boosting

The performance of the Gradient Boosting Model can be determined by using the evaluation metrics. The RMSE value for the test dataset is 0.378. The lower the RMSE value, the better the performance of the model. The absolute difference between the predicted and true values is given by MAE. The MAE value for the test dataset is 0.255. The model's performance is better when the MAE value is lower. The relative squared error (RSE) for the test dataset indicates that the model is performing well, with a value of 0.89.

7.6 Overall Results

Model	**F1 Score**	**Recall**	**Precision**	**Accuracy**
ANN Classification	0.929	0.928	0.930	90.01%
Random Forest	0.754	0.808	0.706	62.74%
KNN Classification	0.924	0.926	0.922	89.29%

Table 6: Comparison of classification models evaluation metrics

From the table [6], the artificial neural network classification model has a high F1 score, recall, precision, and accuracy values. As this is the best-performing model, the compound's that are metals are filtered using this algorithm, and the

corresponding principal components are used as inputs to the regression models to predict the Debye temperature of metals.

Model	**RMSE**	**MAE**	**R Squared**
Gradient Boosting	0.378	0.255	0.89
ANN Regression	0.532	0.304	0.784
MLR	0.440	0.257	0.852

Table 7: Comparison of regression models evaluation metrics

From the table [7], it can be observed that the Gradient Boosting is the best performing model to predict the Debye temperature of metals, followed by Multiple Linear Regression and ANN Regression.

8 Conclusion

The goal of this project was to use machine learning models to estimate the Debye temperature of metals by using a dataset that included elemental and material parameters. The data was prepared for modeling by means of thorough preprocessing procedures such as feature selection, imputation of missing values, and encoding. Metals and non-metals were successfully distinguished in the first classification tasks, and metal-specific Debye temperature predictions were made by regression models. Based on the results, evaluation metrics such as RMSE, MAE, and RSE indicate good accuracy performed by the machine learning models, but there are still certain places where improvement is needed to enhance the performance further. By doing some research, additional features can be incorporated and other machine learning models can be explored in order to improve the prediction accuracy. The model can be refined by conducting cross-validation and sensitivity analysis. Other hyperparameters can be tuned to improve the predictions. This project can be extended to predict material properties other than the Debye temperature parameter.

9 Manual

This section will provide a thorough explanation of how to run the program with various input choices. The executable files and input files for all functions in this model are already in their proper locations. No modifications are required for any program to function properly. The program has to be executed sequentially so that the files are updated accordingly. The main.py file needs to be executed to start the program. The raw data files that are used in this project are already taken from the online repository and available offline for use. Chemical compound formulas, data, and elemental properties are the raw files used. The instructions to be followed before execution of main.py file:

1. Ensure that all the required libraries are installed before executing the main program, which should be in the same version mentioned in Table [4].
2. Files name should not be modified. This would affect the execution of the program.
3. As instructed by their paths in the script, place all necessary files in the appropriate directories.
4. The hyperparameters can be adjusted within the specified range before executing the each classification and regression model.

9.1 Executing main program

1. The python command python main.py is used to execute the main.py file.

To execute the `main.py` file, use the following command:

```
python main.py
```

Once the main.py file is executed, A few prompts indicating which stage of the operation is being carried out will appear as shown in the Fig. [32].

The files in table [8] are used during the execution of data preprocessing process:

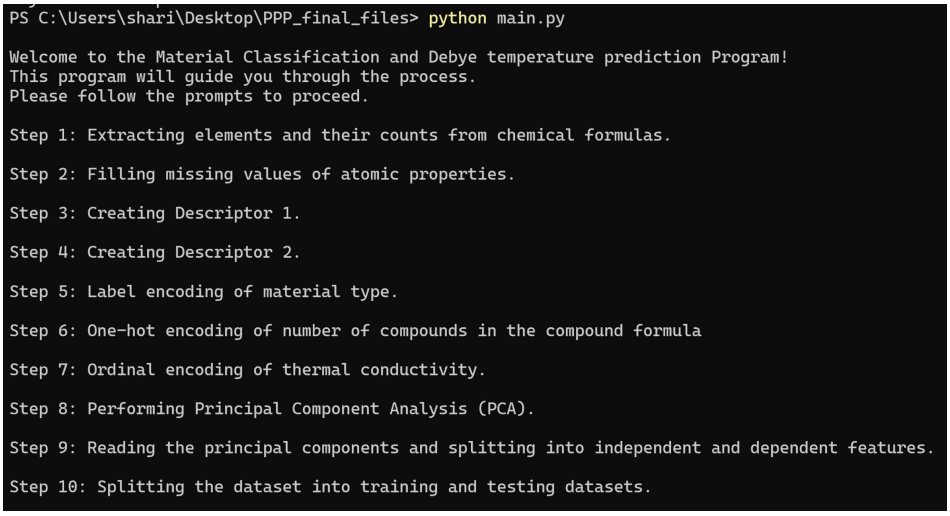


Figure 32: Prompts during the Data preprocessing step

Process	Files used during execution	Executable files
Data preprocessing	raw_compound's_data.csv compound's_to_ele_count.csv Elements_data.csv Elements_data_updated.csv DPP1_weighted_atomic_prop.csv DPP2_ind_ele_col.csv DPP3_label_encode.csv DPP4_ele_composition_num_ele.csv	compound's_to_element_count.py elements_data_updated.py Desc2_weighted_atomic_properties.py Desc1_comp_to_ind_ele.py label_encoding.py one_hot_encode_ele_composition.py ordinal_encod.py
Principal Component Analysis (PCA)	DPP5_ordinal_encode_file.csv principal_components_with_dependent_features.csv	pca.py

Table 8: Files required during execution of data preprocessing and PCA

2. Once the data preprocessing and principal component analysis steps are finished, the principal components are used for classification task. The principal components are split into training and test dataset with a split ratio of 0.8 which contains independent and dependent features(Step 9 and 10 in Fig. [32]).

3. After step 10, the summary of the dataset will be prompted, as shown in the Fig. [33].

4. Following data splitting, a prompt will appear asking which classification model to use. User has three options for selecting the classification algorithms.

1. Input 1 for Deep Neural Network (suitable for handling complex relationships

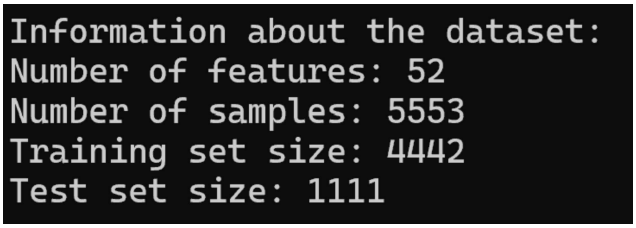


Figure 33: Prompts after dataset split

in the dataset)

2. Input 2 for K-Nearest Neighbours (simple model and classifies the data point based on k neighbours)

3. Input 3 for Random Forest Classification (ensemble method containing multiple decision trees)

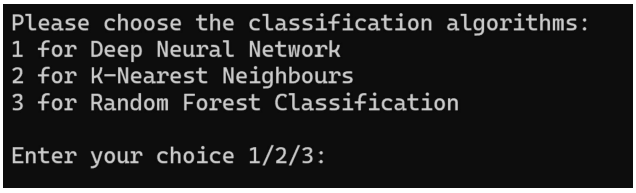


Figure 34: Prompts to choose classification model

5. After completing the training and testing, the evaluation metrics of the selected classification model will be displayed on the screen.

6. After classifying the materials into metals and non-metals, If you want to use regression models to predict the Debye temperature, you will be prompted to enter 1, otherwise, you will be asked to input 0.

7. The files required for the classification and regression models are given in the tables [9 & 10].

8. If you choose 0, the program will terminate. If you choose 1, the program will continue with Debye temperature prediction using regression models. Once the dataset containing only metals data is split into training and test datasets, you

will be prompted to choose a regression algorithm to predict Debye temperature as shown the Fig. [35]

9. Based on the selection of the regression model, the corresponding evaluation metrics will be displayed on the screen and the predicted of debye temperature values will be obtained.

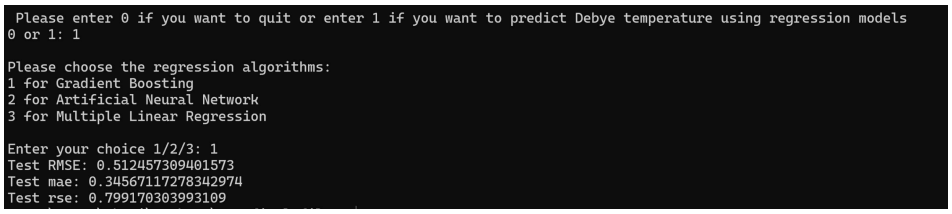


Figure 35: Prompts to choose Regression model

Classification Models	Files used during execution	Executable files
Deep Neural Network	principal_components_with_dependent_features.csv input_file_regression.csv	ANN_classification.py
K-Nearest Neighbours	principal_components_with_dependent_features.csv	Knn_classification.py
Random Forest Classification	principal_components_with_dependent_features.csv	random_forest_classification.py

Table 9: Files required during execution of Classification models

Regression Models	Files used during execution	Executable files
Gradient Boosting Regression	input_file_regression.csv	gradient_boosting_regression.py
Artificial Neural Network	input_file_regression.csv	ANN_regression.py
Multiple Linear Regression	input_file_regression.csv	multiple_linear_regression.py

Table 10: Files required during execution of Regression models

9.2 Executing Test cases

All the test files are stored in Testing folder which is in the main directory.

The python command `python -m pytest` is used to execute all the test cases from the main folder path. To view the test cases in more detail, use the command `python -m pytest -v` which gives information about the result of individual test case. The execution of this command is shown if Fig. [36]. Individual file can be

tested using the command `pytest .\test_Filename`. If the file name is pca.py then the command to test is `pytest .\test_pca.py`

```
python -m pytest
```

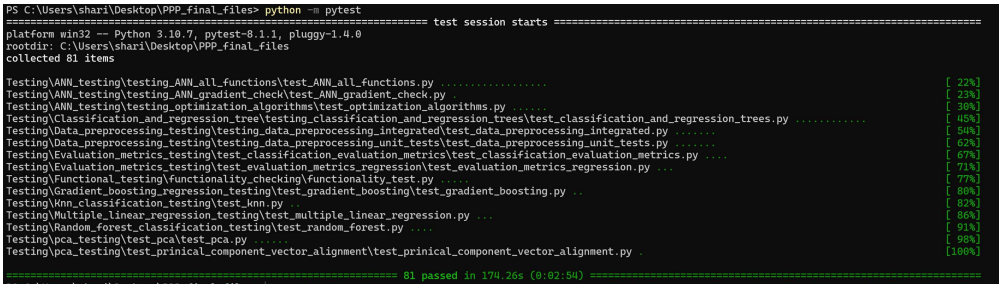


Figure 36: Pytest to test the functions

10 GIT Log

```
commit 6b42e52cf4cad6482b54100698fef10a311bcf3c (HEAD -> main, origin/main)
Author: Harish Somaghatta <s.harish180@gmail.com>
Date:   Wed May 15 02:22:32 2024 +0200

    Modified Machine learning models -KNN, Random Forest, Gradient boosting

commit a916d4bcb599fa41bb1379cbcf0e31e228c6d8b7
Author: Harish Somaghatta <s.harish180@gmail.com>
Date:   Wed Mar 27 17:55:57 2024 +0100

    ANN with gradient check test

commit 5f4a8e89777a3f713c238b1974aa0483c95ee785
Author: Harish Somaghatta <s.harish180@gmail.com>
Date:   Sun Mar 24 02:31:49 2024 +0100

    Test PCA in 3d

commit 29b5a262f6593dd8ddb4928a62276a923730560a
Author: Harish Somaghatta <s.harish180@gmail.com>
Date:   Sun Mar 17 20:14:02 2024 +0100

    Modified and new files(ANN_regression)

commit 9a9ef31b9c9726139a266e52f979bddc7b17d8fb
Author: Harish Somaghatta <s.harish180@gmail.com>
Date:   Tue Feb 27 17:16:44 2024 +0100
```

```
    Modified ANN classification file
:...skipping...
commit 6b42e52cf4cad6482b54100698fef10a311bcf3c (HEAD -> main, origin/main)
Author: Harish Somaghatta <s.harish180@gmail.com>
Date:   Wed May 15 02:22:32 2024 +0200

    Modified Machine learning models -KNN, Random Forest, Gradient boosting

commit a916d4bcb599fa41bb1379cbcf0e31e228c6d8b7
Author: Harish Somaghatta <s.harish180@gmail.com>
Date:   Wed Mar 27 17:55:57 2024 +0100

    ANN with gradient check test

commit 5f4a8e89777a3f713c238b1974aa0483c95ee785
Author: Harish Somaghatta <s.harish180@gmail.com>
Date:   Sun Mar 24 02:31:49 2024 +0100

    Test PCA in 3d

commit 29b5a262f6593dd8ddb4928a62276a923730560a
Author: Harish Somaghatta <s.harish180@gmail.com>
Date:   Sun Mar 17 20:14:02 2024 +0100

    Modified and new files(ANN_regression)

commit 9a9ef31b9c9726139a266e52f979bddc7b17d8fb
Author: Harish Somaghatta <s.harish180@gmail.com>
Date:   Tue Feb 27 17:16:44 2024 +0100

    Modified ANN classification file
```

```
commit 22aa29e0fa9076c406702383d47c9c0fa7b4ba5f
Author: Harish Somaghatta <s.harish180@gmail.com>
Date:   Tue Feb 20 18:55:32 2024 +0100

    pca validation file and modified old files

commit f854db7329f91f79a039b757f8cc3eed426c8317
Author: Harish Somaghatta <s.harish180@gmail.com>
Date:   Wed Jan 31 23:24:26 2024 +0100

    Updated ANN file

commit f996d654fcc4cb8b401500ce91bd77394f7e5c94
Author: Harish Somaghatta <s.harish180@gmail.com>
Date:   Wed Jan 31 12:59:33 2024 +0100

    ANN Backward Propagation

commit aee83bbce907616f81cf2d6b6d919d9e792da2a4
Author: Harish Somaghatta <s.harish180@gmail.com>
Date:   Tue Jan 30 00:53:12 2024 +0100

    ANN Forward Propagation code with test cases

commit e997c5797be0261fe37a2ce92c1ea907d8473133
Author: Harish Somaghatta <s.harish180@gmail.com>
Date:   Mon Jan 29 23:53:00 2024 +0100

    modified PCA file and ANN_classification

commit 7b2c0886ddc189afbc8d51f4ea54e165aa8a9084
Author: Harish Somaghatta <s.harish180@gmail.com>
```

```
Date:   Tue Jan 23 21:18:13 2024 +0100

    modified pca file

commit e68464cb1f77cd98e5c6993e7bbd91e6f2613d00
Author: Harish Somaghatta <s.harish180@gmail.com>
Date:   Tue Jan 23 20:10:39 2024 +0100

    Principal Component Analysis code file

commit c0acd084e34868c6a0b0059a8ccff0ee932aef70
Author: Harish Somaghatta <s.harish180@gmail.com>
Date:   Mon Jan 22 23:24:36 2024 +0100

    Added test case to Data Preprocessing files

commit cb38861a4b645692124e9064d69c4dd43a923e4d
Author: Harish Somaghatta <s.harish180@gmail.com>
Date:   Mon Jan 22 19:25:24 2024 +0100

    Modified compound weighted atomic properties file

commit ad67e882943619d76f4b0595386b71891ca44f58
Author: Harish Somaghatta <s.harish180@gmail.com>
Date:   Fri Jan 19 04:09:28 2024 +0100

    Modified Data Preprocessing Files

commit 2a440ca81d6e9e7d77ddca57049f64175375fc2b
Author: Harish Somaghatta <s.harish180@gmail.com>
Date:   Wed Jan 17 12:36:39 2024 +0100
```

```
    Data Pre-processing files

commit 402986c9e5480edd2f75e600b15f9f6952c4f8f6
Author: Harish Somaghatta <s.harish180@gmail.com>
Date:   Wed Jan 10 19:25:14 2024 +0100

    compound's to elements and their count

commit e3292fe649ebe50d3f4c6dffad620e27077030c5
Merge: 645e764 6f9d6e7
Author: Harish Somaghatta <s.harish180@gmail.com>
Date:   Tue Jan 9 19:28:08 2024 +0100

    Merge branch 'main' of https://gitlab.com/arun.prakash.mimm/2023_ppp_harishsom

commit 645e764ac052b861a1f85581943208abde3de27a
Author: Harish Somaghatta <s.harish180@gmail.com>
Date:   Tue Jan 9 19:23:08 2024 +0100

    compound's to elements and their count
```

References

[1] S. H. Simon, *The Oxford Solid State Basics*. Oxford, United Kingdom: CPI Group (UK) Ltd, Croydon, CR0 4YY, 2013.

[2] C. Kittel, *Introduction to Solid State Physics, 8th ed.* 111 River Street, Hoboken: John Wiley Sons, 2004.

[3] P. A. Filanovich, A.N., "Model description of phonon spectrum of solids: A machine learning approach," *Metallurgical and Materials Transactions*, vol. 52, 09 2021.

[4] S. Curtarolo, W. Setyawan, S. Wang, J. Xue, K. Yang, R. H. Taylor, L. J. Nelson, G. L. Hart, S. Sanvito, M. Buongiorno-Nardelli, N. Mingo, and O. Levy, "Aflowlib.org: A distributed materials properties repository from high-throughput ab initio calculations," *Computational Materials Science*, vol. 58, pp. 227–235, 2012.

[5] A. N. Filanovich and A. A. Povzner, "Machine learning methods for predicting the lattice characteristics of materials," in *2020 Ural Symposium on Biomedical Engineering, Radioelectronics and Information Technology (USBEREIT)*, pp. 0414–0416, 2020.

[6] C. Toher, C. Oses, D. Hicks, E. Gossett, F. Rose, O. Levy, and S. Curtarolo, *The AFLOW Fleet for Materials Discovery*, pp. 1785–1812. Cham: Springer International Publishing, 2020.

[7] D. T. Larose and C. D. Larose, *Discovering knowledge in data, 2nd ed. : an introduction to data mining, 2nd ed.* Hoboken, New Jersey: John Wiley Sons, 2014.

[8] K. Hamidieh, "A data-driven statistical model for predicting the critical temperature of a superconductor," 2018.

[9] K. J. H. K. Rudolph, A., "Statistics and geodata analysis using python (sogapy)," 2023.

[10] L. Rutkowski, M. Jaworski, L. Pietruczuk, and P. Duda, "The cart decision tree for mining data streams," *Information Sciences*, vol. 266, pp. 1–15, 2014.

[11] L. X. Iranzad, Reza, "A review of random forest-based feature selection methods for data science education and applications," *International Journal of Data Science and Analytics*, 2024.

[12] C. A. Bentéjac, Candice, "A comparative analysis of gradient boosting algorithms," *Artificial Intelligence Review*, 2021.

[13] H. Zhang and Y. Zhou, "A neural network-based weighted voting algorithm for multi-target classification in wsn," *Sensors*, vol. 24, no. 1, 2024.

[14] K. Suzuki, *Artificial Neural Networks - Methodological Advances and Biomedical Applications*. 04 2011.

[15] O. A. Montesinos López, A. Montesinos López, and J. Crossa, *Fundamentals of Artificial Neural Networks and Deep Learning*, pp. 379–425. Cham: Springer International Publishing, 2022.

[16] M. Z. Mulla, "Cost, activation, loss function—,neural network, deep learning," 2020. Medium.

[17] A. Ng, "Neural networks and deep learning," Coursera.

[18] D. Soydaner, "A comparison of optimization algorithms for deep learning," *International Journal of Pattern Recognition and Artificial Intelli gence*, 2019.

[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[20] G. S. Prayoga, "Multiple linear regression and gradient descent," 2022. Medium.

[21] S. K. Agrawal, "Metrics to evaluate your classification model to take the right decisions," 2024. analyticsvidhya.

[22] R. Agrawal, "Know the best evaluation metrics for your regression model," 2023. analyticsvidhya.