Helwan University,

Facility of Science,

Department of Physics and Computer Science

Final Year Project

Prediction of Superconductor critical temperature using Machine learning techniques

By:

Mohamed Ahmed Ahmed Mohamed Ali

Most

Supervised By:

Ph.D. Mourad Raafat

Supervisor(s)

Date of examination

12 / 7 / 2021

ACKNOWLEDGMENT

First and foremost, praises and thanks to the God, the Almighty, for His showers of blessings throughout my research work to complete the research successfully.

I would like to express my deep and sincere gratitude to my research supervisor and my mentor, Dr. Mourad Raafat., Doctor of Philosophy - Ph.D., Computer Science, Professor, Helwan University, for giving me the opportunity to do research and providing invaluable guidance throughout this research. His dynamism, vision, sincerity and motivation have deeply inspired me. He has taught me the methodology to carry out the research and to present the research works as clearly as possible. It was a great privilege and honor to work and study under his guidance. I am extremely grateful for what he has offered me. I would also like to thank him for his friendship, empathy, and great sense of humor.

I also would like to thank Dr. Ahmed Ezzat, Dr. Ahmed Madbouly, and Dr. Rashad Ragab. for their support and their helpful knowledge, several other professors and Teaching Assistants gave helpful advice and knowledge throw the years I spend in faculty of science.

Special thanks for the great research and effort of Kam Hamidieh, Statistics Department, The Wharton School, University of Pennsylvania

Finally, my thanks go to all the people who have supported me to complete the research work directly or indirectly.

DECLARATION

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Bachelor of Science in *(physics and computer science, mathematics and computer science)* is entirely my own work, that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: _____

Registration No.: _____

Date: 12 / 7 / 2021.

ABSTRACT

Superconductivity is the focus of research since it was discovered by Dutch physicist Heike Kamerlingh Onnes of Leiden University In 1911 in 1913, he won a Nobel Prize due to its unique properties and promising technology even though some features of this unique phenomenon remain poorly understood; prime among these is the connection between superconductivity and chemical/structural properties of materials. In order to fill that gap, we want to build a machine learning model that predicts the critical temperatures *(Tc)* of superconducting material from the 21,000+ known superconductors available via the SuperCon database. The model uses features based only on the chemical compositions that were collected SuperCon database Then filtered the irrelevant features. A Regression models are developed to predict the values of *Tc*. It shows strong predictive power, with out-of-sample accuracy of about 92%. these models also demonstrate good performance, with learned predictors offering potential insights into the mechanisms behind superconductivity in different families of materials.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter One

1 INTRODUCTION

Superconductivity, despite being the subject of intense physics, chemistry, and materials science research for more than a century, remains among one of the most puzzling scientific topics.[1] It is an intrinsically quantum phenomenon caused by a finite attraction between paired electrons, with unique properties including zero DC resistivity, Meissner, and Josephson effects, and with an ever-growing list of current and potential applications. There is even a profound connection between phenomena in the superconducting state and the Higgs mechanism in particle physics.[2] However, understanding the relationship between superconductivity and materials' chemistry and structure present significant theoretical and experimental challenges. In particular, despite focused research efforts in the last 30 years, the mechanisms responsible for high-temperature superconductivity in cuprate and iron-based
families remain elusive.[3,4]

They have Signiant practical applications. Perhaps the best-known application is in the Magnetic Resonance Imaging (MRI) systems widely employed by health care professionals for detailed internal body imaging. Other prominent applications include the superconducting coils used to maintain high magnetised in the Large Hadron Collider at CERN, where the existence of Higgs Boson was recently confirmed, and the extremely sensitive magnetic field measuring devices called SQUIDs (Super-conducting Quantum Interference Devices). Furthermore, superconductors could revolutionize the energy industry as frictionless (zero resistance) superconducting wires and electrical system may transport and deliver electricity with no energy loss; see Hassenzahl (2000)[5].

Quantum supremacy was recently achieved by Google using a superconducting microprocessor Sycamore[6]. An avalanche of similar results is now expected. The future of superconductors has never looked brighter. However, if Sycamore and other superconducting microprocessors are to and a wider circle of users, their operating temperature will have to be increased significantly. Sycamore is made of aluminium (Tc = 1.175 K) and indium (Tc = 3.41 K) and operates at temperatures below 20 mK[6]. Such low temperatures require a dilution refrigerator with 3He, which is exceedingly rare and expensive.

However, the widespread applications of superconductors have been held back by two majors' issues: (1) A superconductor conducts current with zero resistance only at or below its superconducting critical temperature (Tc). Often impractically, a superconductor must be cooled to extremely low temperatures near or below the boiling temperature of nitrogen (77 K) before exhabiting the zero-resistance property. (2) The scientific model and theory that predicts Tc is an open problem which has been baring the scientific community since the discovery of superconductivity in 1911 by Heike Kamerlingh Onnes, in Leiden.

In the absence of any theory-based prediction models, simple empirical rules based on experimental results have guided researchers in synthesizing superconducting materials for many years.
For example, the eminent experimental physicist Matthias (1955)[7] concluded that Tc is related to the number of available valence electrons per atom. (A few of these rules came to be known as the Matthias's rules.) It is now well known that many of the simple empirical rules are violated; see Conder (2016)[7].

This clearly illustrates the need for new superconducting materials with higher critical temperatures. However, finding new superconductors, especially with high Tc, is a very difficult endeavor[8]. In recent years, there has been a surge of interest in using artificial intelligence (AI), in particular machine learning (ML) and deep learning (DL), in materials physics[9-11]. The idea is that by using the existing information in materials' databases, several attempts have been made in predicting the critical temperatures of superconductors, or more generally, predicting new materials with potentially high Tc.

Application of statistical methods in the context of superconductivity began in the early eighties with simple clustering methods.[12,13] In particular, three “golden” descriptors confine the 60 known (at the time) superconductors with Tc > 10 K to three small islands in space: the averaged valence-electron numbers, orbital radii differences, and metallic electronegativity differences

Recent developments, however, allow a different approach to investigate what ultimately determines the superconducting critical temperatures (Tc) of materials. Extensive databases covering various measured and calculated materials properties have been created over the years.[14-18] The sheer quantity of accessible information also makes possible, and even necessary, the use of data-driven approaches, e.g., statistical and machine learning (ML) methods.

Such algorithms can be developed/trained on the variables collected in these databases, and employed to predict macroscopic properties, such as the melting temperatures of binary compounds,[19] the likely crystal structure at a given composition,[20] band gap energies[21,22], and density of states16 of certain classes of materials.

Several attempts have been made in predicting the critical temperatures of superconductors, or more generally, predicting new materials with potentially high Tc. Several prominent efforts have been by Stanev et.al[23], Hamidieh[24] and Zeng et al.[25]. Different AI approaches were used in these papers: Stanev et al. used both classification and regression models, Hamidieh used an XGBoosted statistical model, and Zeng et al. used convolutional neural networks (CNN).

In a recent work by Zhou et.al[26], the properties of the atoms were learned from the chemical compositions of compounds from a large database, without any additional information. Inspired by this approach, we made a similar attempt in predicting superconducting materials’ critical temperatures. superconducting, we used supervised machine learning and achieved statistical parameters comparable, and in some instances exceeding previous attempts. Below we describe in details the procedure used, and then the results of our study.

Valentin et al. (2017)[27] and our work are the only papers that focus on statistical models to predict Tc for a broad class of materials. However, Owolabi et al. (2014)[27] and Owolabi and Olatunji (2015)[27] focus on predicting Tc for Fe and MgB2 based superconductors respectively.

In this study, we take an entirely data-driven approach to create a statistical model that predicts Tc. The superconductor data comes from the Superconducting Material Database maintained by Japan's National Institute for Materials Science (NIMS).[28] After some data pre-processing, 21,263 superconductors are used.

Data extracted from the SuperCon database, which contains an exhaustive list of superconductors, including many closely related materials varying only by small changes in stoichiometry (doping plays a significant role in optimizing Tc). The order-of-magnitude increase in training data (i) presents crucial subtleties in chemical composition among related compounds, (ii) affords family-specific modelling exposing different superconducting mechanisms, and (iii) enhances model performance overall.
It also enables the optimization of several model construction procedures. Large sets of independent variables can be constructed and rigorously filtered by predictive power (rather than selecting them by intuition alone). These advances are crucial to uncovering insights into the emergence/suppression of superconductivity with composition.

In addition, most materials from the list share a peculiar feature in their electronic band structure: one (or more) flat/nearly flat bands just below the energy of the highest occupied electronic state. The associated large peak in the density of states (infinitely large in the limit of truly flat bands) can lead to strong electronic instability, and has been discussed recently as one possible way to high-temperature superconductivity.[29,30]

Taking advantage of this immense increase of readily accessible and potentially relevant information, we develop several ML methods modelling Tc from the complete list of reported superconductors.[28] In their simplest form, these methods take as input a number of predictors generated from the elemental composition of each material. Models developed with these basic features are surprisingly accurate, despite lacking information of relevant properties, such as space group, electronic, structure, and phonon energies.

We also discuss the factors that limit the learning process, most notably the wrong entries into the database.

Chapter Two

2 LITERATURE SURVEY.

2.1. History of Superconductors

Superconductors, materials that have no resistance to the flow of electricity, are one of the last great frontiers of scientific discovery[31]. Not only have the limits of superconductivity not yet been reached, but the theories that explain superconductor behavior seem to be constantly under review. In 1911superconductivity was first observed in mercury by Dutch physicist Heike Kamerlingh Onnes of Leiden University . When he cooled it to the temperature of liquid helium, 4 degrees Kelvin (-452F, -269C), its resistance suddenly disappeared. The Kelvin scale represents an "absolute" scale of temperature. Thus, it was necessary for Onnes to come within 4 degrees of the coldest temperature that is theoretically attainable to witness the phenomenon of superconductivity. Later, in 1913, he won a Nobel Prize in physics for his research in this area.

Figure 1: Walther Meissner and Robert Ochsenfeld

The next great milestone in understanding how matter behaves at extreme cold temperatures occurred in 1933. German researchers Walther Meissner and Robert Ochsenfeld discovered that a superconducting material will repel a magnetic field. A magnet moving by a conductor induces currents in the conductor. This is the principle on which the electric generator operates. But, in a superconductor the induced currents exactly mirror the field that would have otherwise penetrated the superconducting material - causing the magnet to be repulsed. This phenomenon is known as strong diamagnetism and is today often referred to as the "Meissner effect" (an

eponym). The Meissner effect is so strong that a magnet can actually be levitated over a superconductive material.

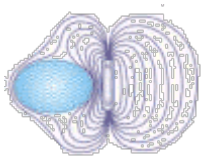


Figure 2: superconductor repelling magnetic field

In subsequent decades other superconducting metals, alloys and compounds were discovered. In 1941 niobium-nitride was found to superconduct at 16 K. In 1953 vanadium-silicon displayed superconductive properties at 17.5 K. And, in 1962 scientists at Westinghouse developed the first commercial superconducting wire, an alloy of niobium and titanium (NbTi). High-energy, particle-accelerator electromagnets made of copper-clad niobium-titanium were then developed in the 1960s at the Rutherford-Appleton Laboratory in the UK, and were first employed in a superconducting accelerator at the Fermilab Tevatron in the US in 1987.

Figure 3: John Bardeen, Leon Cooper, and John Schrieffer

The first widely-accepted theoretical understanding of superconductivity was advanced in 1957 by American physicists John Bardeen, Leon Cooper, and John Schrieffer. Their *Theories of Superconductivity* became known as the BCS theory - derived from the first letter of each man's last name - and won them a Nobel prize in 1972. The mathematically-complex BCS theory explained superconductivity at temperatures close to absolute zero for elements and simple alloys. However, at higher temperatures and with different superconductor systems, the BCS theory has subsequently become inadequate to fully explain how superconductivity is occurring.

Figure 4: Brian D. Josephson

Another significant theoretical advancement came in 1962 when Brian D. Josephson (above), a graduate student at Cambridge University, predicted that electrical current would flow between 2 superconducting materials - even when they are separated by a non-superconductor or insulator. His prediction was later confirmed and won him a share of the 1973 Nobel Prize in Physics. This tunneling phenomenon is today known as the "Josephson effect" and has been applied to electronic devices such as the SQUID, an instrument capable of detecting even the weakest magnetic fields.

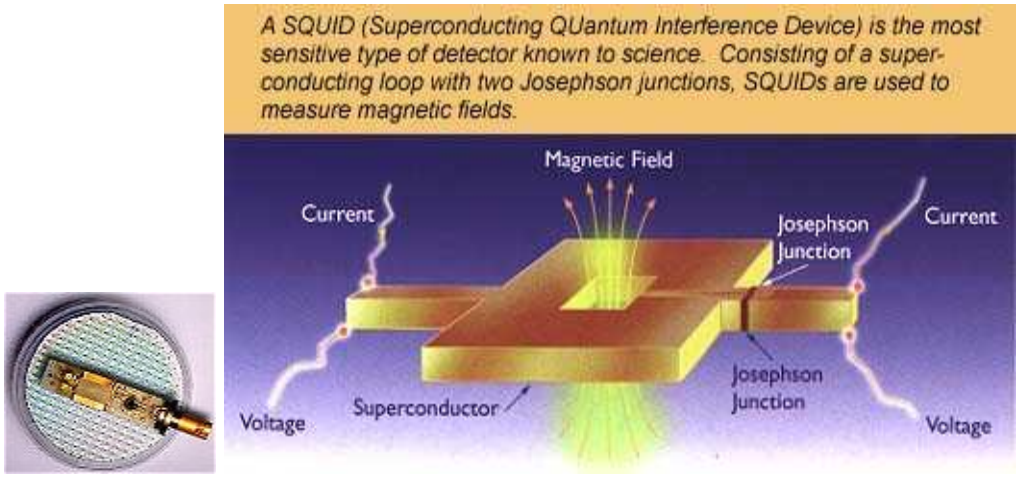


Figure 5: SQUID graphic courtesy Quantum Design

The 1980's were a decade of unrivaled discovery in the field of superconductivity. In 1964 Bill Little of Stanford University had suggested the possibility of organic (carbon-based) superconductors. The first of these theoretical superconductors was successfully synthesized in 1980 by Danish researcher Klaus Bechgaard of the University of Copenhagen and 3 French team members. $(TMTSF)_2PF_6$ had to be cooled to an incredibly cold 1.2K transition temperature

(known as Tc) and subjected to high pressure to superconduct. But its mere existence proved the possibility of "designer" molecules - molecules fashioned to perform in a predictable way.

Figure 6: Alex Müller and Georg Bednorz

Then, in 1986, a truly breakthrough discovery was made in the field of superconductivity. Alex Müller and Georg Bednorz, researchers at the IBM Research Laboratory in Rüschlikon, Switzerland, created a brittle ceramic compound that superconducted at the highest temperature then known: 30 K. What made this discovery so remarkable was that ceramics are normally insulators. They don't conduct electricity well at all. So, researchers had not considered them as possible high-temperature superconductor candidates. The lanthanum, barium, copper and oxygen compound that Müller and Bednorz synthesized, behaved in a not-as-yet-understood way. The discovery of this first of the superconducting copper-oxides (cuprates) won the 2 men a Nobel Prize the following year. It was later found that tiny amounts of this material were actually superconducting at 58 K, due to a small amount of lead having been added as a calibration standard - making the discovery even more noteworthy.

Müller and Bednorz' discovery triggered a flurry of activity in the field of superconductivity. Researchers around the world began "cooking" up ceramics of every imaginable combination in a quest for higher and higher Tc's. In January of 1987 a research team at the University of Alabama-Huntsville substituted yttrium for lanthanum in the Müller and Bednorz molecule and achieved an incredible 92 K Tc. For the first time a material (today referred to as YBCO) had been found that would superconduct at temperatures warmer than liquid Nitrogen - a commonly available coolant. Additional milestones have since been achieved using exotic - and often toxic - elements in the base perovskite ceramic. For more than 20 years the mercury copper-oxides

held the record for highest transition temperature at 138 K. However, recently the thallium copper-oxides have moved into the lead. When incorporated into a 3212 structure, thallium, barium, tellurium, copper and oxygen will produce a Tc near 147K with a purity comparable to the mercuric cuprates.

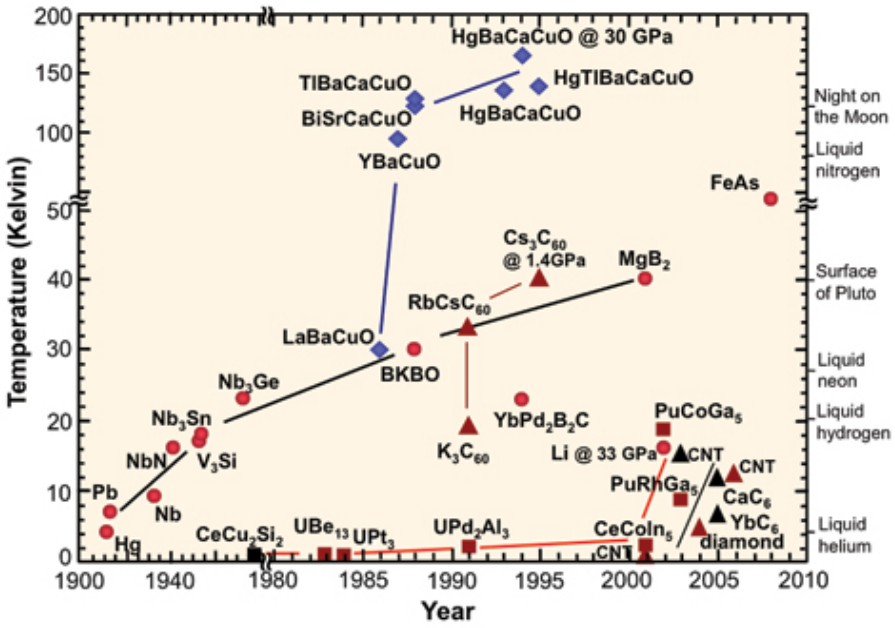


Figure 7: timeline showing the history of critical temperature discoveries

The graph illustrates a timeline showing the history of critical temperature discoveries. Materials with critical temperatures falling above the boiling point of liquid nitrogen (around 77 K) are known as high-temperature materials. The dramatic increase in T_c seen in the middle of the graph is the result of the discovery of superconductive cuprates and perovskites with high T_c in 1986 and 1987.

2.2. Physics of Superconductor

A superconductor is a material that can conduct electricity or transport electrons from one atom to another with no resistance. This means no heat, sound or any other form of energy would be released from the material when it has reached "critical temperature" (T_c), or the temperature at which the material becomes superconductive[102]. Unfortunately, most materials must be in an extremely low energy state (very cold) in order to become superconductive[32]. Research is underway to develop compounds that become superconductive at higher temperatures. Currently, an excessive amount of energy must be used in the cooling process making superconductors inefficient and uneconomical[33].

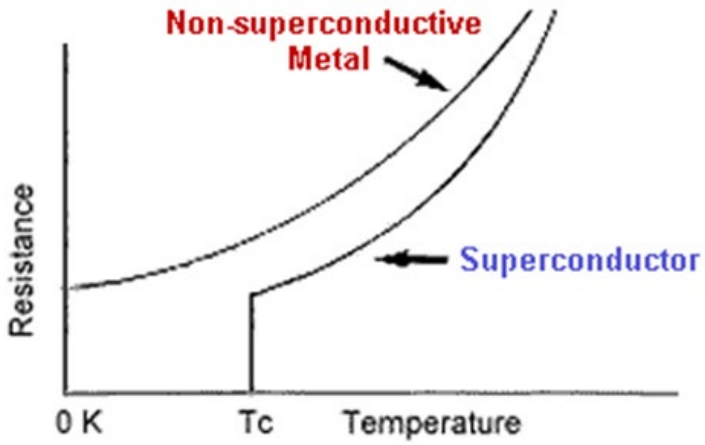


Figure 8:relation bet temp and resistance for non supercon and supercon materials (Tc: critical temp)

2.2.1. The Meissner effects

The second defining characteristic of a superconducting material is much less obvious than its zero electrical resistance. It was over 20 years after the discovery of superconductivity that Meissner and Ochsenfeld published a paper describing this second characteristic. They discovered that when a magnetic field is applied to a sample of tin, say, in the superconducting state, the applied field is excluded, so that $B = 0$ throughout its interior. This property of the superconducting state is known as the **Meissner effect**.[34]

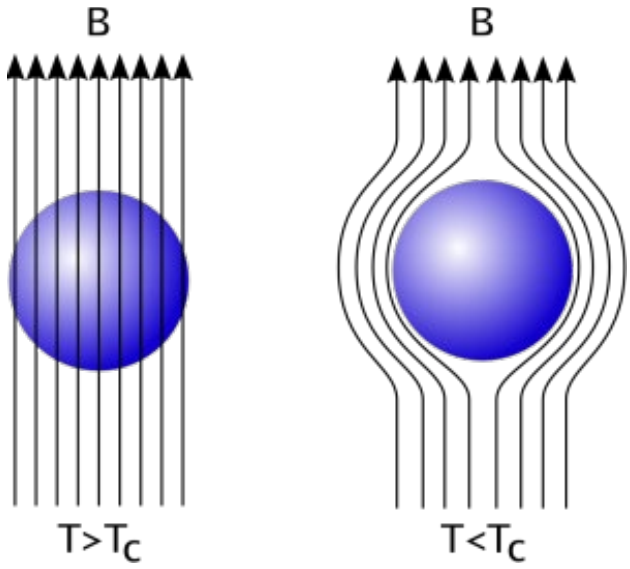


Figure 9: The Meissner effect: a superconductor's magnetic flux above (left) and below critical temperature.

2.2.2. Superconductors come in two different types

- **Type I Superconductors**

  A type I superconductor consists of basic conductive elements that are used in everything from electrical wiring to computer microchips. At present, type I superconductors have T_cs between 0.000325 °K and 7.8 °K at standard pressure. Some type I superconductors require incredible amounts of pressure in order to reach the superconductive state. One such material is sulfur which, requires a pressure of 9.3 million atmospheres (9.4×10^{11} N/m^2) and a temperature of 17 °K to reach superconductivity. Some other examples of type I superconductors include Mercury - 4.15 °K, Lead - 7.2 °K, Aluminum - 1.175 °K and Zinc - 0.85 °K. Roughly half of the elements in the periodic table are known to be superconductive.[35]

- **Type II Superconductors**

  A type II superconductor is composed of metallic compounds such as copper or lead. They reach a superconductive state at much higher temperatures when compared to type I superconductors. The cause of this dramatic increase in temperature is not fully understood. The highest T_c reached at stardard pressure, to date, is 135 °K or -138 °C by a compound ($HgBa_2Ca_2Cu_3O_8$) that falls into a group of superconductors known as cuprate perovskites. This group of superconductors generally has a ratio of 2 copper

atoms to 3 oxygen atoms, and is considered to be a ceramic. Type II superconductors can also be penetrated by a magnetic field whereas a type I cannot. [35]

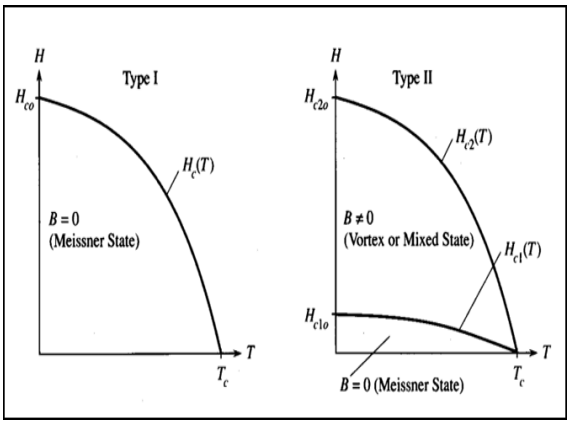


Figure 10:Magnetic phase diagram for type-I and type-2 superconductor

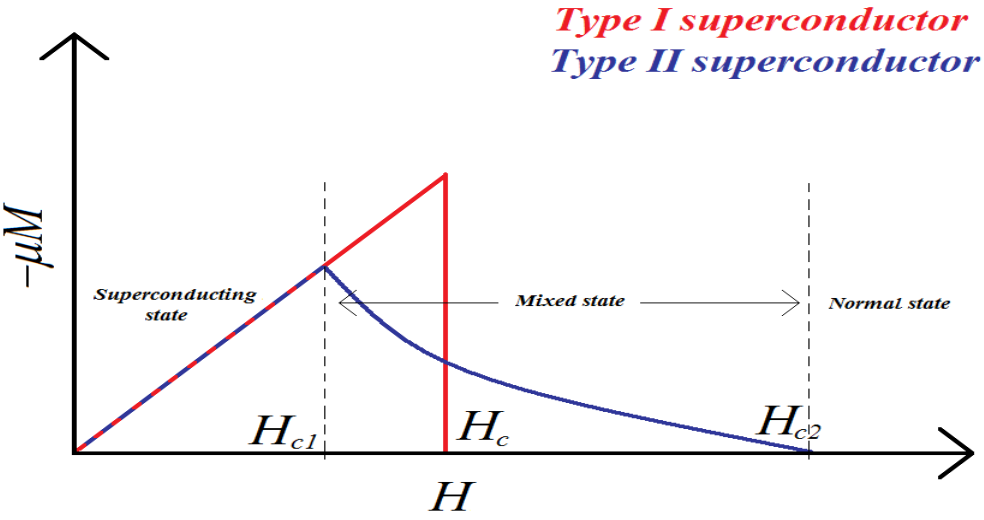


Figure 11:: The induction in the long cylinder as a function of the applied field for Type I and Type II superconductors; (b) The reversible magnetization curve of a long cylinder of Type I and Type II superconductor (After De Gennes, 1966)

2.3. Applications of Superconductor

Superconductors are not available on a wide commercial scale due to the extensive cooling necessary to reach superconductive states. They are common in a few specialized applications, including:

- **MAGLEV trains** use superconductive magnets to practically eliminate friction between the train and the tracks. The use of conventional electromagnets would waste vast quantities of energy via heat loss and necessitate the use of an unwieldy magnet, whereas superconductors result in superior efficiency and smaller magnets.[107]
- **Magnetic resonance imaging (MRI)** uses superconductor-generated magnetic fields to interact with hydrogen atoms and fat molecules within the human body. These atoms and molecules then release energy that is detected and formed into a graphic image. MRI is a widely used radiographic method for medical diagnosis or staging of diseases such as cancer. [36]
- **Electric generators** built with superconductive wire have achieved 99% efficiency ratings in experimental tests but have yet to be built commercially. [36]
- **Electric power generation** using superconductive cables and transformers has been experimentally tested and demonstrated. [36]
- **Quantum computing** many quantum computing researchers are adapting the technology of superconductors into the use for creating advanced quantum computers. [36]

2.4. Machine Learning.

Machine Learning Theory, also known as Computational Learning Theory, aims to understand the fundamental principles of learning as a computational process. This field seeks to understand at a precise mathematical level what capabilities and information are fundamentally needed to learn different kinds of tasks successfully[37], and to understand the basic algorithmic principles involved in getting computers to learn from data and to improve performance with

feedback. The goals of this theory are both to aid in the design of better automated learning methods and to understand fundamental issues in the learning process itself. Machine Learning Theory draws elements from both the Theory of Computation and Statistics and involves tasks such as:

• Creating mathematical models that capture key aspects of machine learning, in which one can analyze the inherent ease or difficulty of different types of learning problems.

• Proving guarantees for algorithms (under what conditions will they succeed, how much data and computation time is needed) and developing machine learning algorithms that provably meet desired criteria.

• Mathematically analyzing general issues, such as: “Why is Occam’s Razor a good idea?”, “When can one be confident about predictions made from limited data?”, “How much power does active participation add over passive observation for learning?”, and “What kinds of methods can learn even in the presence of large quantities of distracting

2.4.1. Basic Principles of Machine Learning

Machine learning algorithms aim to optimize the performance of a certain task by using examples and/or past experience.[38] Generally speaking, machine learning can be divided into three main categories, namely, supervised learning, unsupervised learning, and reinforcement learning. Supervised machine learning is based on the same principles as a standard fitting procedure: it tries to find the unknown function that connects known inputs to unknown outputs. This desired result for unknown domains is estimated based on the extrapolation of patterns found in the labeled training data. Unsupervised learning is concerned with finding patterns in unlabeled data, as, e.g., in the clustering of samples. Finally, reinforcement learning treats the problem of finding optimal or sufficiently good actions for a situation in order to maximize a reward.[81] In other words, it learns from interactions. Finally, halfway between supervised and unsupervised learning lies semi-supervised learning. In this case, the algorithm is provided with both unlabeled as well as labeled data. Techniques of this category are particularly useful when available data are incomplete and to learn representations.[39]

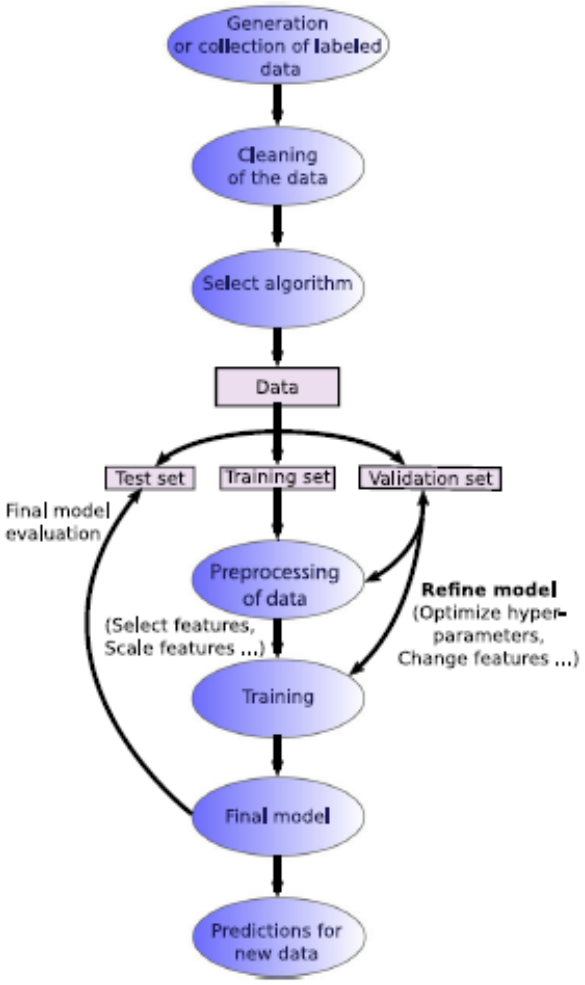


Figure 12:supervised learning workflow

As supervised learning is by far the most widespread form of machine learning in materials science, we will concentrate on it in the following discussion. Figure 1 depicts the workflow applied in supervised learning. One generally chooses a subset of the relevant population for which values of the target property are known or creates the data if necessary. This process is accompanied by the selection of a machine learning algorithm that will be used to fit the desired target quantity. Most of the work consists in generating, finding, and cleaning the data to ensure that it is consistent, accurate, etc. Second, it is necessary to decide how to map the properties of the system, i.e., the input for the model, in a way that is suitable for the chosen algorithm. This implies to translate the raw information into certain features that

will be used as inputs for the algorithm. Once this process is finished, the model is trained by optimizing its performance, usually measured through some kind of cost function. Usually this entails the adjustment of hyperparameters that control the training process, structure, and properties of the model. The data are split into various sets. Ideally, a validation dataset separates from the test and training sets is used for the optimization of the hyperparameters

2.4.2. Advances of Machine Learning in Physics

In recent years, the availability of large datasets combined with the improvement in algorithms and the exponential growth in computing power led to an unparalleled surge of interest in the topic of machine learning. Nowadays, machine learning algorithms are successfully employed for classification, regression, clustering, or dimensionality reduction tasks of large sets of especially high-dimensional input data.[39] In fact, machine learning has proved to have superhuman abilities in numerous fields (such as playing go,[40] self-driving cars,[41] image classification,[42] etc). As a result, huge parts of our daily life, for example, image and speech recognition,[43,44] web-searches,[45] fraud detection,[46] email/spam filtering, [47] credit scores,[48] and many more are powered by machine learning algorithms. While data-driven research, and more specifically machine learning, have already a long history in biology[49] or chemistry,[50] they only rose to prominence recently in the field of solid-state materials science. Traditionally, experiments used to play the key role in finding and characterizing new materials. Experimental research must be conducted over a long time period for an extremely limited number of materials, as it imposes high requirements in terms of resources and equipment. Owing to these limitations, important discoveries happened mostly through human intuition or even serendipity.[51] A first computational revolution in materials science was fueled by the advent of computational methods,[52] especially density functional theory (DFT),[53,54] Monte Carlo simulations, and molecular dynamics, that allowed researchers to explore the phase and composition space far more efficiently. In fact, then combination of both experiments and computer simulations has allowed to cut substantially the time and cost of materials design.[55-56] The constant increase in computing power and the development of more efficient codes also allowed for computational high-throughput studies[57] of large material groups in order

to screen for the ideal experimental candidates. These large-scale simulations and calculations together with experimental high throughput studies are producing an enormous amount of data making possible the use of machine learning methods to materials science.
As these algorithms start to find their place, they are heralding a second computational revolution. Because the number of possible materials is estimated to be as high as a googol (10100), this revolution is doubtlessly required. This paradigm change is further promoted by projects like the materials genome initiative (Materials genome initiative) that aim to bridge the gap between experiment and theory and promote a more data-intensive and systematic research approach. A multitude of already successful machine learning applications in materials science can be found, e.g., the prediction of new stable materials, the calculation of numerous material properties, and the speeding up of first principal calculations.

2.4.3. Algorithms

In this section, we briefly introduce and discuss Algorithms used in this research we used many regression algorithms we start by linear regression ending with ensemble random forest going by (lasso, ridge regression, elastic-net, Bayseian regression, KNN

- **Linear Regression**

It is used to estimate real values (cost of houses, number of calls, total sales etc.) based on continuous variable(s). Here, we establish relationship between independent and dependent variables by fitting a best line. This best fit line is known as regression line and represented by a linear equation Y= a *X + b.

The best way to understand linear regression is to relive this experience of childhood. Let us say, you ask a child in fifth grade to arrange people in his class by increasing order of weight, without asking them their weights! What do you think the child will do? He / she would likely look (visually analyze) at the height and build of people and arrange them using a combination of these visible parameters. This is linear regression in real life! The child has actually figured out that height and build would be correlated to the weight by a relationship, which looks like the equation above.

In this equation:

- Y – Dependent Variable
- a – Slope
- X – Independent variable
- b – Intercept

These coefficients a and b are derived based on minimizing the sum of squared difference of distance between data points and regression line.

Look at the below example. Here we have identified the best fit line having linear equation **y=0.2811x+13.9**. Now using this equation, we can find the weight, knowing the height of a person.

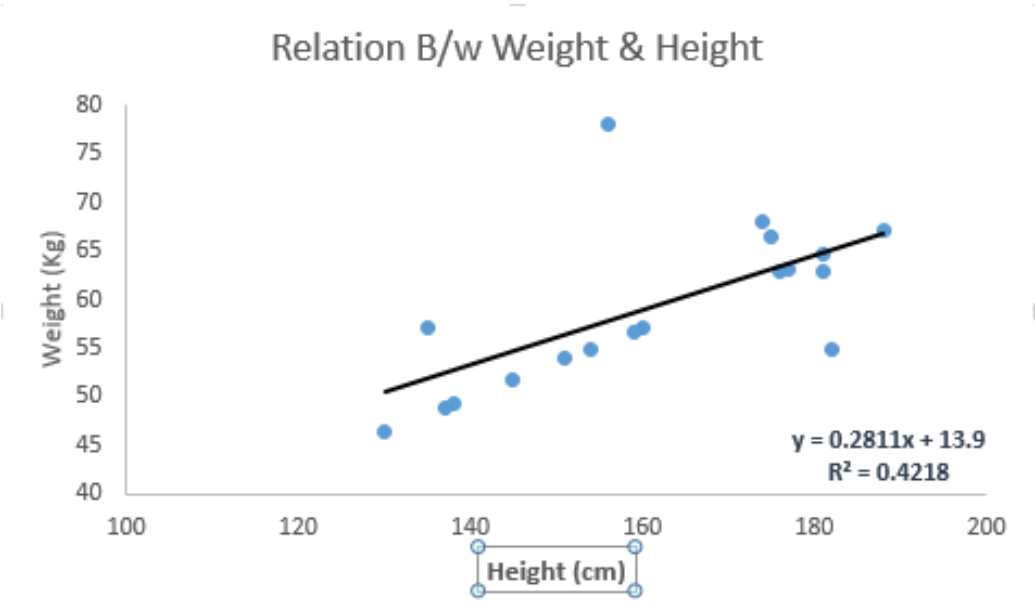


Figure 13: relation between weight and Hight

Linear Regression is mainly of two types: Simple Linear Regression and Multiple Linear Regression. Simple Linear Regression is characterized by one independent variable. And, Multiple Linear Regression (as the name suggests) is characterized by multiple (more than 1)

independent variables. While finding the best fit line, you can fit a polynomial or curvilinear regression. And these are known as polynomial or curvilinear regression.

- **Lasso**

The **Lasso** is a linear model that estimates sparse coefficients. It is useful in some contexts due to its tendency to prefer solutions with fewer non-zero coefficients, effectively reducing the number of features upon which the given solution is dependent. For this reason, Lasso and its variants are fundamental to the field of compressed sensing. Under certain conditions, it can recover the exact set of non-zero coefficients.

Mathematically, it consists of a linear model with an added regularization term. The objective function to minimize is:

$$\min_w \frac{1}{2n_{\text{samples}}} ||Xw - y||_2^2 + \alpha||w||_1$$

The lasso estimate thus solves the minimization of the least-squares penalty with α||w||1 added, where α is a constant and ||w||1 is the ℓ1-norm of the coefficient vector.
The implementation in the class **Lasso** uses coordinate descent as the algorithm to fit the coefficients

- **Ridge regression**

Ridge regression addresses some of the problems of Ordinary Least Squares by imposing a penalty on the size of the coefficients. The ridge coefficients minimize a penalized residual sum of squares:

$$\min_w ||Xw - y||_2^2 + \alpha||w||_2^2$$

The complexity parameter α≥0 controls the amount of shrinkage: the larger the value of α, the greater the amount of shrinkage and thus the coefficients become more robust to collinearity.

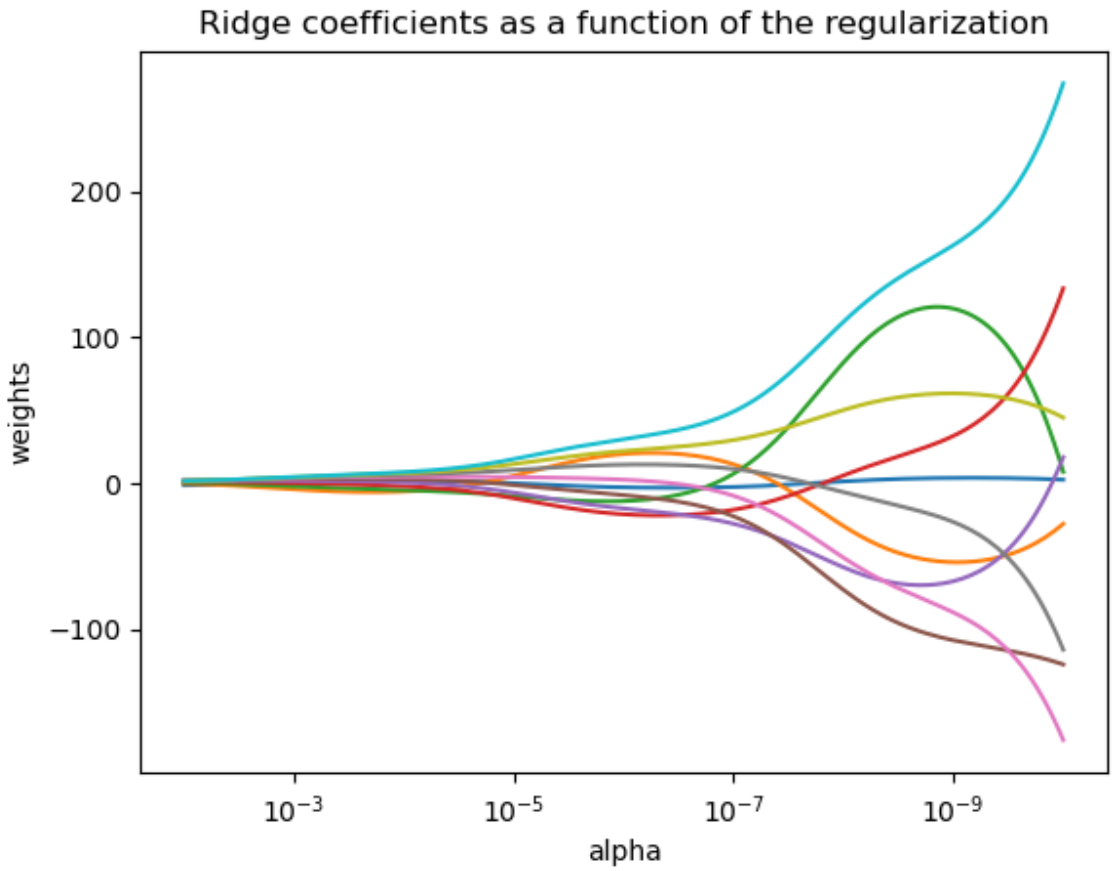


Figure 14: ridge coefficients as function of regularization

As with other linear models, **Ridge** will take in its `fit` method arrays X, y and will store the coefficients w of the linear model in its `coef_` member:

- **Elastic-Net**

Elastic-Net is a linear regression model trained with both ℓ1 and ℓ2-norm regularization of the coefficients. This combination allows for learning a sparse model where few of the weights are non-zero like **Lasso**, while still maintaining the regularization properties of **Ridge**. We control the convex combination of ℓ1 and ℓ2 using the `l1_ratio` parameter.

Elastic-net is useful when there are multiple features which are correlated with one another. Lasso is likely to pick one of these at random, while elastic-net is likely to pick both.

A practical advantage of trading-off between Lasso and Ridge is that it allows Elastic-Net to inherit some of Ridge's stability under rotation.

The objective function to minimize is in this case

$$\min_{w} \frac{1}{2n_{\text{samples}}} ||Xw - y||_2^2 + \alpha\rho||w||_1 + \frac{\alpha(1-\rho)}{2}||w||_2^2$$

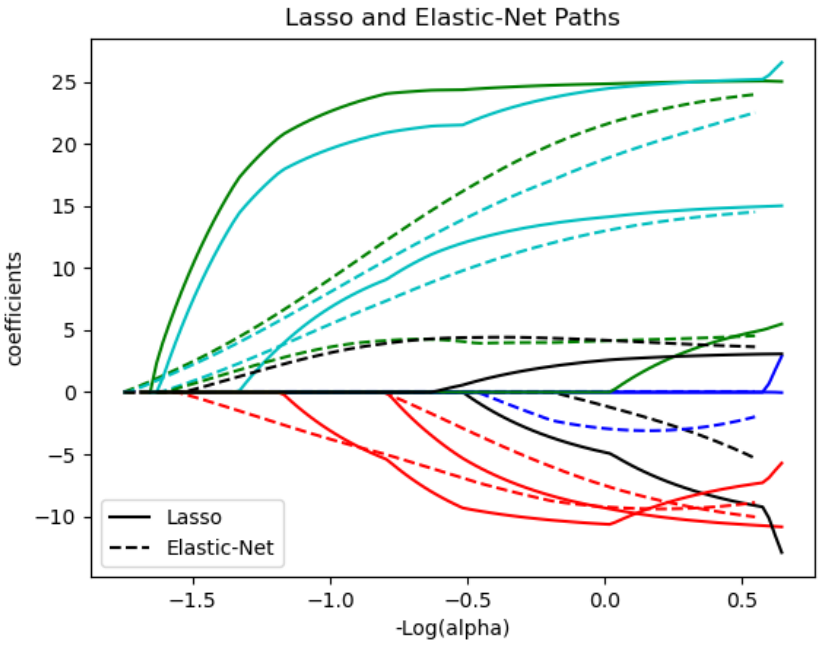


Figure 15: lasso and elastic-net paths

The class **Elastic-NetCV** can be used to set the parameters alpha (α) and l1_ratio (ρ) by cross-validation.

- **Bayesian regression**

Bayesian regression techniques can be used to include regularization parameters in the estimation procedure: the regularization parameter is not set in a hard sense but tuned to the data at hand.

This can be done by introducing uninformative priors over the hyper parameters of the model. The ℓ2 regularization used in Ridge regression and classification is equivalent to finding a maximum a posteriori estimation under a Gaussian prior over the coefficients w with precision λ−1. Instead of setting lambda manually, it is possible to treat it as a random variable to be estimated from the data.

To obtain a fully probabilistic model, the output y is assumed to be Gaussian distributed around Xw:

$$p(y|X,w,\alpha) = \mathcal{N}(y|Xw,\alpha)$$

where α is again treated as a random variable that is to be estimated from the data.

The advantages of Bayesian Regression are:

- It adapts to the data at hand.
- It can be used to include regularization parameters in the estimation procedure.

The disadvantages of Bayesian regression include:

- Inference of the model can be time consuming.

- **KNN**

he principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point and predict the label from these. The number of samples can be a user-defined constant (k-nearest neighbor learning) or vary based on the local density of points (radius-based neighbor learning). The distance can, in general, be any metric measure: standard Euclidean distance is the most common choice. Neighbors-based methods are known as *non-generalizing* machine learning methods, since they simply "remember" all of its training data (possibly transformed into a fast indexing structure such as a Ball Tree or KD Tree).

Despite its simplicity, nearest neighbors have been successful in a large number of classification and regression problems, including handwritten digits and satellite image scenes. Being a non-parametric method, it is often successful in classification situations where the decision boundary is very irregular.

The classes in **sklearn.neighbors** can handle either NumPy arrays or scipy.sparse matrices as input. For dense matrices, a large number of possible distance metrics are supported. For sparse matrices, arbitrary Minkowski metrics are supported for searches.

There are many learning routines which rely on nearest neighbors at their core. One example is kernel density estimation, discussed in the density estimation section.

Neighbors-based regression can be used in cases where the data labels are continuous rather than discrete variables. The label assigned to a query point is computed based on the mean of the labels of its nearest neighbors.

scikit-learn implements two different neighbors regressors: **KNeighborsRegressor** implements learning based on the k nearest neighbors of each query point, where k is an integer value specified by the user. **RadiusNeighborsRegressor** implements learning based on the neighbors within a fixed radius r of the query point, where r is a floating-point value specified by the user.

The basic nearest neighbor's regression uses uniform weights: that is, each point in the local neighborhood contributes uniformly to the classification of a query point. Under some circumstances, it can be advantageous to weight points such that nearby points contribute more to the regression than faraway points. This can be accomplished through the weights keyword. The default value, weights = 'uniform', assigns equal weights to all points. weights = 'distance' assigns weights proportional to the inverse of the distance from the query point. Alternatively, a user-defined function of the distance can be supplied, which will be used to compute the weights.

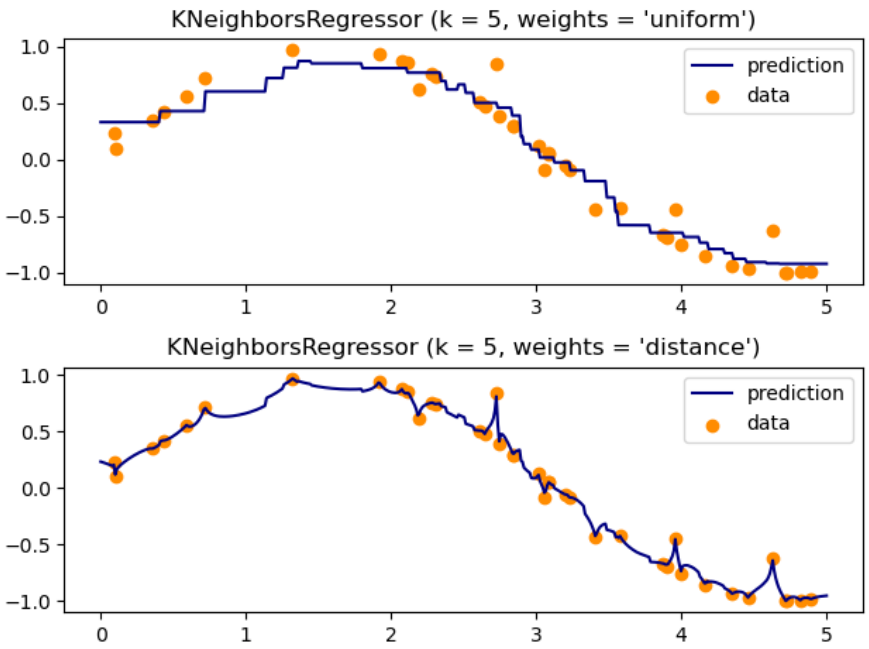


Figure 16: KNN regressor

- **Ensemble Methods**

The goal of **ensemble methods** is to combine the predictions of several base estimators built with a given learning algorithm in order to improve generalizability / robustness over a single estimator.

Two families of ensemble methods are usually distinguished:

- In **averaging methods**, the driving principle is to build several estimators independently and then to average their predictions. On average, the combined estimator is usually better than any of the single base estimator because its variance is reduced.

  Examples: Bagging methods, Forests of randomized trees, …

- By contrast, in **boosting methods**, base estimators are built sequentially, and one tries to reduce the bias of the combined estimator. The motivation is to combine several weak models to produce a powerful ensemble.

Examples: AdaBoost, Gradient Tree Boosting, …

- **Random Forest**

The Random Forest algorithm and the Extra-Trees method. Both algorithms are perturb-and-combine techniques [B1998] specifically designed for trees. This means a diverse set of classifiers is created by introducing randomness in the classifier construction. The prediction of the ensemble is given as the averaged prediction of the individual classifiers.

As other classifiers, forest classifiers have to be fitted with two arrays: a sparse or dense array X of shape (n_samples, n_features) holding the training samples, and an array Y of shape (n_samples,) holding the target values (class labels) for the training samples:

each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set.

Furthermore, when splitting each node during the construction of a tree, the best split is found either from all input features or a random subset of size max_features. (See the parameter tuning guidelines for more details).

The purpose of these two sources of randomness is to decrease the variance of the forest estimator. Indeed, individual decision trees typically exhibit high variance and tend to overfit. The injected randomness in forests yield decision trees with somewhat decoupled prediction errors. By taking an average of those predictions, some errors can cancel out. Random forests achieve a reduced variance by combining diverse trees, sometimes at the cost of a slight increase in bias. In practice the variance reduction is often significant, hence yielding an overall better model.

In contrast to the original publication [B2001], the scikit-learn implementation combines classifiers by averaging their probabilistic prediction, instead of letting each classifier vote for a single class.

2.5 Importance of Databases

Machine learning in materials science is mostly concerned with supervised learning. The success of such methods depends mainly on the amount and quality of data that is available, and this turns out to be one of the major challenges in material informatics.[90]

This is especially problematic for target properties that can only be determined experimentally in a costly fashion (such as the critical temperature of superconductors). For this reason, databases such as the materials project,[91] the inorganic crystal structure database,[92] and others (Materials genome initiative, The NOMAD archive, Supercon, National Institute of Materials Science 2011)[93-105] that contain information on numerous properties of known materials are essential for the success of materials informatics.

In order for these databases and for materials informatics to thrive, a FAIR treatment of data[106] is absolutely required. A FAIR treatment encompasses the four principles: findability, accessibility, interoperability, and repurposability.[107] In other words, researchers from different disciplines should be able to find and access data, as well as the corresponding metadata, in a commonly accepted format. This allows the application of the data for new purposes.

Traditionally, negative results are often discarded and left unpublished. However, as negative data are often just as important for machine learning algorithms as positive results ,[108] a cultural adjustment toward the publication of unsuccessful research is necessary. In some disciplines with a longer tradition of data-based research (like chemistry), such databases already exist.[108] In a similar vein, data that emerges as a side product but are not essential for a publication are often left unpublished. This eventually results in a waste of resources as other researchers are then required to repeat the work. In the end, every single discarded calculation will be sorely missed in future machine learning applications.

Chapter Three

3 METHODOLOGY.

3. Framework

We choice this framework in order to start our project with a clear vision and define method.
Our work frame that we going to follow is specifically design for machine learning projects. Since there will be a lot of tires and error, we decided to continue using agile methodology is still emerging today: agile development. These programming-centric methodologies have few rules and practices, all of which are fairly easy to follow, they focus on streamlining the SDLC by eliminating much of the modeling and documentation overhead and the time spent on those tasks. Instead, projects emphasize simple, iterative application development. Examples of agile development methodologies include extreme programming, Scrum, and the Dynamic Systems Development Method (DSDM). The agile development approach, as described next, typically is used in conjunction with object-oriented methodologies.[110]

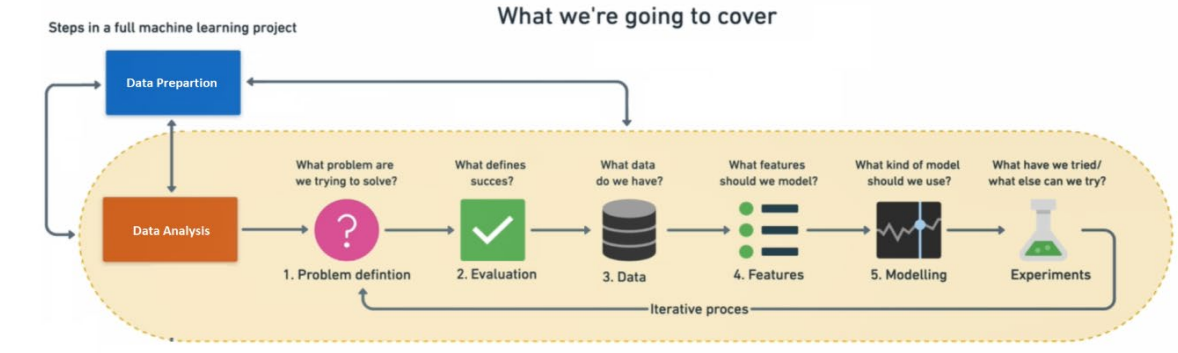


Figure 17: Framework for machine learning project

3.1. Problem Definition: What problem do we have?

Superconductivity is the focus of research since it was discovered by Dutch physicist Heike Kamerlingh Onnes of Leiden University In 1911 in 1913, he won a Nobel Prize its unique properties and promising technology even though some features of this unique phenomenon remain poorly understood; prime among these is the connection between superconductivity and chemical/structural properties of materials. In order to fill that gap, we want build a machine learning model that predicts the critical temperatures (Tc) of superconducting material from the 21,000+ known superconductors available via the SuperCon database

3.2. Evaluation: What defines success?

The goal that we build a functional model with lowest possible Root-mean-squared-error (RMSE) < ±9.9 Kelvin and the highest possible R2 score at least 0.90 That excide pervious Research.

3.3. Data: What Data do we have?

The success of any Machine Learning method ultimately depends on access to reliable and plentiful data. Superconductivity data used in this work is extracted from the SuperCon database,[18] created and maintained by the Japanese National Institute for Materials Science. It houses information such as the Tc and reporting journal publication for superconducting materials known from experiment. Assembled within it is a uniquely exhaustive list of all reported superconductors, as well as related non superconducting compounds. As such, SuperCon is the largest database of its kind, and has never before been employed in masse for machine learning modeling. After some data preprocessing, 21,263 superconductors are used.

3.3.1 Data Preparation

This section describes the detailed steps for the data preparation and feature extraction. Subsection (2.1) describes how the element data is obtained and processed. Subsection (2.2) describes the data preparation from NIMS Superconducting Material Database.

Subsection (2.3) details how the features are extracted.

3.3.1.1 Element Data Preparation

The element data with 46 variables and 86 rows (corresponding to 86 elements) are obtained by using the ElementData function from Mathematica Version 11.1 [111] .Appendix (A) lists the information sources for the element properties used by ElementData.
The first ionization energy data came from [112] and is merged with the Mathematica data. About 12% of the entries out of the 3956 (= 46 _ 86) entries are missing.
In choosing the properties, we are guided by Conder (2016)[113] but we also use our judgement to pick certain properties. For example, we drop the boiling point variable, and instead use the fusion heat variable which has no missing values, and is highly correlated with the boiling point variable.
We had also gained some experience and insight creating some initial models for predicting Tc of elements only. We settle on 8 properties shown in table (1).
With the choice of the above variables, we are only missing the atomic radii of La and Ce;
we replace them with their covalent radii since atomic radii and covalent radii have very high correlation (≈ 0.95) and approximately on the same scale and range. Some bias may be introduced

Variable	**Units**	**Description**
Atomic Mass	atomic mass units (AMU)	total proton and neutron rest masses
First Ionization Energy	kilo-Joules per mole (kJ/mol)	energy required to remove a valence electron

Atomic Radius	picometer (pm)	calculated atomic radius
Density	kilograms per meters cubed (kg/m^3)	density at standard temperature and pressure
Electron Affinity	kilo-Joules per mole (kJ/mol)	energy required to add an electron to a neutral atom
Fusion Heat	kilo-Joules per mole (kJ/mol)	energy to change from solid to liquid without temperature change
Thermal Conductivity	watts per meter-Kelvin ($W/(m \times K)$)	thermal conductivity coefficient κ
Valence	no units	typical number of chemical bonds formed by the element

Table 1: This table shows the properties of an element which are used for creating features to predict T_c.

into our data with this minor imputation. We add a small constant of 1.5 to the electron affinity values of all the elements to prevent issues when taking logarithm of 0.

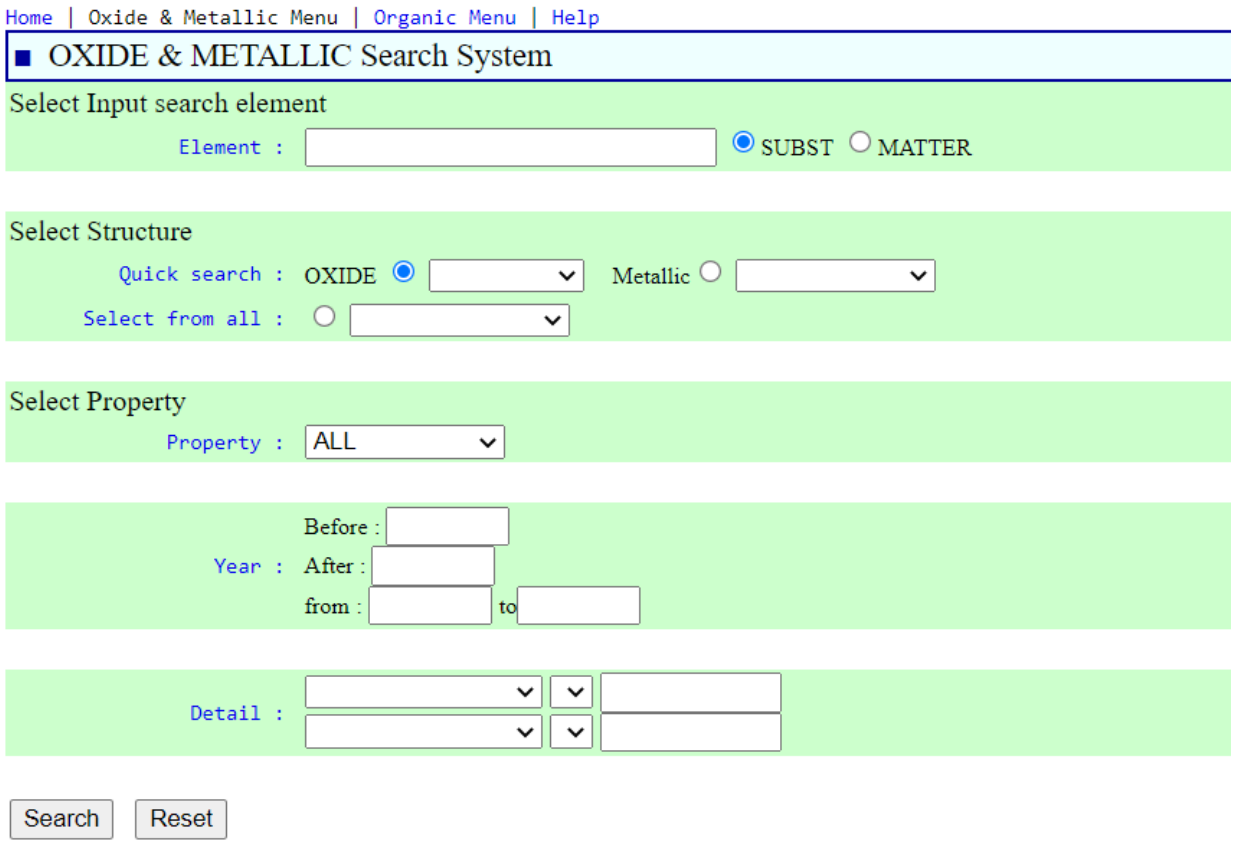


Figure 18 :: This is a screen shot of from Superconducting Material Database accessed on July 1, 2021.

3.3.1.2 Superconducting Material Data Preparation

Superconducting Material Database is supported by the NIMS, a public institution based in Japan. The database contains a large list of superconductors, their critical temperatures, and the source references mostly from journal articles. To our knowledge, this is the most comprehensive database of superconductors. Access to the database requires a login id and password but this is provided with a simple registration process.

We accessed the data on Jan 24, 2021 at SuperCon[114] Once logged in, we chose "OXIDE & METALLIC" material. Figure (18) shows a screen shot of the menu. We clicked on the "search" button to get *all* the data. We obtained 31,611 rows of data in a comma separated file format. The key columns (variables) were "element", the chemical formula of the material, and "Tc", the critical temperature. Variable "num" was a unique identifier for each row. Column "refno"

contained links to the referenced source. The next few steps describe the manual cleanup process:

1. We remove columns “ma1” to “mj2”.
2. We sort the data by “Tc” from the highest to lowest.
3. The critical temperature for the following “num” variables is mistakenly shifted by one column to the right. We fix these by recording them under the “Tc” column: 31020, 31021, 31022, 31023, 31024, 31025, 153150, 153149, 42170, 42171, 30716, 30717, 30718, 30719,150001, 150002, 150003, 150004, 150005, 150006, 150007, 30712, 30713, 30714, 30715.
4. The following are removed since the critical temperatures seemed to have been mis recorded; They have critical temperatures over 203 K which as of July 2017 was the highest reliable recorded critical temperature. La0.23Th0.77Pb3 (num = 111620), Pb2C1Ag2O6 (num = 9632), Er1Ba2Cu3O7-X (num = 140)
5. All rows with “Tc” = 0 or missing are removed.
6. Columns with headings “nums”, “mo1”, “mo2”, “oz”, “str3”, “tcn”, “tcfig”, “refno” are removed.
7. We manually change all materials with oxygen content formula such as O7-X to the best oxygen content approximation. For example, O7-X is changed to O7, O5+X is changed to O5, etc. This certainly introduces some error into our data but it is impossible to go document by document to get better estimates of the oxygen contents. At this point our data has two columns: “element” and “Tc”.
8. We use R statistical software by R Core Team (2017)[115] and the CHNOSZ package by Dick (2008)[6] to perform a preliminary check of the validity of the chemical formulas. The CHNOSZ package has a function makeup which reads the chemical formula in string format and breaks up the formula into the elements and their ratios.

In some cases, it throws an error or a warning when the chemical formula does not make sense. For example, it throws a warning message if Pb-2O is checked; Negative number of Pb does not make sense. However, the function does not check whether the material could actually exist. See figure (19) to get a sense of how this function works. With the help of the CHNOSZ package, we make the following modifications:

(a) Yo975Yb0.025Ba2Cu3O, Yo975Yb0.025Ba2Cu3O, Yo975Yb0.025Ba2Cu3O are removed. There is no element with the symbol Yo. It's likely that Y0.975 was mis recorded as Yo975 but we can't be sure.

(b) Bi1.7Pb0.3Sr2Ca1Cu2O0, La1.85Nd0Ca1.15Cu2O5.99, Bi0Mo0.33Cu2.67Sr2Y1O7.41,

(c) Y0.5Yb0.5Ba2Sr0Cu3O7 are removed since some elements had coefficients of zero.

(d) Y2C2Br0.5!1.5 is removed. The exclamation sign throws an error message.

(e) Y1Ba2Cu3O6050 is removed. The coefficient of 6050 for oxygen is possibly a mistake.

(f) Hg1234O10 is removed. The coefficient of 1234 for mercury is possibly a mistake.

(g) Nd185Ce0.15Cu1O4 is removed. The coefficient of 185 for Neodymium is possibly a mistake. There is a Nd1.85Ce0.15Cu1O4 already in the data.

(h) Bi1.6Pb0.4Sr2Cu3Ca2O1013 is changed to Bi1.6Pb0.4Sr2Cu3Ca2O10.13 since nearby rows in the data have formulas with O10. xx.

(i) Y1Ba2Cu285Ni0.15O7 is changed to Y1Ba2Cu2.85Ni0.15O7 since nearby rows in the data have formulas with Cu2.xx.

9. The column headings of "Tc" and "element" are changed to "critical temp" and "material" respectively.

6750 rows are left out because T_c is either zero or missing. At this point we have 24,861 rows of data.

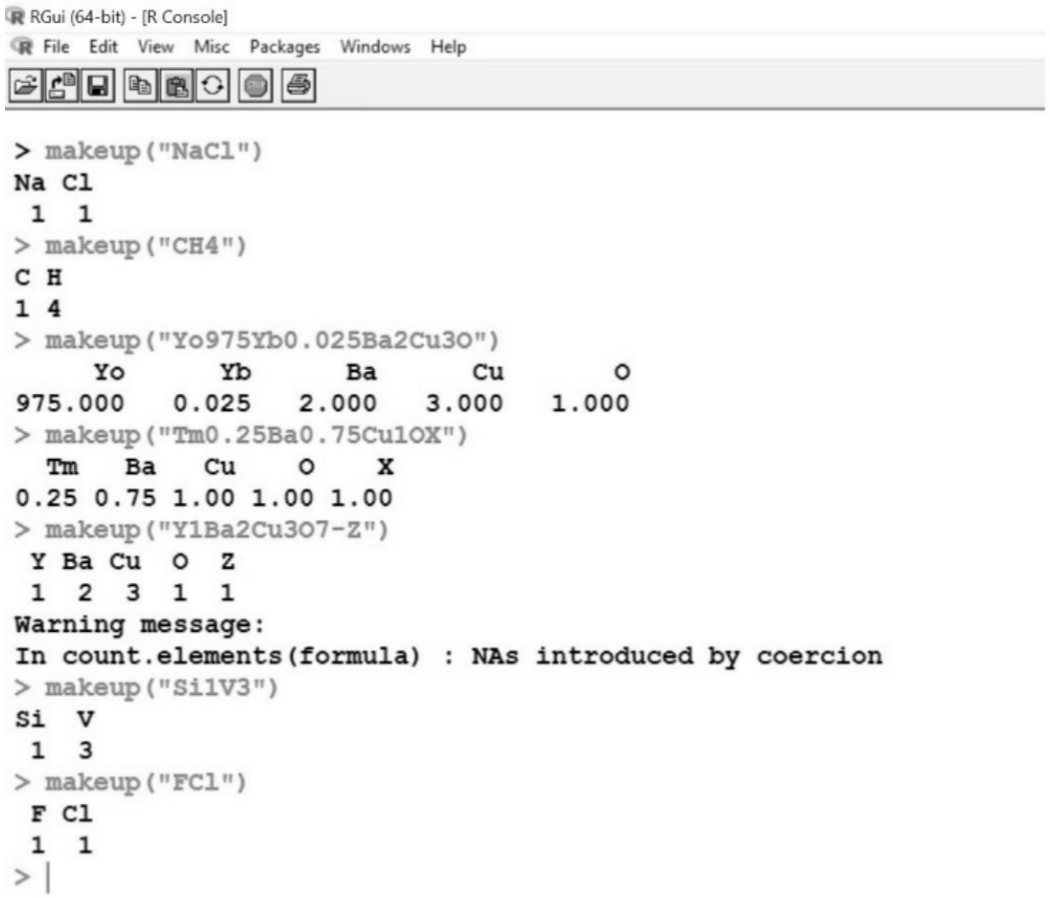


Figure 19[117]: : This screen shot is intended give you a sense of how the CHNOSZ package by Dick (2008) [116] works.

The first two materials NaCl and CH4 are correctly broken up. (These two are not superconductors and they are shown for illustration purposes.). $Yo_{975}Yb_{0.025}Ba_2Cu_3O$ was a material in the database but this is obviously a mistake since no element with the symbol Yo exists. The same is true for the next material with X. However, no warnings are issued. A warning is issued for $Y_1Ba_2Cu_3O_{7-Z}$. The next material SiV_3 was in the database and is correctly broken up. FCl is just given as another example. It is not a superconductor and was not in the database. The makeup command correctly breaks up the material but obviously does not check for the existence of FCl.

The rest of the data preparation is done in R Core Team (2017)[5]. We exclude any superconductor that has an element with an atomic number greater than 86. This eliminates an additional 973 rows of data. For example, superconductors that have uranium are left out. We remove the

repeating rows. It would be impossible to manually check to see whether the repeated rows are genuine independent reports from independent experiments or they are just duplicate reporting's. After all the data preparation and clean up, we end up with 21,263 rows of data or about 67% of the original data we started with.

Feature & Description	**Formula**	**Sample Value**
Mean	$= \mu = (t1 + t2)/2$	35.5
Weighted mean	$= v = (p1t1) + (p2t2)$	44.43
Geometric mean	$= (t1t2)1/2$	33.23
Weighted geometric mean	$= (t1)^{p1} (t2)^{p2}$	43.21
Entropy	$= -w1 \ln(w1) - w2 \ln(w2)$	0.63
Weighted entropy	$= -A \ln(A) - B \ln(B)$	0.26
Range	$= t1 - t2 \ (t1 > t2)$	25
Weighted range	$= p1t1 - p2t2$	37.86
Standard deviation	$= [(1/2) ((t1 - \mu)^2 + (t2 - \mu)^2)]^{1/2}$	12.5
Weighted standard deviation	$= [p1(t1 - v)^2 + p2(t2 - v)^2)]^{1/2}$	8.75

Table 2: This table summarizes the procedure for feature extraction from material's chemical formula.

The last column serves as an example; features based on thermal conductivities for Re_7Zr_1 are derived and reported to two decimal places. Rhenium and Zirconium's thermal conductivity coefficients are $t_1 = 48$ and $t_2 = 23$ W/(m×K) respectively.

Here: $p_1 = {}^{6}/_{7}$, $p_2 = {}^{1}/_{7}$, $\omega_1 = {}^{48}/_{71}$, $\omega_2 = {}^{23}/_{71}$, $A = \frac{p_1\omega_1}{p_1\omega_1 + p_2\omega_2} \approx 0.926$, $B = \frac{p_2\omega_2}{p_1\omega_1 + p_2\omega_2} \approx 0.074$.

3.3.1.3 Feature Extraction

In this section, we describe the feature extraction process through a detailed example: Consider Re_7Zr_1 with T_c = 6.7 K, and focus on the features extracted based on thermal conductivity. Rhenium and Zirconium's thermal conductivity coefficients are t_1 = 48 and t_2 = 23 W/(m×K) respectively. The ratios of the elements in the material are used to define features:

$$p_1 = \frac{6}{6+1} = {}^{6}/_{7}\ , p_2 = \frac{1}{6+1} = {}^{1}/_{7} \qquad (1)$$

The fractions of total thermal conductivities are used as well:

$$\omega_1 = \frac{t_1}{t_1 + t_2} = \frac{48}{48+23} = \frac{48}{71}\ ,\ \omega_2 = \frac{t_2}{t_1 + t_2} = \frac{23}{48+23} = \frac{23}{71} \qquad (2)$$

We need a couple of intermediate values based on equations (1) and (2):

$$A = \frac{p_1\omega_1}{p_1\omega_1 + p_2\omega_2} \approx 0.926\ , B = \frac{p_2\omega_2}{p_1\omega_1 + p_2\omega_2} \approx 0.074.$$

Once we have obtained the values p_1, p_2, w_1, w_2, A, and B, we can extract 10 features from Rhenium and Zirconium's thermal conductivities as shown in table (2).

We repeat the same process above with the 8 variables listed in table (1). For example, for features based on atomic mass, just replace t_1 and t_2 with the atomic masses of Rhenium and Zirconium respectively, then carry on with the calculations of p_1, p_2, w_1, w_2, A, B, and finally calculate the 10 features defined in table (2). This gives us 8 × 10 = 80 features. One additional feature, a numeric variable counting the number of elements in the superconductor, is also extracted. We end up with 81 features in total.

In summary: We have data with 21,263 rows and 82 columns: 81 columns corresponding to the features extracted and 1 column of the observed T_c values.

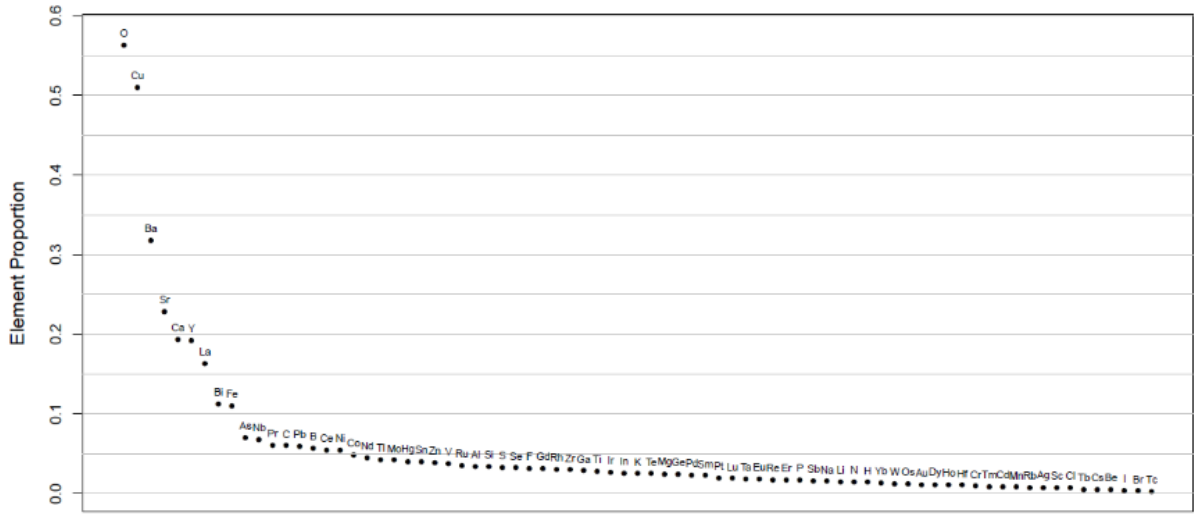


Figure 20:: This figure shows the proportions of the superconductors that had each element.

We also considered but did not implement features that simply indicate whether an element is present in the superconductor or not. For example, we could have had a column that indicated whether say oxygen is in the material or not. However, this approach would have added a large number of indicator variables to our data, made model selection and assessment too complicated, and increased the chances of over-fitting.

3.3.2 Data Analysis

This section has two parts: Basic summaries of the data are given in subsection (3.1).
The statistical models are described in subsection (3.2).

3.3.2.1. Descriptive Analysis

Figure (20) shows the proportions of the superconductors that had each element. For example, Oxygen is present in about 56% of the superconductors. Copper, barium, strontium, and calcium are the next most abundant elements.

Iron-based superconductors and cuprates are of particular interest in many research groups so

we report some summary statistics in table (3). Iron is present in approximately 11% of the superconductors. The mean *Tc* of superconductors with iron is 26:9 _ 21:4 K. The non-iron containing

superconductors' mean is 35:4 _ 35:4 K; the mean and standard deviations happened to be the same. A t-distribution based 95% confidence interval suggests that iron containing superconductors mean Tc is lower than the non-iron's by 7.4 to 9.5 K. Cuprates comprise approximately 49.5% of the superconductors. The cuprates mean Tc is 59:9 _ 31:2 K. The non-cuprates mean Tc is 9:5 _ 10:7 K. A t-distribution based 95% confidence interval indicates that the cuprates mean Tc

	Size	Min	Q1	Median	Q3	Max	Mean	SD
Iron	2339	0.02	11.3	21.7	35.5	130.0	26.9	21.4
Non-Iron	18924	0.0002	4.8	19.6	68.0	185.0	35.4	35.4
Cuprate	10532	0.001	31.0	63.1	86.0	143	59.9	31.2
Non-Cuprate	10731	0.0002	2.5	5.7	12.2	185	9.5	10.7

Table 3: This table reports summary statistics on iron-based versus non-iron, and cuprate versus non-cuprate superconductors.

The Size is the total number of observations of the material out of 21,263 materials. For example, 2,339 out of 21,263 materials contained iron. The rest of the columns report summary statistics for the observed critical temperatures (K): *min = minimum, Q1 = first quartile, Median = median, Q3 = third quartile, Max = maximum, and SD = standard*

Min	Q1	Median	Q3	Max	Mean	SD
0.00021	5.4	20	63	185.0	34.4	34.2

Table 4: This table reports the summary statistics for the critical temperatures values (K) of all 21,263 superconductors.

The column headers are the min = minimum, Q1 = first quartile, median, Q3 = third quartile, Max = maximum, and SD = standard deviation of the superconducting critical temperatures (K). is higher than the non-cuprates mean T_c by 49.8 to 51.0 K.

Figure (21) shows the histogram of T_c values. The values are right skewed with a bump around 80 K. Table (21) shows the summary statistics for T_c values.

Figure (22) shows the mean T_c grouped by elements. Mercury containing superconductors have the highest T_c at around 80 K on average. However, this is not the full story. Figure (23) shows the standard deviation of T_c grouped by elements. Although mercury containing superconductors have the highest T_c on average, these same materials show the fourth highest variability in T_c. In fact, a plot of the mean T_c versus the standard deviation of T_c in figure (24) shows that on average the higher the mean T_c, the higher the variability in T_c per element.

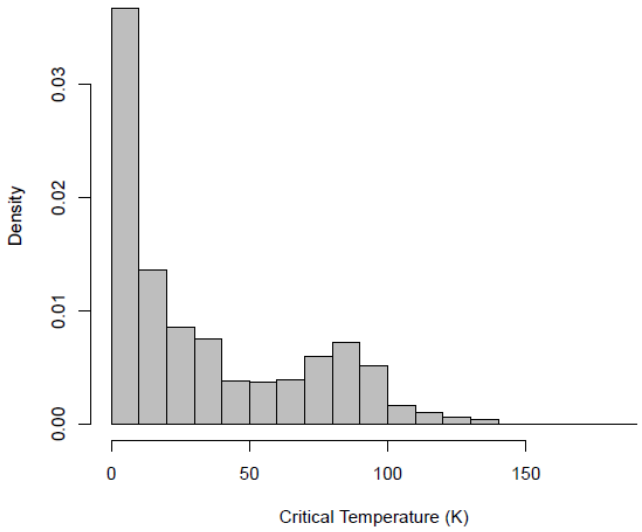


Figure 21:the distribution of the superconducting critical temperatures (K) of all 21,263 superconductors.

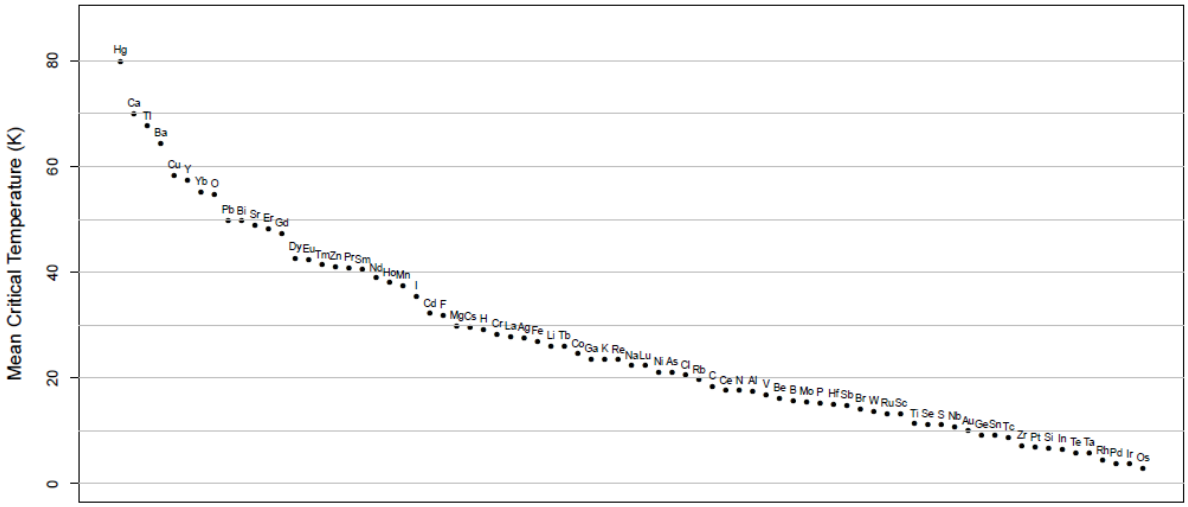


Figure 22: This Figure shows the mean superconducting critical temperature grouped by elements. On average, mercury containing materials had the highest superconducting critical temperature followed by calcium and so on.

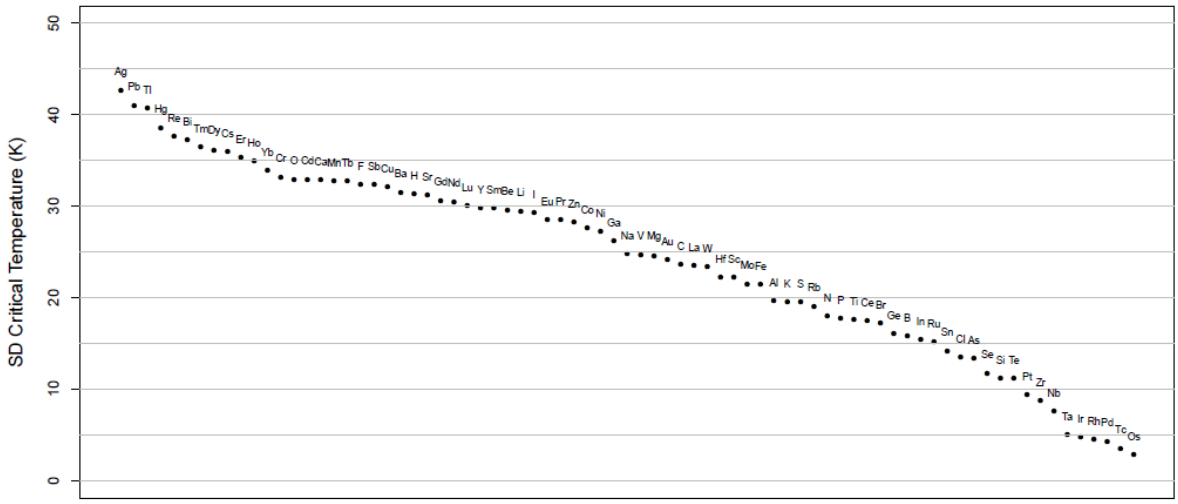


Figure 23: This Figure shows the standard deviation (SD) of critical temperature grouped by elements. Silver containing materials had the highest variability followed by lead and so on.

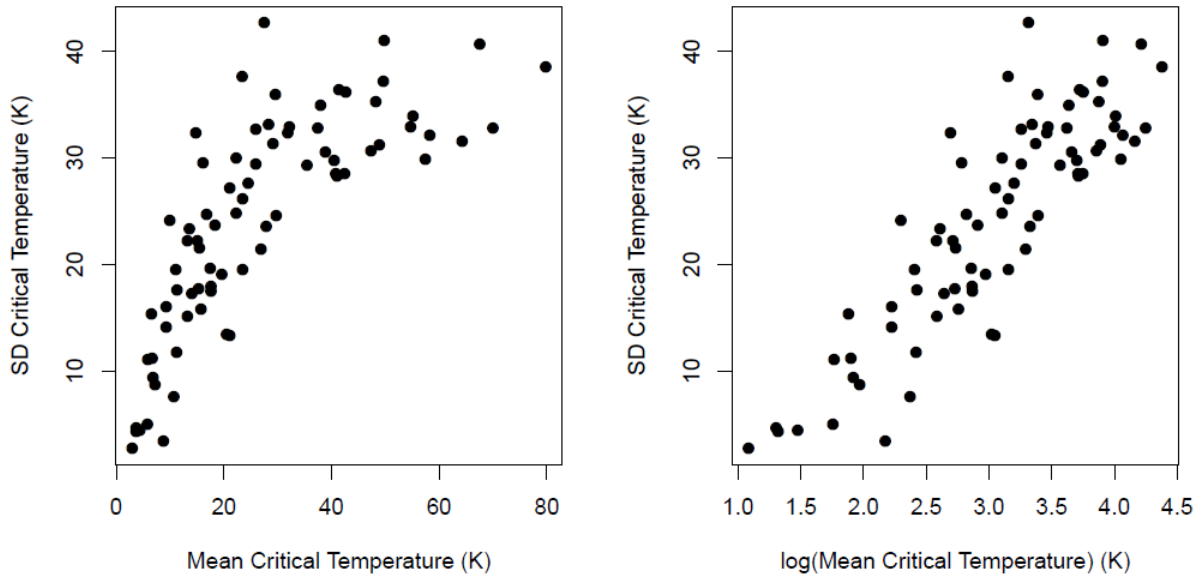


Figure 24:: The left panel shows the relationship between the mean critical temperature and standard deviation (SD) per element. The right panel shows the logarithm of the mean critical temperature versus SD. On average the higher the mean critical temperature, the higher the variability in critical temperature per element.

The average absolute value of the correlation among the features is 0.35. This indicates that the features are highly correlated. Motivated by this result, we attempted to reduce the dimensionality of the data using principal component analysis (PCA). However, our PCA analysis did not show any benefits in reducing the dimensionality since a large number of principal components were needed to capture a substantial percentage of the data variation; we abandoned the PCA approach.

3.4. Features Engineering: What features should we model?

Filtering out irrelevant features

We going to filter out irrelevant data Using Pandas Library, we start by merging the two .csv file Materials and Properties together.

Initially the data shape was 21263 rows by 168 features. The head of the original data frame looked like this;

```
In [1]: import pandas as pd
        import numpy as np
        Dataframe1 = pd.read_csv('Properties.csv')
        Other = pd.read_csv('Materials.csv').drop(['critical_temp','material'],axis=1)
        Dataframe = pd.concat([Dataframe1,Other],axis=1)
        original_columns = len(Dataframe.columns)
        print(Dataframe.shape)
        Dataframe.head()

        (21263, 168)
```

Out[1]:

	number_of_elements	mean_atomic_mass	wtd_mean_atomic_mass	gmean_atomic_mass	wtd_gmean_atomic_mass	entropy_atomic_mass	wtd_entropy_atomic
0	4	88.944468	57.862692	66.361592	36.116612	1.181795	1.(
1	5	92.729214	58.518416	73.132787	36.396602	1.449309	1.(
2	4	88.944468	57.885242	66.361592	36.122509	1.181795	0.9
3	4	88.944468	57.873967	66.361592	36.119560	1.181795	1.(
4	4	88.944468	57.840143	66.361592	36.110716	1.181795	1.1

5 rows × 168 columns

Figure 25: data shape before filtering.

Now we will Applying Filter Methods in Python for Feature Selection [118,119]

Machine learning and deep learning algorithms learn from data, which consists of different types of features. The training time and performance of a machine learning algorithm depends heavily on the features in the dataset. Ideally, we should only retain those features in the dataset that actually help our machine learning model learn something.

Unnecessary and redundant features not only slow down the training time of an algorithm, but they also affect the performance of the algorithm. The process of selecting the most suitable features for training the machine learning model is called "feature selection".

There are several advantages of performing feature selection before training machine learning models, some of which have been enlisted below:

- Models with a smaller number of features have higher explain ability
- It is easier to implement machine learning models with reduced features
- Fewer features lead to enhanced generalization which in turn reduces overfitting
- Feature selection removes data redundancy
- Training time of models with fewer features is significantly lower

- Models with fewer features are less prone to errors

Several methods have been developed to select the most optimal features for a machine learning algorithm. One category of such methods is called filter methods. In this article, we will study some of the basic filter methods for feature selection.

Filter Methods for Feature Selection

Filter's methods belong to the category of feature selection methods that select features independently of the machine learning algorithm model. This is one of the biggest advantages of filter methods. Features selected using filter methods can be used as an input to any machine learning models. Another advantage of filter methods is that they are very fast. Filter methods are generally the first step in any feature selection pipeline.

Filter methods can be broadly categorized into two categories: **Univariate Filter Methods** and **Multivariate filter methods**.

The univariate filter methods are the type of methods where individual features are ranked according to specific criteria. The top N features are then selected. Different types of ranking criteria are used for univariate filter methods, for example fisher score, mutual information, and variance of the feature.

One of the major disadvantages of univariate filter methods is that they may select redundant features because the relationship between individual features is not taken into account while making decisions. Univariate filter methods are ideal for removing constant and quasi-constant features from the data.

Multivariate filter methods are capable of removing redundant features from the data since they take the mutual relationship between the features into account. Multivariate filter methods can be used to remove duplicate and correlated features from the data.

In this article, we will see how we can remove constant, quasi-constant, duplicate, and correlated features from our dataset with the help of Python.

3.4.1. Removing Constant features

Constant features are the type of features that contain only one value for all the outputs in the dataset. Constant features provide no information that can help in classification of the record at hand. Therefore, it is advisable to remove all the constant features from the dataset. Constant features have values with zero variance since all the values are the same. We can find the constant columns using the VarianceThreshold function of Python's Scikit Learn Library. Execute the following script to import the required libraries and the dataset:

```
In [2]: # remove all columns that have a very small variance
        # this gets rid of features which are the same value 95% of the time
        threshold=0.05
        from sklearn.feature_selection import VarianceThreshold
        selector = VarianceThreshold(threshold=threshold).fit(Dataframe)
        Dataframe = pd.DataFrame(selector.transform(Dataframe),columns=Dataframe.columns[selector.get_support()])
        Dataframe.shape
Out[2]: (21263, 132)
```

Figure 26: data shape after removing constant features.

- This will Drop features that varied 5% of less across all rows (using VarianceThreshold) We ended up with 132 features.

3.4.2. Removing Duplicate Features

Duplicate features are the features that have similar values. Duplicate features do not add any value to algorithm training, rather they add overhead and unnecessary delay to the training time. Therefore, it is always recommended to remove the duplicate features from the dataset before training.

```
In [3]: #drop any duplicate features
        Dataframe = Dataframe.transpose().drop_duplicates(keep='first').transpose()
        print(Dataframe.shape)
        (21263, 132)
```

Figure 27: data shape after removing duplicate features.

- Drop duplicated features (using .transpose().drop_duplicates(keep='first').transpose()) There is Non Duplicated Features so there is nothing has change

3.4.3. Removing Correlated Features

In addition to the duplicate features, a dataset can also contain correlated features. Two or more than two features are correlated if they are close to each other in the linear space.

Take the example of the feature set for a fruit basket, the weight of the fruit basket is normally correlated with the price. The more the weight, the higher the price.

Correlation between the output observations and the input features is very important and such features should be retained. However, if two or more than two features are mutually correlated, they convey redundant information to the model and hence only one of the correlated features should be retained to reduce the number of features.

Removing Correlated Features using corr() Method

To remove the correlated features, we can make use of the corr() method of the pandas data frame. The corr() method returns a correlation matrix containing correlation between all the columns of the data frame. We can then loop through the correlation matrix and see if the correlation between two columns is greater than threshold correlation, add that column to the set of correlated columns. We can remove that set of columns from the actual dataset.

```
In [4]: #Now drop any correlated features
        correlation_matrix = Dataframe.corr()
        mutually_correlated_features = set()
        for i in range(len(correlation_matrix.columns)):
            for j in range(i):
                if abs(correlation_matrix.iloc[i, j]) > 0.8:
                    colname = correlation_matrix.columns[i]
                    if colname!='critical_temp':
                        mutually_correlated_features.add(colname)
        print(len(mutually_correlated_features))
        Dataframe.drop(mutually_correlated_features,axis=1,inplace=True)
        Dataframe.shape

        55

Out[4]: (21263, 77)
```

Figure 28: data shape after removing correlated features greater than threshold.

- Dropping features that were highly correlated with each other (greater than |+-0.8|) We ended up with 55 features.

Using corr() Method to Removing Correlated Features again this time to get rid of features that an absolute correlation with the target is less than 0.1

```
In [5]: #This gets rid of features that have a absolute correlation with the target less than 0.1
        correlation_threshhold = 0.1

        corr = pd.DataFrame(Dataframe.corr()['critical_temp'])
        corr['abs'] = np.abs(corr['critical_temp'])
        corr = corr.sort_values(by='abs',ascending=False).drop('abs',axis=1).dropna().reset_index()
        corr = corr.rename(columns={'index':'feature','critical_temp':'corr'}).loc[1:]

        low_correlated_features = list(corr[np.abs(corr['corr'])<=correlation_threshhold]['feature'])
        Dataframe.drop(low_correlated_features,axis=1,inplace=True)
        print(str(original_columns-Dataframe.shape[1])+' features were found to be irrelevant')
        Dataframe.to_csv('data_filtered.csv')
        Dataframe.shape

        132 features were found to be irrelevant

Out[5]: (21263, 36)
```

Figure 29: data shape after removing correlated features with correlation target less than 0.1.

- Dropping features with a very small correlation with the target variable (less than |+-0.1|)

Conclusion

Feature selection plays a vital role in the performance and training of any machine learning model. Different types of methods have been proposed for feature selection for machine learning algorithms. We found that there are 132 features were found to be irrelevant out of the original 168 had very little relevance., and only 36 feature is important to our model.

3.5. Modeling: What Kind of model should we use?

3.5.1. Selecting the right model

From the problem definition we can determine the kind model we want to build now we move to the more difficult challenge of predicting Tc. We want a model that can predict a numerical value. Creating a regression model may enable better understanding of the factors controlling Tc of known superconductors, while also serving as an organic part of a system for identifying potential new ones. several regression models are presented focusing on Tc of materials.

Once we have a list of relevant predictors, various ML models can be applied to the data.[120,121]

At the beginning we looked up Machine Learning Map from Scikit-learn

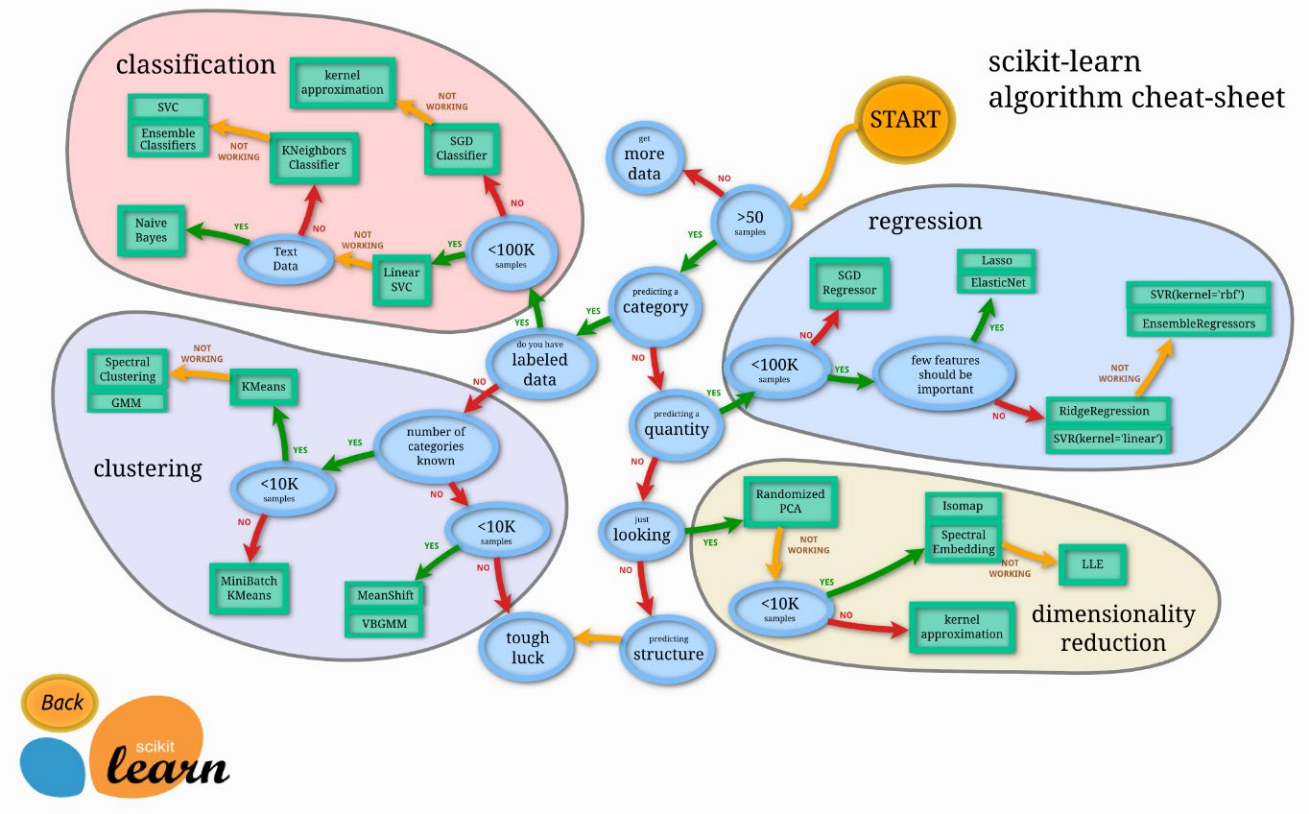


Figure 30: SK-learn algorithm cheat-sheet.

We found it quite useful as it serves as road map from any machine learning project.

Between it and several other machine learning Researches we had look up with nearly the same goal in mind [122-124] and Several books [120,121]we found out that All ML algorithms in this work are variants of the random forest method.[125]

Even though we decided to test Seven regression algorithm and do a model comparison between them.

3.5.2. Model Comparison

Seven different types of regression were performed on the data, with the intention of simply estimating the performance of each one without tuning hyperparameters. The metrics of evaluation for each one was the R2 score and the root mean squared error (RMSE). I also measured the time required for each model to fit and predict the data. After all the models and metrics were observed, the best performing one will then be selected for hyperparameter tuning.

- Ordinary Least Squares
- Ridge Regression
- Lasso Regression
- Elastic Net regression
- Bayesian Ridge
- K-nearest neighbors' regression
- Random Forest Regression

In [7]:
```
import time
from sklearn.metrics import r2_score,mean_squared_error
from sklearn.linear_model import Lasso,Ridge,ElasticNet, BayesianRidge, LinearRegression
from sklearn import neighbors
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
import warnings
warnings.filterwarnings('ignore')

#Models will test

models = {'OLS':LinearRegression(),'ElasticNet':ElasticNet(),
          'BayesianRidge':BayesianRidge(),'Lasso':Lasso(),
         'Ridge':Ridge(),'KNN':neighbors.KNeighborsRegressor(),
         'rff':RandomForestRegressor()}

def model_performance(X,y,i):
    times =[]
    keys = []
    mean_squared_errors = []
    R2_scores = []
    # Features Columns
    features = X.columns
    # Applying Function to Target data (Critical Temp.)
    y_t = y
    # Splits Data to test
    X_train, X_test, y_train, y_test = train_test_split(X, y_t, test_size=0.3, random_state=i)
    # Applying Function to Target Test data (Critical Temp.)
    y_test0 = y_test

    for k,v in models.items():
        model = v
        t0=time.time()
```

```
        model.fit(X_train, y_train)
        train_time = time.time()-t0
        t1 = time.time()

        pred = model.predict(X_test)
        predict_time = time.time()-t1
        pred = pd.Series(pred)

        Time_total = train_time+predict_time
        times.append(Time_total)
        # Test Model Accu
        R2_scores.append(r2_score(y_test0,pred))
        mean_squared_errors.append(mean_squared_error(y_test0,pred))
        keys.append(k)

    table = pd.DataFrame({'model':keys, 'RMSE':mean_squared_errors,'R2 score':R2_scores,'time':times})
    table['features'] = pd.Series([len(features) for i in range(len(R2_scores))])
    table['RMSE'] = table['RMSE'].apply(lambda x: np.sqrt(x))
    return table
```

In [8]:
```
model_performance(X,y,i)
```

Out[8]:

	model	RMSE	R2 score	time	features
0	OLS	19.815985	0.662988	0.105659	36
1	ElasticNet	20.389810	0.643187	0.031242	36
2	BayesianRidge	19.826111	0.662643	0.031255	36
3	Lasso	20.248298	0.648123	0.031240	36
4	Ridge	19.816210	0.662980	0.015621	36
5	KNN	16.577389	0.764144	0.312456	36
6	rff	9.574936	0.921316	19.061962	36

Figure 31: comparison between different algorithms used in the model.

The process of evaluating the models was done by a for loop. At each loop, a sequentially increasing number of features from the data was used on all the models. The order of the features fed into the models was based upon the magnitude of correlation each feature had with the critical_temperature. Hence the first feature used was range_ThermalConductivity which had a correlation of 0.687654, and the last feature was mean_fie which had a correlation of 0.102268. The loop passed through all 36 features. The first iteration had the first feature, the second had the first two features, the third had the first three, and so on. *Each model was initially fit without tuning hyperparameters*.

At each iteration the seven models were fit on a 70-30 train test split. The R2 and RMSE were then measured.

In [9]:
```
tables = pd.DataFrame()
for index in corr.index:
    features = list(corr['feature'].loc[:index])
    _ = df[features]
    tables = pd.concat([tables,model_performance(_,y,i=index)],axis=0)
tables.to_csv('seven_regressor_results_fitted.csv')
tables.sort_values(by='RMSE',ascending=True)
```

Out[9]:

	model	RMSE	R2 score	time	features
6	rff	9.264790	0.927283	13.108001	21
6	rff	9.277698	0.926946	16.179189	29
6	rff	9.297075	0.927010	15.850330	30
6	rff	9.407632	0.925269	19.913276	35
6	rff	9.499391	0.922856	13.499238	24
...	...	...	...	...	...
1	ElasticNet	24.660717	0.473031	0.004012	1
2	BayesianRidge	24.660729	0.473031	0.004020	1
4	Ridge	24.660740	0.473030	0.009976	1
0	OLS	24.660740	0.473030	0.004496	1
5	KNN	24.789200	0.473606	0.124970	2

252 rows × 5 columns

Then we saved all data as csv file called Seven_regressor_results_fitted.

3.5.3. Evaluation of models Performance

Then we checked for the result of the model with min RMSE

In [10]:
```
import pandas as pd
results = pd.read_csv('seven_regressor_results_fitted.csv').drop('Unnamed: 0',axis=1)
results['model'] = results['model'].apply(lambda x: 'RandomForestRegressor' if x=='rff' else x)
results[results['RMSE']==min(results['RMSE'])]
```

Out[10]:

	model	RMSE	R2 score	time	features
146	RandomForestRegressor	9.26479	0.927283	13.108001	21

Figure 32: the best fit model.

The Model with the Min RMSE was as predicted is Random Forest Regressor Algorithm

```
In [11]: from functools import reduce
ols = results[results['model']=='OLS']
ols.columns = [i+'_OLS' for i in ols.columns if i!='features']+['features']
EN = results[results['model']=='ElasticNet']
EN.columns = [i+'_ElasticNet' for i in EN.columns if i!='features']+['features']
BR = results[results['model']=='BayesianRidge']
BR.columns = [i+'_BayesianRidge' for i in BR.columns if i!='features']+['features']
lasso = results[results['model']=='Lasso']
lasso.columns = [i+'_Lasso' for i in lasso.columns if i!='features']+['features']
ridge = results[results['model']=='Ridge']
ridge.columns = [i+'_ridge' for i in ridge.columns if i!='features']+['features']
KNN = results[results['model']=='KNN']
KNN.columns = [i+'_KNN' for i in KNN.columns if i!='features']+['features']
rf = results[results['model']=='RandomForestRegressor']
rf.columns = [i+'_randomforest' for i in rf.columns if i!='features']+['features']

tables = [ols,EN,BR,lasso,ridge,KNN,rf]
df_merged = reduce(lambda  left,right: pd.merge(left,right,on=['features'],
                                                how='outer'), tables)
df_merged.drop([i for i in df_merged.columns if 'model_' in i],axis=1,inplace=True)

R2 = [i for i in df_merged.columns if 'R2 score' in i]
RMSE = [i for i in df_merged.columns if 'RMSE_' in i]
time = [i for i in df_merged.columns if 'time_' in i]
df_merged = df_merged[['features']+R2+RMSE+time]
df_merged.head()
```

Them we sorted each model with its own R2 Score and RMSE Score and Time and Features like in the next figure

Out[11]:

	features	R2 score_OLS	R2 score_ElasticNet	R2 score_BayesianRidge	R2 score_Lasso	R2 score_ridge	R2 score_KNN	R2 score_randomforest	RMSE_OLS	RMSE_ElasticNet	...
0	1	0.473030	0.473031	0.473031	0.473032	0.473030	0.558830	0.567656	24.660740	24.660717	...
1	2	0.500006	0.495768	0.500005	0.499163	0.500006	0.473606	0.582583	24.159590	24.261749	...
2	3	0.511529	0.510263	0.511548	0.510908	0.511530	0.676947	0.722966	24.042463	24.073623	...
3	4	0.530996	0.529062	0.531018	0.530416	0.530997	0.762697	0.840121	23.540262	23.588768	...
4	5	0.537598	0.533981	0.537559	0.535726	0.537597	0.755658	0.877952	23.279022	23.369879	...

5 rows × 22 columns

Figure 33: sorted models with its own R2 Score and RMSE and Time and Features

Graphing the evaluation metrics of each model as into three graphs discrping the relations between

- Number of features uesd VS R2 Score
- Number of features uesd VS RMSE

- Number of features uesd VS Model Evaluation time

In [12]:
```
import matplotlib.pyplot as plt
df_merged[['features']+[i for i in df_merged.columns if 'R2' in i]]

fig = plt.figure(figsize=(20,15))

plt.subplot(3,1,1)
for i in [i for i in df_merged.columns if 'R2' in i]:
    plt.plot(df_merged['features'],df_merged[i],label=i.split('_')[1])
plt.title('Number of features used vs R2 score',fontsize=20)
plt.legend(fontsize=15)
plt.xlabel('Number of features')
plt.ylabel('R2 score',fontsize=20)
plt.ylim(0,1)
plt.xlim(1,36)

plt.subplot(3,1,2)
for i in [i for i in df_merged.columns if 'RMSE' in i]:
    plt.plot(df_merged['features'],df_merged[i],label=i.split('_')[1])
plt.legend(fontsize=15)
plt.title('Number of features used vs RMSE',fontsize=20)
plt.xlabel('Number of features')
plt.ylabel('RMSE',fontsize=20)
plt.xlim(1,36)

plt.subplot(3,1,3)
for i in [i for i in df_merged.columns if 'time' in i]:
    plt.plot(df_merged['features'],df_merged[i],label=i.split('_')[1])
plt.legend(fontsize=15)
plt.xlabel('Number of features')
plt.ylabel('time (sec)',fontsize=20)
plt.title('Number of features used vs Model evaluation time',fontsize=20)
plt.xlim(1,36)

plt.subplots_adjust(hspace=0.4)
plt.savefig('Model_Comparison.png')
plt.show()
```

Every line the graphs shown describe performance of a model in term of their Plot as

- OLS: Linear Regression (Blue)
- ElasticNet: Elastic Net (Orange),
- BayesianRidge: Bayesian Ridge (Green),
- Lasso: Lasso (Red),
- Ridge: Ridge (Purple),

- KNN: K Neighbors Regressor (Brown),
- RF: Random Forest Regressor (Pink).

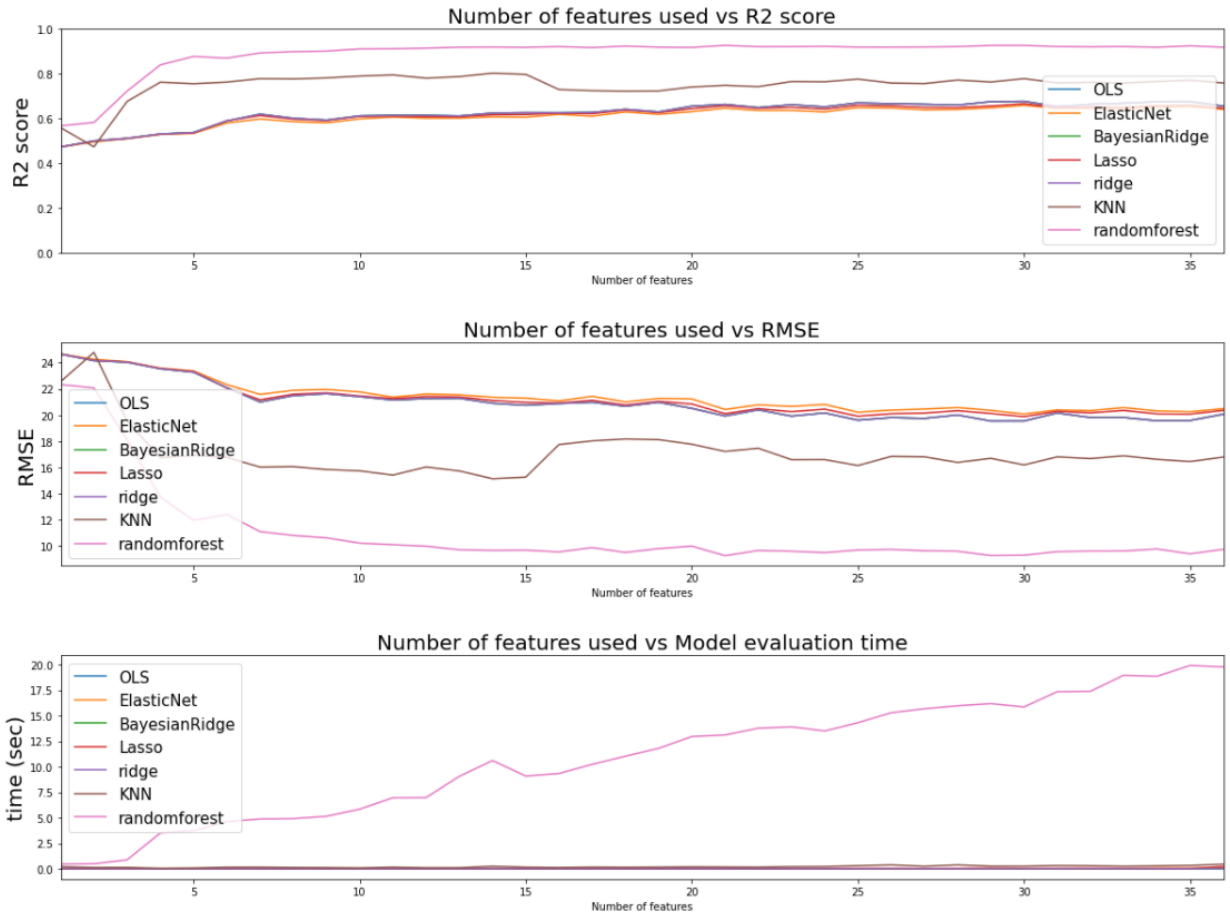


Figure 34: relation between number of features with R2 score and RMSE and model evaluation time.

We discovered that Random Forest Regressor have

- The highest R2 Score of all the model vs Number of Features
- The lowest RMSE Score of all model vs Number of Features
- The most evaluation of time of all model vs Number of Features

Then we ordered all model in ascending order in term of their performance

The best performance metrics by each model were without tuning hyperparameters;

In [13]:
```
best_results = pd.DataFrame()
for model in results['model'].unique():
    _  = results[(results['model']==model)].sort_values(by='RMSE',ascending=True).reset_index().loc[0]
    best_results = pd.concat([best_results,_],axis=1)
best_results = best_results.transpose().drop('index',axis=1)
best_results = best_results.sort_values(by='RMSE',ascending=True).reset_index().drop('index',axis=1)
best_results = best_results[['model','RMSE','R2 score', 'features','time']]
best_results = best_results.rename(columns={'features':'# of features used'})
for feature in ['RMSE','R2 score','time']:
    best_results[feature] = best_results[feature].apply(lambda x: round(x,4))
best_results
```

Out[13]:

	model	RMSE	R2 score	# of features used	time
0	RandomForestRegressor	9.2648	0.9273	21	13.1080
1	KNN	15.1408	0.8031	14	0.2474
2	BayesianRidge	19.5468	0.6774	30	0.0199
3	OLS	19.5474	0.6757	29	0.0180
4	Ridge	19.5475	0.6757	29	0.0080
5	Lasso	19.8837	0.6661	30	0.0419
6	ElasticNet	20.0870	0.6593	30	0.0359

Figure 35: ordered models from best to worst.

The Result was:

1. Random Forest Regression
2. K-nearest neighbors' regression
3. Bayesian Ridge
4. Ordinary Least Squares
5. Ridge Regression
6. Lasso Regression
7. Elastic Net regression

Thus, the best performing model (Random Forest regressor) was then fine-tuned with its hyperparameters

3.5.4. Random Forest Model

We tried running Random Forest Regressor without tunning its hyperparameters after Train_test_split, Splitting our data 75% training data and 25% Testing data.

```
In [14]: import random
         import matplotlib.pyplot as plt
         from sklearn.ensemble import RandomForestRegressor
         from sklearn.model_selection import train_test_split
         from sklearn.metrics import r2_score,mean_squared_error
         import pandas as pd
         import numpy as np

         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=40)


         rf = RandomForestRegressor()

         rf.fit(X_train, y_train)
         y_pred_test = pd.Series(rf.predict(X_test))
         y_pred_train = pd.Series(rf.predict(X_train))


         min_rmse = round(np.sqrt(mean_squared_error(y_test,y_pred_test)),4)
         r2 = r2_score(y_pred_test,y_test)
         print('rmse = ', min_rmse)
         print('R2 = ', r2)

         rmse =  9.4056
         R2 =  0.91800312404402
```

The Score R2 and RMSE of Random Forest Regressor without tunning its hyperparameters is:

- R2 = 0.918
- RMSE = 9.4056

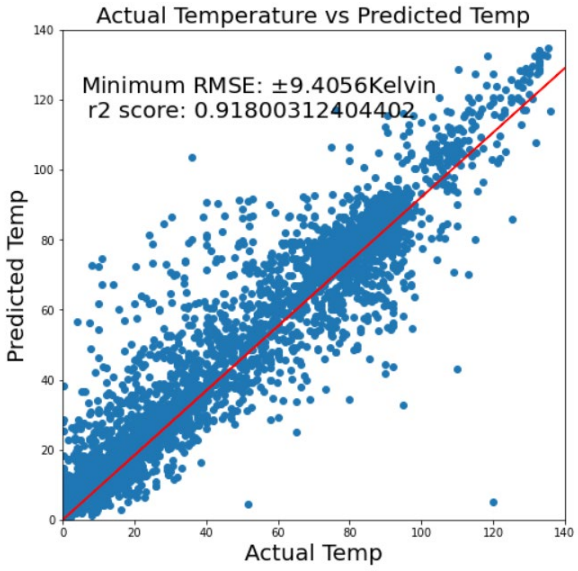


Figure 36: relation between actual temperature and predicted temperature for the model without tuning.

Also, it wasn't a very good fit Thus, the best performing model (Random Forest regressor) was then fine-tuned with its hyperparameters

Chapter Four

4 EXPERIMENT AND RESULTS.

4.1. Random Forest Model: Tuning hyperparameters

It is based on creating a set of individual decision trees (hence the "forest"), each built to solve the same regression problem. The model then combines their results, either by voting or averaging depending on the problem. The deeper individual tree is, the more complex the relationships the model can learn, but also the greater the danger of overfitting, i.e., learning some irrelevant information or just "noise". To make the forest more robust to overfitting, individual trees in the ensemble are built from samples drawn with replacement (a bootstrap sample) from the training set. In addition, when splitting a node during the construction of a tree, the model chooses the best split of the data only considering a random subset of the features. The random forest models above are developed using scikit-learn a powerful and efficient machine learning Python library.[126] Hyperparameters of these models include the number of trees in the forest, the maximum depth of each tree, the minimum number of samples required to split an internal node, and the number of features to consider when looking for the best split. To optimize the classifier and the combined/family-specific regressors.

Because the main goal is to get the most accurate conclusions about superconductors, accuracy, in this context, is more important than efficiency. Hence, despite random forest regressor being the slowest model to fit the data (over 1 second), it was chosen to be the most important because it had the lowest loss function.

We started with importing our data and separating the training data, the target column into X, Y.

In [15]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
import warnings
warnings.filterwarnings('ignore')
scaler = StandardScaler()

df = pd.read_csv('data_filtered.csv').drop('Unnamed: 0',axis=1)

X = df.drop('critical_temp',axis=1)
X = pd.DataFrame(scaler.fit_transform(X),columns=X.columns)
y = df['critical_temp']
```

In [31]:
```
import random
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score,mean_squared_error
import warnings
warnings.filterwarnings('ignore')
train_errors = []
test_errors = []
train_errors_log2 = []
test_errors_log2 = []
train_errors_sqrt = []
test_errors_sqrt = []



forest_size = [5,10,20,30,40,50,75,85,95,100,105,115,125,150,175,200,225,250,275,300]
for size in forest_size:
    rf = RandomForestRegressor(n_estimators=size)
    rf_sqrt = RandomForestRegressor(n_estimators=size,max_features='sqrt')
    rf_log2 = RandomForestRegressor(n_estimators=size,max_features='log2')

    y_t = y

    X_train, X_test, y_train, y_test = train_test_split(X, y_t, test_size=0.3, random_state=random.randint(1,101))
    rf.fit(X_train,y_train)
    rf_sqrt.fit(X_train,y_train)
    rf_log2.fit(X_train,y_train)

    y_pred_test = pd.Series(rf.predict(X_test))
    y_pred_train = pd.Series(rf.predict(X_train))

    y_pred_test_sqrt = pd.Series(rf_sqrt.predict(X_test))
    y_pred_train_sqrt = pd.Series(rf_sqrt.predict(X_train))

    y_pred_test_log2 = pd.Series(rf_log2.predict(X_test))
    y_pred_train_log2 = pd.Series(rf_log2.predict(X_train))
```

```
    test_error = np.sqrt(mean_squared_error(y_test,y_pred_test))
    test_errors.append(test_error)
    train_error = np.sqrt(mean_squared_error(y_train,y_pred_train))
    train_errors.append(train_error)

    test_error_sqrt = np.sqrt(mean_squared_error(y_test,y_pred_test_sqrt))
    train_error_sqrt = np.sqrt(mean_squared_error(y_train,y_pred_train_sqrt))
    test_errors_sqrt.append(test_error_sqrt)
    train_errors_sqrt.append(train_error_sqrt)

    test_error_log2 = np.sqrt(mean_squared_error(y_test,y_pred_test_log2))
    train_error_log2 = np.sqrt(mean_squared_error(y_train,y_pred_train_log2))
    test_errors_log2.append(test_error_log2)
    train_errors_log2.append(train_error_log2)
```

The hyperparameter that was tuned was the number of decision trees in the random forest. To do this, the GridSearch function in scikit-learn is employed, which generates and compares candidate models from a grid of parameter values. We used a trial-and-error approach where We cycled through a range of forest sizes. In conjunction, we also tried using different 'max feature' parameters, specifically log2, sqrt and the default. To reduce computational expense, models are not optimized at each step of the backward feature selection process.

In [32]:
```
zipped = list(zip(forest_size,test_errors))
zipped_sqrt = list(zip(forest_size,test_errors_sqrt))
zipped_log2 = list(zip(forest_size,test_errors_log2))
m = min([i[1] for i in zipped])
m_sqrt =min([i[1] for i in zipped_sqrt])
m_log2 = min([i[1] for i in zipped_log2])
min_error = [i for i in zipped if i[1]==m]
min_error_sqrt = [i for i in zipped_sqrt if i[1]==m_sqrt]
min_error_log2 = [i for i in zipped_log2 if i[1]==m_log2]
best_number_trees = min_error[0][0]
best_number_trees_sqrt = min_error_sqrt[0][0]
best_number_trees_log2 = min_error_log2[0][0]
print('Best Number trees: ',best_number_trees)
print('Best test RMSE: ',min_error[0][1])
print('best_number_trees_log2',best_number_trees_log2)
print('Best test RMSE log2: ',min_error_log2[0][1])
print('best_number_trees_sqrt: ',best_number_trees_sqrt)
print('best test RMSE sqrt: ',min_error_sqrt[0][1])
```

```
Best Number trees:  105
Best test RMSE:  9.390667279248321
best_number_trees_log2 105
Best test RMSE log2:  9.300329344921378
best_number_trees_sqrt:  105
best test RMSE sqrt:  9.304854483331358
```

Figure 37:the best results for the hyper parameters.

In [33]:
```
plt.figure(figsize=(20,5))

plt.title('''Random forest test and train errors for all features, log2 features and sqrt2 features''',fontsize=20)
plt.plot(forest_size,test_errors,label='test')
plt.plot(forest_size,train_errors,label='train')

plt.plot(forest_size,test_errors_sqrt,label='test_sqrt')
plt.plot(forest_size,train_errors_sqrt,label='train_sqrt')

plt.plot(forest_size,test_errors_log2,label='test_log2')
plt.plot(forest_size,train_errors_log2,label='train_log2')

plt.ylabel('RMSE',fontsize=20)
plt.xlabel('Number of decision trees in the random forest',fontsize=20)
plt.axvline(best_number_trees,linestyle='dashed')
plt.text(best_number_trees,7,'The best fit model \n had '+str(best_number_trees)+' estimators',fontsize=20)
plt.legend()
plt.savefig('RF_hyper_parameter_tuning.png')
plt.show()
```

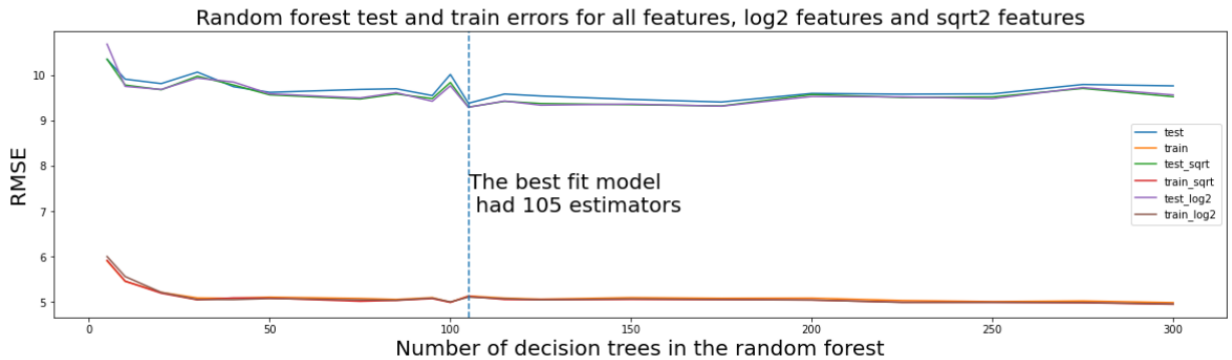


Figure 38: relation between errors and number of trees in the model.

After lots of trial and error, we estimated the best fit n_estimators in the random forest are 105 Tree.

We tested out model using the best estimated hyperparameters by using 75% of the data as training data and 25% of the data as testing data

In [34]:
```
import random
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score,mean_squared_error
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
import pickle


scaler = StandardScaler()


df = pd.read_csv('data_filtered.csv').drop('Unnamed: 0',axis=1)
X = df.drop('critical_temp',axis=1)
X = pd.DataFrame(scaler.fit_transform(X),columns=X.columns)
y = df['critical_temp']


rf = RandomForestRegressor(n_estimators=200,max_features='log2')

# Tranning Data is 75% and Testing Data is 25%

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=40)

rf.fit(X_train,y_train)
```

```
y_pred_test = pd.Series(rf.predict(X_test))
y_pred_train = pd.Series(rf.predict(X_train))
min_rmse = round(np.sqrt(mean_squared_error(y_test,y_pred_test)),4)
r2 = r2_score(y_pred_test,y_test)
print('rmse = ', min_rmse)
print('R2 = ', r2)


# save the model to disk
filename = 'finalized_model.sav'
pickle.dump(rf, open(filename, 'wb'))
```

```
rmse =  9.1522
R2 =  0.921773197063502
```

Figure 39: score of the tuned model and the errors.

After a lot of trial and error the outcome of the model we quit promising with the lowest RMSE score, we have accomplished.

4.2. Model Evaluation

4.2.1. Checking Model Accuracy

The Random Forrest regression after fine tuning hyperparameters After lots of trial and error, a RMSE of ±9.1522 Kelvin was found, R2 Score = 0.921773197063502 before tuning it was ±9.4056Kelvin, R2 = 0.918.

4.2.2. Checking for a good fit

As in any regression analysis, R2 score and RMSE are not the only standards of evaluation. Examining the plot of the predicted values versus actual values is important, as is looking for normality and homoskedascity.

The plot of the predicted temperatures versus actual temperature was:

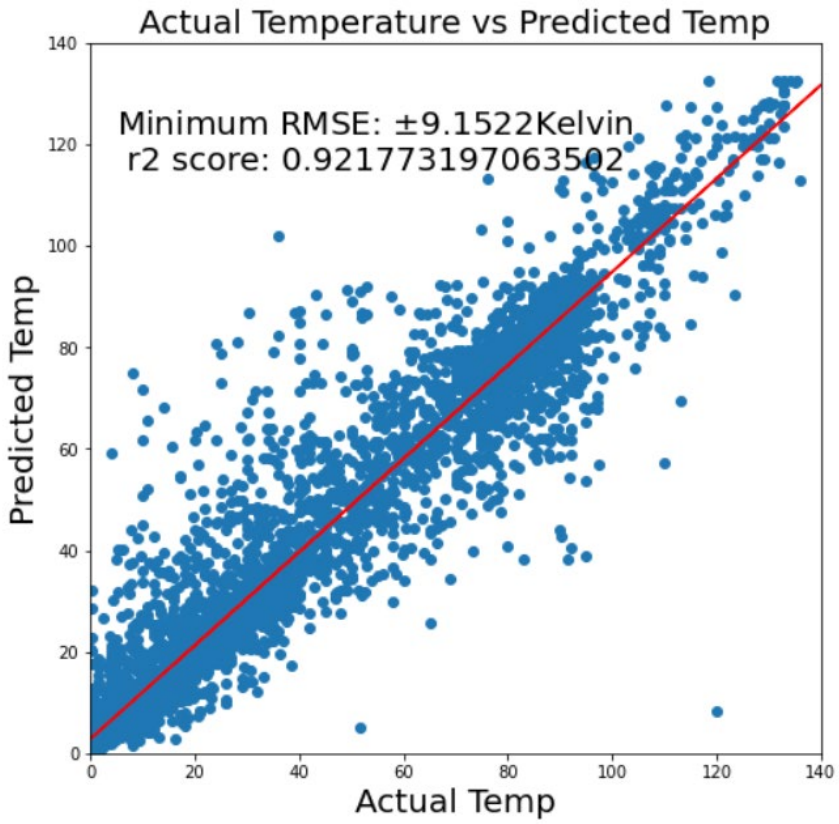


Figure 40: relation between actual temperature and predicted temperature for the tuned model.

4.2.3. Normality of errors & Homoskedascity

Using seaborn's distplot and probplot functions, I could plot the distribution of errors as well as their Q-Q plots (for both the train sets and the test sets). Approximately normality was observed.

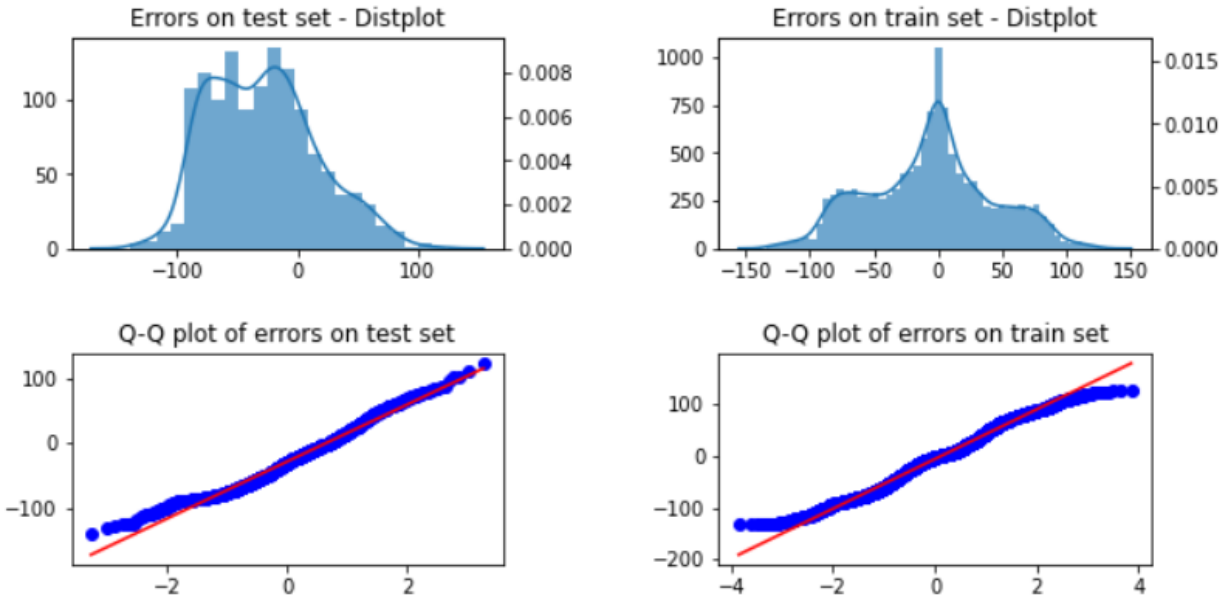


Figure 41: distribution of errors as well as their Q-Q plots (for both the train sets and the test sets)

4.3. Normalize Target

We attempted to improve our regression problem; we want the target variable to be as close to a normal distribution[127] as possible. Sometimes this is possible to do, other times not. In this case we were able to transform the critical temperature to be approximately normal.

Originally the data was very, very positively skewed. After a lot of trial and error, we used an exponential transformation to get the data (approximately) normal.

Unfortunately, in the best fit there remained a slight skew. The highest peak wasn't simply an outlier and couldn't be dropped or imputed. Despite this, the results observed at the end were pretty good

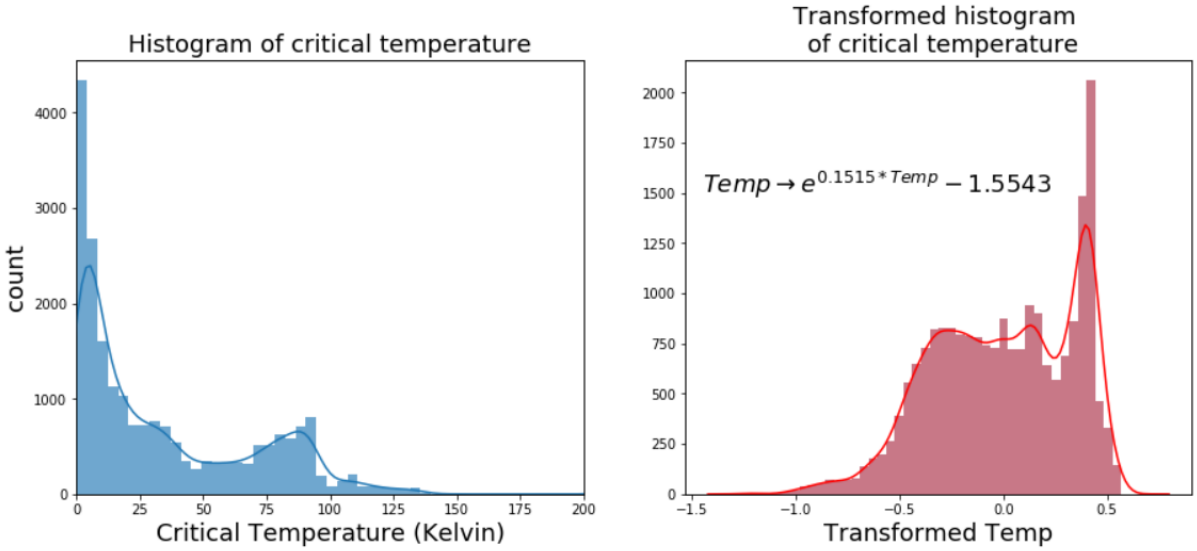


Figure 42: histograms for critical temperature before and after normalization.

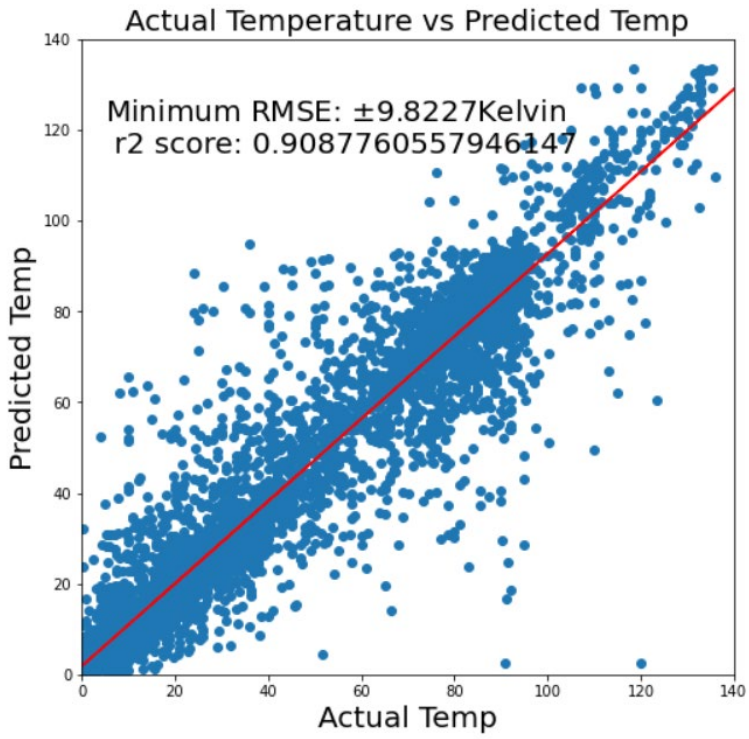
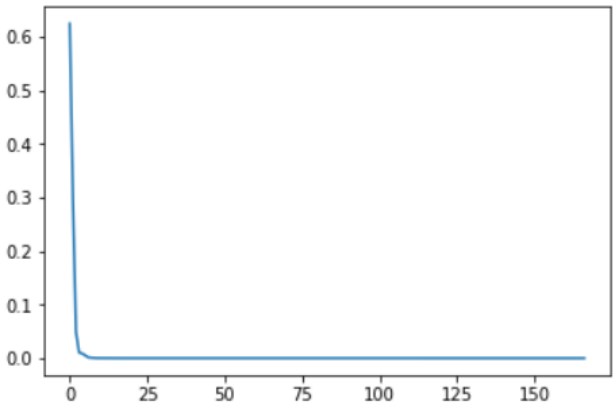


Figure 43: relation between actual temperature and predicted temperature for normalized data model.

Unfortunately, the result of our model was less than the original result, so we abandoned the Normalization approach.

4.4. PCA & EDC

Due to the average absolute value of the correlation among the features is 0.35. This indicates that the features are highly correlated. Motivated by this result, we attempted to reduce the dimensionality of the data using principal component analysis (PCA).



Out[7]:

	0	1	2	3
number_of_elements	-0.000128	0.000097	0.000012	0.000008
mean_atomic_mass	0.002584	0.001786	-0.003668	0.005150
wtd_mean_atomic_mass	0.003847	0.000806	0.001116	-0.005058
gmean_atomic_mass	0.003218	0.000472	-0.004055	0.005165
wtd_gmean_atomic_mass	0.004507	-0.000370	0.000707	-0.004934
...	...	...	...	...
Pb	0.000002	0.000002	-0.000003	-0.000003
Bi	-0.000011	0.000008	-0.000006	-0.000074
Po	-0.000000	0.000000	-0.000000	-0.000000
At	-0.000000	0.000000	-0.000000	-0.000000
Rn	-0.000000	0.000000	-0.000000	-0.000000

167 rows × 4 columns

Figure 44:PCA score.

However, our PCA analysis did not show any benefits in reducing the dimensionality since a large number of principal components were needed to capture a substantial percentage of the data variation; we abandoned the PCA approach.

Chapter Five

5 CONCLUSION/DISCUSSION/FUTURE WORK.

5.1. Feature Importance

One of the more convenient things about the random forest package is it does feature importance analysis automatically. Here, we can see the relative importance of each measure related to the increase of the *Tc*.

```
Feature ranking:
1. Cu (0.130620)
2. range_ThermalConductivity (0.089820)
3. O (0.088649)
4. range_fie (0.073867)
5. Ba (0.059423)
6. wtd_mean_ThermalConductivity (0.056350)
7. wtd_range_Valence (0.049207)
8. number_of_elements (0.047997)
9. Ca (0.034026)
10. mean_ThermalConductivity (0.030237)
11. wtd_entropy_fie (0.027584)
12. gmean_ThermalConductivity (0.026659)
13. wtd_range_fie (0.024842)
14. mean_FusionHeat (0.022408)
15. range_atomic_mass (0.021051)
16. range_Density (0.020379)
17. wtd_mean_ElectronAffinity (0.020109)
18. wtd_entropy_ThermalConductivity (0.019414)
19. wtd_mean_fie (0.019276)
20. range_ElectronAffinity (0.018199)
21. wtd_range_ElectronAffinity (0.016724)
22. wtd_range_atomic_mass (0.013419)
23. mean_Density (0.011996)
24. wtd_range_atomic_radius (0.011819)
25. Sr (0.010716)
26. mean_atomic_mass (0.010651)
27. mean_atomic_radius (0.009496)
28. mean_ElectronAffinity (0.009328)
29. mean_fie (0.008402)
30. range_Valence (0.005584)
31. range_FusionHeat (0.004461)
32. Y (0.003470)
33. Bi (0.002556)
34. Tl (0.001017)
35. S (0.000240)
```

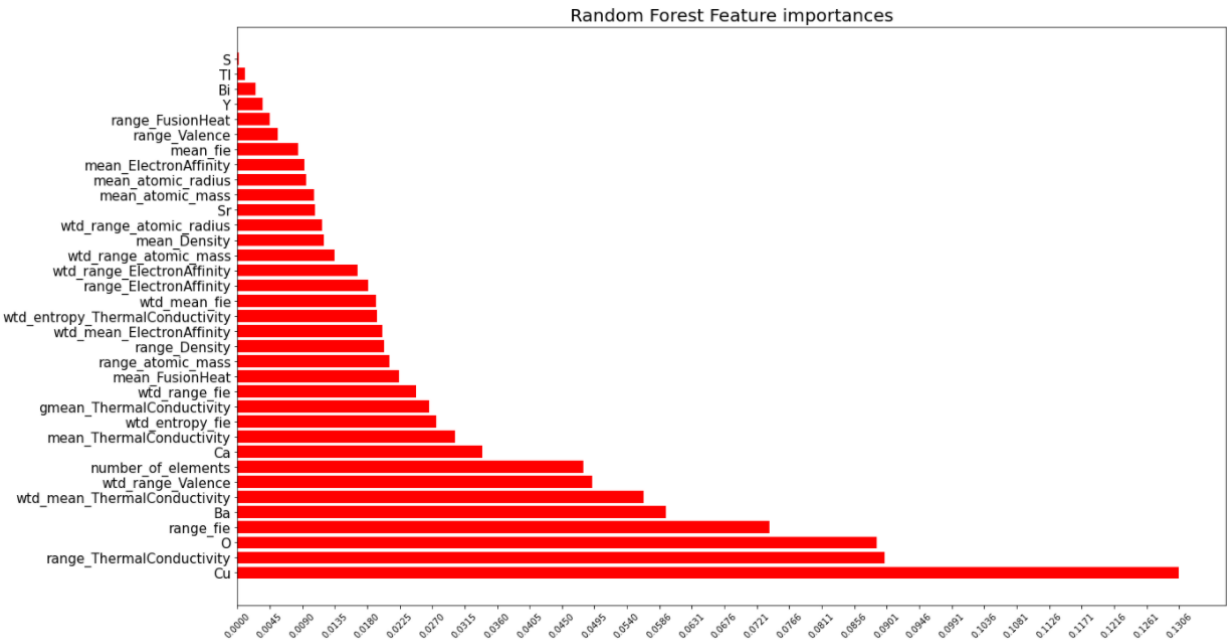


Figure 45: feature importance and its effect on the critical temperature.

5.2. Summery

- Superconductor is a zero-resistance material with unique properties that depends highly on its critical temperature.
- The most important properties of superconductor are conductivity of current with no energy loss and its perfect para magnetism.
- The benefit of finding superconductor with high Tc that it could solve the world energy problem by reducing the energy lost during transferring and lead to new innovative inventions.
- Machine learning could help us in predicting new superconductors with high Tc due.
- Due to the availability of numerous databases like supercon, material science and many others, machine learning (statistical approaches) could be useful to guide a breakthrough material science
- We developed a framework specifically for machine learning, it consists of number of steps:
  1. Problem definition
  2. Evaluation: to overcome the previews results of predeceasing researches.
  3. Data: preparation and analysis.
  4. Features: filtering irreverent data.
  5. Modeling: multiple regression models were compared and in the final model we found best model is random forest regression.

6. Experiment:
    - tried various technique for filtering and data like PCA did not show any benefits in reducing the dimensionality
    - we tried to normalize the data results weren't promising
    - We finetuned the hyperparameters of the random forest model resulted in RMSE of ±9.1522 Kelvin, R2 Score = 0.92177319706
- Due to the use of machine, we are one step closer to understand the relation between unique properties of zero DC resistivity and Tc and what features affects it.
- It will be cost and time saving method for determining the Tc of superconducting materials.

5.3. Future Work

- We aim to increase our sample of data by importing new features from different databases like AFLOW and material science.
- We aim to make a classification model and use only high Tc superconductors.
- Combine the classification and regression model in one pipeline.
- Using machine learning technique like deep learning to predict more superconducting materials with high Tc.

Reference

1. Hirsch, J. E., Maple, M. B. & Marsiglio, F. Superconducting materials: conventional, unconventional and undetermined. Phys. C. 514, 1–444 (2015).
2. Anderson, P. W. Plasmons, gauge invariance, and mass. Phys. Rev. 130, 439–442 (1963).
3. Chu, C. W., Deng, L. Z. & Lv, B. Hole-doped cuprate high temperature superconductors. Phys. C. 514, 290–313 (2015).
4. Paglione, J. & Greene, R. L. High-temperature superconductivity in iron-based materials. Nat. Phys. 6, 645–658 (2010).
5. Hassenzahl, W. V. (2000). Applications of superconductivity to electric power systems. IEEE Power Engineering Review, 20(5), 4-7.
6. F. Arute, et al., Nature 574, 505 (2019).
7. Matthias, B. T. (1955). Empirical relation between superconductivity and the number of electrons per atom. Phys. Rev., 97, 74{76.
8. S. Uchida, High Temperature Superconductivity: The Road to Higher Critical Temperature, Springer Japan (2015).
9. O. Isayev, D. Fourches, E.N. Muratov, C. Oses, K. Rasch, A. Tropsha and S. Curtarolo, Chemistry of Materials 27,735 (2015).
10. J. Hill, G. Mulholland, K. Persson, R. Seshadri, C.Wolverton and B. Meredig, MRS Bulletin, 41, 399 (2016).
11. J. Yuan, V. Stanev, C. Gao, I. Takeuchi and K. Jin, Superconductor Science and Technology, 32, 123001 (2019).
12. Villars, P. & Phillips, J. C. Quantum structural diagrams and high-Tc superconductivity. Phys. Rev. B 37, 2345–2348 (1988).
13. Rabe, K. M., Phillips, J. C., Villars, P. & Brown, I. D. Global multinary structural chemistry of stable quasicrystals, high-TC ferroelectrics, and high-Tc superconductors. Phys. Rev. B 45, 7650–7676 (1992).
14. Bergerhoff, G., Hundt, R., Sievers, R. & Brown, I. D. The inorganic crystal structure data base. J. Chem. Inf. Comput. Sci. 23, 66–69 (1983).
15. Curtarolo, S. et al. AFLOW: an automatic framework for high-throughput materials discovery. Comput. Mater. Sci. 58, 218–226 (2012).
16. Landis, D. D. et al. The computational materials repository. Comput. Sci. Eng. 14,

51–57 (2012).

17. Saal, J. E., Kirklin, S., Aykol, M., Meredig, B. & Wolverton, C. Materials design and discovery with high-throughput density functional theory: the Open Quantum Materials Database (OQMD). JOM 65, 1501–1509 (2013).
18. Jain, A. et al. Commentary: the Materials Project: a materials genome approach to accelerating materials innovation. APL Mater. 1, 011002 (2013).
19. Seko, A., Maekawa, T., Tsuda, K. & Tanaka, I. Machine learning with systematic density-functional theory calculations: application to melting temperatures of single- and binary-component solids. Phys. Rev. B 89, 054303–054313 (2014).
20. Balachandran, P. V., Theiler, J., Rondinelli, J. M. & Lookman, T. Materials prediction via classification learning. Sci. Rep. 5, 13285–13301 (2015).
21. Pilania, G. et al. Machine learning bandgaps of double perovskites. Sci. Rep. 6, 19375 (2016).
22. Isayev, O. et al. Universal fragment descriptors for predicting electronic properties of inorganic crystals. Nat. Commun. 8, 15679 (2017).
23. V. Stanev, C. Oses, A.G. Kusne, E. Rodriguez, J. Paglione, S. Curtarolo and I. Takeuchi, NPJ Computational Materials, 4, 29 (2018).
24. K. Hamidieh, Computational Materials Science, 154, 346 (2018).
25. S. Zeng, Y. Zhao, G Li, R. Wang, X. Wang and J. Ni, NPJ Computational Materials, 5, 84 (2019).
26. Q. Zhou, P. Tang, S. Liu, J. Pan, Q. Yan and S.-C. Zhang, Proceedings of the National Academy of Sciences, 115, E6411 (2018).
27. Valentin, S., Oses, C., Kusne, A. G., Rodriguez, E., Paglione, J., Curtarolo, S., and Takeuchi, I. (2017). Machine learning modeling of superconducting critical temperature. https://arxiv.org/abs/1709.02727.
28. National Institute of Materials Science, Materials Information Station, SuperCon, http://supercon.nims.go.jp/index_en.html (2011).
29. Kopnin, N. B., Heikkilä, T. T. & Volovik, G. E. High-temperature surface superconductivity in topological flat-band systems. Phys. Rev. B 83, 220503 (2011).
30. Peotta, S. & Törmä, P. Superfluidity in topologically nontrivial flat bands. Nat.Commun. 6, 8944 (2015).
31. Superconductors.org

32. Hirsch, J. E., Maple, M. B. & Marsiglio, F. Superconducting materials: conventional,unconventional and undetermined. Phys. C. 514, 1–444 (2015).
33. *Introduction to Superconductivity,* A.C. Rose-Innes and F.H. Rhoderidick, Pergamon, 1978.
34. Meissner and Ochsenfeld revisited December 2000European Journal of Physics
35. *The physics of superconductors: introduction to fundamentals and applications*, V. V. Schmidt, P. Muller, and A. V. Ustinov, New York: Springer, (1997).
36. *IEEE Transactions on Applied Superconductivity*, a quartly publication beginning March 1991.
37. Machine Learning Theory - Carnegie Mellon University
38. Alpaydin, E. Introduction to Machine Learning (The MIT Press, Cambridge, MA,2014).
39. Sutton, R. S. & Barto, A. G. Reinforcement Learning (The MIT Press, Cambridge, MA, 2018).
40. Marsland, S. Machine Learning (CRC Press, Taylor & Francis Inc., Boca Raton, FL, 2014).
41. Silver, D. et al. Mastering the game of go with deep neural networks and tree search. Nature 529, 484–489 (2016).
42. . Bojarski, M. et al. End to end learning for self-driving cars. Preprint at arXiv:1604.07316 (2016).
43. He, K., Zhang, X., Ren, S. & Sun, J. Delving deep into rectifiers: surpassing humanlevel
44. performance on ImageNet classification. In 2015 IEEE International Conference
45. on Computer Vision (ICCV) (eds Bajcsy, R. & Hager, G.) 1026–1034 (IEEE, Piscataway, NJ, 2015).
46. Liu, S.-S. & Tian, Y.-T. Facial expression recognition method based on gabor wavelet features and fractional power polynomial kernel PCA. In Advances in
47. Neural Networks - ISNN 2010 (eds Zhang, L., Lu, B.-L. & Kwok, J.) 144–151 (Springer, Berlin, Heidelberg, 2010).
48. Waibel, A. & Lee, K.-F. (eds) Readings in Speech Recognition (Morgan Kaufmann, Burlington, MA, 1990).
49. Pazzani, M. & Billsus, D. Learning and revising user profiles: the identification of interesting web sites. Mach. Learn. 27, 313–331 (1997).
50. Chan, P. K. & Stolfo, S. J. Toward scalable learning with non-uniform class and cost distributions: a case study in credit card fraud detection. In KDD'98 Proc. Fourth International Conference on Knowledge Discovery and Data Mining (eds Agrawal, R., Stolorz, P. & Piatetsky, G.) 164–168 (AAAI Press, New York, NY, 1998).
51. Guzella, T. S. & Caminhas, W. M. A review of machine learning approaches to spam filtering. Expert Syst. Appl. 36, 10206–10222 (2009).
52. Huang, C.-L., Chen, M.-C. & Wang, C.-J. Credit scoring with a data mining

approach based on support vector machines. Expert Syst. Appl. 33, 847–856 (2007).

53. Baldi, P. & Brunak, S. Bioinformatics: The Machine Learning Approach (The MIT Press, Cambridge, MA, 2001).
54. Noordik, J. H. Cheminformatics Developments: History, Reviews and Current Research (IOS Press, Amsterdam, 2004).
55. Rajan, K. Materials informatics. Mater. Today 8, 38–45 (2005).
56. Martin, R. M. Electronic Structure: Basic Theory and Practical Methods (Cambridge University Press, Cambridge, 2008).
57. Hohenberg, P. & Kohn, W. Inhomogeneous electron gas. Phys. Rev. 136, B864–B871 (1964).
58. Kohn, W. & Sham, L. J. Self-consistent equations including exchange and correlation effects. Phys. Rev. 140, A1133–A1138 (1965).
59. Olson, G. B. Designing a new material world. Science 288, 993–998 (2000).
60. Oganov, A. R. (ed.) Modern Methods of Crystal Structure Prediction (Wiley-VCH Verlag GmbH & Co. KGaA, Weinheim, 2010).
61. Oganov, A. R. & Glass, C. W. Crystal structure prediction using ab initio evolutionary techniques: principles and applications. J. Chem. Phys. 124, 244704 (2006).
62. Newnham, R. E. Properties of materials: anisotropy, symmetry, structure (Oxford University Press, Oxford, 2005).
63. Curtarolo, S. et al. The high-throughput highway to computational materials design. Nat. Mater. 12, 191–201 (2013).
64. Green, M. L. et al. Fulfilling the promise of the materials genome initiative with high-throughput experimental methodologies. Appl. Phys. Rev. 4, 011105 (2017).
65. Koinuma, H. & Takeuchi, I. Combinatorial solid-state chemistry of inorganic materials. Nat. Mater. 3, 429–438 (2004).
66. Suram, S. K., Haber, J. A., Jin, J. & Gregoire, J. M. Generating information-rich high-throughput experimental materials genomes using functional clustering via multitree genetic programming and information theory. ACS Comb. Sci. 17, 224–233 (2015).
67. Potyrailo, R. et al. Combinatorial and high-throughput screening of materials libraries: review of state of the art. ACS Comb. Sci. 13, 579–633 (2011).
68. Walsh, A. The quest for new functionality. Nat. Chem. 7, 274–275 (2015).
69. Lookman, T., Eidenbenz, S., Alexander, F. & Barnes, C. (eds) Materials Discovery and Design by Means of Data Science and Optimal Learning (Springer International

Publishing, Basel, 2018).

70. Ryan, K., Lengyel, J. & Shatruk, M. Crystal structure prediction via deep learning. J. Am. Chem. Soc. 140, 10158–10168 (2018).
71. Nouira, A., Sokolovska, N. & Crivello, J.-C. Crystalgan: learning to discover crystallographic structures with generative adversarial networks. Preprint at arXiv:1810.11203 (2018).
72. Graser, J., Kauwe, S. K. & Sparks, T. D. Machine learning and energy minimization approaches for crystal structure predictions: a review and new horizons. Chem. Mater. 30, 3601–3612 (2018).
73. Balachandran, P. V., Kowalski, B., Sehirlioglu, A. & Lookman, T. Experimental search for high-temperature ferroelectric perovskites guided by two-step machine learning. Nat. Commun. 9, 1668 (2018).
74. Oliynyk, A. O., Adutwum, L. A., Harynuk, J. J. & Mar, A. Classifying crystal structures of binary compounds AB through cluster resolution feature selection and support vector machine analysis. Chem. Mater. 28, 6672–6681 (2016).
75. Li, W., Jacobs, R. & Morgan, D. Predicting the thermodynamic stability of perovskite oxides using machine learning models. Comput. Mater. Sci. 150, 454–463 (2018).
76. Ward, L. et al. Including crystal structure attributes in machine learning models of formation energies via Voronoi tessellations. Phys. Rev. B 96, 024104 (2017).
77. Faber, F. A., Lindmaa, A., von Lilienfeld, O. A. & Armiento, R. Machine learning energies of 2 million elpasolite (ABC2D6) crystals. Phys. Rev. Lett. 117, 135502 (2016).
78. Zheng, X., Zheng, P. & Zhang, R.-Z. Machine learning material properties from the periodic table using convolutional neural networks. Chem. Sci. 9, 8426–8432 (2018).
79. Carrete, J., Li, W., Mingo, N., Wang, S. & Curtarolo, S. Finding unprecedentedly low-thermal-conductivity half-Heusler semiconductors via high-throughput materials modeling. Phys. Rev. X 4, 011019 (2014).
80. Kim, C., Pilania, G. & Ramprasad, R. From organized high-throughput data to phenomenological theory using machine learning: the example of dielectric breakdown. Chem. Mater. 28, 1304–1311 (2016).
81. Seko, A., Maekawa, T., Tsuda, K. & Tanaka, I. Machine learning with systematic density-functional theory calculations: application to melting temperatures of single- and binary-component solids. Phys. Rev. B 89, 054303 (2014).
82. Xie, T. & Grossman, J. C. Crystal graph convolutional neural networks for an

accurate and interpretable prediction of material properties. Phys. Rev. Lett. 120, 145301 (2018).

83. Isayev, O. et al. Universal fragment descriptors for predicting properties of inorganic crystals. Nat. Commun. 8, 15679 (2017).
84. Furmanchuk, A., Agrawal, A. & Choudhary, A. Predictive analytics for crystalline materials: bulk modulus. RSC Adv. 6, 95246–95251 (2016).
85. Kauwe, S. K., Graser, J., Vazquez, A. & Sparks, T. D. Machine learning prediction of heat capacity for solid inorganics. Integr. Mater. Manuf. Innov. 7, 43–51 (2018).
86. Kim, C., Pilania, G. & Ramprasad, R. Machine learning assisted predictions of intrinsic dielectric breakdown strength of ABX3 perovskites. J. Phys. Chem. C 120, 14575–14580 (2016).
87. Yuan, F. & Mueller, T. Identifying models of dielectric breakdown strength from high-throughput data via genetic programming. Sci. Rep. 7, 17594 (2017).
88. Gaultois, M. W. et al. Perspective: Web-based machine learning models for realtime screening of thermoelectric materials properties. APL Mater. 4, 053213 (2016).
89. Nguyen, H., Maeda, S.-i. & Oono, K. Semi-supervised learning of hierarchical representations of molecules using neural message passing. Preprint at arXiv:1711.10168 (2017).
90. Zhang, Y. & Ling, C. A strategy to apply machine learning to small datasets in materials science. npj Comput. Mater. 4, 25 (2018).
91. Jain, A. et al. Commentary: The materials project: a materials genome approach to accelerating materials innovation. APL Mater. 1, 011002 (2013).
92. F. H. Allen, G. G. & Sievers, R. (eds) Crystallographic Databases (International Union of Crystallography, Chester, 1987).
93. Saal, J. E., Kirklin, S., Aykol, M., Meredig, B. & Wolverton, C. Materials design and discovery with high-throughput density functional theory: the open quantum materials database (OQMD). JOM 65, 1501–1509 (2013).
94. Kirklin, S. et al. The open quantum materials database (OQMD): assessing the accuracy of DFT formation energies. npj Comput. Mater. 1, 15010 (2015).
95. Groom, C. R., Bruno, I. J., Lightfoot, M. P. & Ward, S. C. The Cambridge structural database. Acta Crystallogr. Sect. B Struct. Sci. Cryst. Eng. Mater. 72, 171–179 (2016).
96. Hachmann, J. et al. The Harvard clean energy project: large-scale computational screening and design of organic photovoltaics on the world community grid. J. Phys. Chem. Lett. 2, 2241–2251 (2011).

97. Puchala, B. et al. The materials commons: a collaboration platform and information repository for the global materials community. JOM 68, 2035–2044 (2016).
98. Mullin, R. Citrine informatics. CEN Glob. Enterp. 95, 34–34 (2017).
99. de Jong, M. et al. Charting the complete elastic properties of inorganic crystalline compounds. Sci. Data 2, 150009 (2015).
100. Zakutayev, A. et al. An open experimental database for exploring inorganic materials. Sci. Data 5, 180053 (2018).
101. Villars, P., Okamoto, H. & Cenzual, K. ASM Alloy Phase Diagrams Database (ASM International, Materials Park, OH, 2006).
102. Gražulis, S. et al. Crystallography open database (COD): an open-access collection of crystal structures and platform for world-wide collaboration. Nucleic Acids Res. 40, D420–D427 (2011).
103. Villars, P. et al. The Pauling file, binaries edition. J. Alloy. Comp. 367, 293–297 (2004).
104. Gorai, P. et al. TE design lab: a virtual laboratory for thermoelectric material design. Comput. Mater. Sci. 112, 368–376 (2016).
105. Haastrup, S. et al. The Computational 2D Materials Database: high-throughput modeling and discovery of atomically thin crystals. 2D Mater. 5, 042002 (2018).
106. Wilkinson, M. D. et al. The FAIR guiding principles for scientific data management and stewardship. Sci. Data 3, 160018 (2016).
107.. Draxl, C. & Scheffler, M. NOMAD: the FAIR concept for big data-driven materials science. MRS Bull. 43, 676–682 (2018).
108. Raccuglia, P. et al. Machine-learning-assisted materials discovery using failed experiments. Nature 533, 73–76 (2016).

109.University of Alaska Fairbanks—Superconductivity

110. Systems Analysis Design UML Version 2.0 An Object-Oriented Approach Third Edition

111.Wolfram and Research (2017). Mathematica, version 11.2.

112. http://www.ptable.com/.

113. Conder, K. (2016). A second life of the matthiass rules. Superconductor Science and Technology,29(8)

114. http://supercon.nims.go.jp/supercon/material_ menu.

115. R Core Team (2017). R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria.

116.Dick, J. M. (2008). Calculation of the relative metastabilities of proteins using the chnosz software package. Geochemical Transactions, 9(10).

117.https://www.semanticscholar.org/paper/A-Data-Driven-Statistical-Model-for-Predicting-the-Hamidieh/b3bea0ac481f0869cb746f3b44d5689bf1a9b924/figure/2

118. https://stackabuse.com/applying-filter-methods-in-python-for-feature-selection/

119. https://stackabuse.com/applying-wrapper-methods-in-python-for-feature-selection/

120. Bishop, C. Pattern Recognition and Machine Learning. (Springer-Verlag, NY, 2006).

121. Hastie, T., Tibshirani, R. & Friedman, J. H. The Elements of Statistical Learning: Data

Mining, Inference, and Prediction. (Springer-Verlag, NY, 2001).

122.Machine learning modeling of superconducting critical temperature,

123. A Data-Driven Statistical Model for Predicting the Critical Temperature of a Superconductor.

124. Predicting new superconductors and their critical temperatures using unsupervised machine learning

125. Breiman, L. Random forests. Mach. Learn. 45, 5–32 (2001).

126. Pedregosa, F. et al. Scikit-learn: Machine Learning in Python. J. Mach. Learn. Res.12, 2825–2830 (2011).

127. https://github.com/mwaskom/seaborn/issues/479