REGRESSION OF SUPERCONDUCTING CRITICAL TEMPERATURE

A Project report
submitted in partial fulfillment of the requirements for the award of the
Degree of

BACHELOR OF TECHNOLOGY
In
COMPUTER SCIENCE AND ENGINEERING
By

D. Harsha Praneeth	15191A0518
P. Suresh	15191A0541
U. Hanumanthu	15191A0557

Under the guidance of
Sri A. Naresh M.Tech, (Ph.D)
Assist. Prof (Ad hoc)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY ANANTAPUR
COLLEGE OF ENGINEERING
(Autonomous)
PULIVENDULA – 516390
ANDHRA PRADESH – INDIA
2015 - 2019

REGRESSION OF SUPERCONDUCTING CRITICAL TEMPERATURE

A Project report
submitted in partial fulfillment of the requirements for the award of the
Degree of

BACHELOR OF TECHNOLOGY
In
COMPUTER SCIENCE AND ENGINEERING
By

D. Harsha Praneeth	15191A0518
P. Suresh	15191A0541
U. Hanumanthu	15191A0557

Under the guidance of
Sri A. Naresh **M.Tech, (Ph.D)**
Assist. Prof (Ad hoc)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY ANANTAPUR
COLLEGE OF ENGINEERING
(Autonomous)
PULIVENDULA – 516390
ANDHRA PRADESH – INDIA

2015 – 2019

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY ANANTAPUR
COLLEGE OF ENGINEERING
(Autonomous)
PULIVENDULA - 516 390

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERITIFICATE

This is to certify that the project entitled –**REGRESSION OF SUPERCONDUCTING CRITICAL TEMPERATURE**‖ is being submitted by

D. Harsha Praneeth	15191A0518
P. Suresh	15191A0541
U.Hanumanthu	15191A0557

In fulfillment for the award of the Degree of Bachelor of Technology in Computer science & engineering to the **Jawaharlal Nehru Technological University, Anantapur college of Engineering (Autonomous) Pulivendula**, is a record of bonafide work carried out under my guidance and supervision. The results embodied in this project report have not been submitted to any other University or Institute for the award of any Degree or Diploma.

Signature of the Project guide

Signature of the Head of the Department

Sri. G. Murali M.E. (Ph.D)
Assistant Professor & Head
of the Department of CSE
JNTUACE, Pulivendula

Sri A. Naresh M.Tech, (Ph.D)
Assist. Prof (Ad hoc)

Signature of the external examiner

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY ANANTAPUR
COLLEGE OF ENGINEERING (AUTONOMOUS) PULIVENDULA - 516390

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

STUDENT DECLARATION

We hereby declare that this submission is our own work and that to the best of our Knowledge and belief, it contains no material previously published or written by another person or material which has been accepted for the award of any degree or diploma of any university or institute of higher learning.

D. Harsha Praneeth
(15191A0518)

P. Suresh
(15191A0541)

U. Hanumanthu
(15191A0557)

Acknowledgement

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of people who made it possible, whose constant guidance and encouragement crowned our efforts with success. It is a pleasant aspect that I have now the opportunity to express my guidance for all of them.

I wish to thank Prof. **S. Srinivas Kumar** Honorable Vice Chancellor, JNTUA. His motivation words that created burning desire in us and that eventually led to successfully completing this project work.

The man who has helped us a lot in times of trouble, and who helped us in recovering from intricate problems is Prof **G. Shankara Sekhar Raju,** principal, JNTUA College of Engineering, Pulivendula. I am thankful to him.

It’s my pleasure to say thanks to Prof. **Prof. G.V. Subba Reddy**, Vice Principle, JNTUA College of Engineering, Pulivendula for motivating us to having vision and persistence to work in spite of many obstacles.

Foremost, we gratefully acknowledge my sincere gratitude to our respected **G. Murali,** Assistant Professor, Head of the Department, Department of Computer Science and Engineering, JNTUA College of Engineering, Pulivendula for her valuable guidance, timely suggestions, and lively intense throughout the work. We always thankful to her for providing me the opportunity to do work.

The real mentor and motivator of this project **A. Naresh,** Assistant Professor of Computer Science and Engineering Department, JNTUA College of Engineering, Pulivendula. His wide knowledge and logical way of thinking have made a deep impression on me. His understanding, encouragement and personal guidance have provided the basis for this thesis. His source of inspiration for innovative ideas and his kind support is well to all his students and colleagues.

Last but far from least, I also thank my family members, my friends, staff and faculty members of CSE Dept for their moral support and constant encouragement, I am very much thankful to one and all that helped me for the successful completion of the project.

with gratitude

D. Harsha Praneeth (15191A0518)

P. Suresh (15191A0541)

U. Hanumanthu (15191A0557)

Abstract

Superconductivity is being studied since its discovery more than a century ago. The numerous applications of the superconductors made it a subject of intense research. Despite being studied for so long, some of its properties remain a mystery. One of the interesting properties of a super conductor is its critical temperature. Superconductors exhibit zero electrical resistance when maintained at the critical temperature. The value of critical temperature is different for each superconducting material. This value is experimentally calculated by measuring resistance against the temperature of the material.

Superconductors are generally a composition of different elements. So, there are many potential materials which can be superconductors but not discovered yet. Researchers even today are looking for materials which can act as a superconductor at near room temperature. Understanding the relationship between superconductivity and materials' chemistry and structure presents significant theoretical and experimental challenges. But today we have enough data covering various properties of superconducting materials to use data-driven approaches like statistics and machine learning to predict properties of interest like critical temperature.

In this project, by taking advantage of the immense increase of readily accessible and potentially relevant information, we develop several machine learning methods modeling critical temperature of a super conductor based on its chemical properties. The final model will give an estimate of critical temperature of a superconductor based on its chemical properties. This estimate provides confidence on a newly discovered material to continue further research on it, if the provided estimate is in any way interesting.

List of contents

List of tables

List of Figures

Chapter 1

Introduction

Machine learning is the science of getting computers to act without being explicitly programmed. In the past decade, machine learning has given us self-driving cars, practical speech recognition, effective web search, and a vastly improved understanding of the human genome. Machine learning is so pervasive today that you probably use it dozens of times a day without knowing it. Many researchers also think it is the best way to make progress towards human-level AI.

Superconductivity is one of the mainstream research topics in physics, chemistry and materials science. Machine Learning allows a different approach to look at superconductivity. The sheer quantity of data available on superconductors these days make it possible to employ data driven approaches like statistics and Machine Learning to predict certain properties of superconductors.

1.1 Superconductivity

Superconductivity is a phenomenon of exactly zero electrical resistance and expulsion of magnetic flux fields occurring in certain materials, called superconductors, when cooled below a characteristic critical temperature. The electrical resistance of a metallic conductor decreases gradually as temperature is lowered. In ordinary conductors, such as copper or silver, this decrease is limited by impurities and other defects. Even near absolute zero, a real sample of a normal conductor shows some resistance. In a superconductor, the resistance drops abruptly to zero when the material is cooled below its critical temperature. An electric current through a loop of superconducting wire can persist indefinitely with no power source.

In 1911, when Kamerlingh Onnes was studying the behavior of metals at low temperatures he first observed this fascinating phenomenon superconductivity. Onnes and his colleagues, found that the resistance of mercury, when cooled below 4.2 K, dropped to practically zero. He also observed that very high currents can be passed through mercury in this superconducting state until a maximum current density was reached. At that point, the mercury would return to the normal state. After years of hard work Onnes became the first man who reached 4.2 K mark by liquefying Helium, which later led to the beginning of a new era of superconductivity. In 1913, Kamerlingh Onnes won Nobel Prize in Physics "for his investigations on the properties of matter at low temperatures which led, inter alia, to the production of liquid helium".

Superconductors are not just better than ordinary conductors of electricity, they are different by mechanism and order. From those days, it has been a fascinating phenomenon and an attracting subject to physicists, chemists, technologists, material scientists, experimentalists and theoreticians due to various reasons. All these groups of people saw a big future even if their perspectives were different. But soon everyone understood that even if the dreams were beautiful the path is not so easy. The fascination of physicists with superconductivity is from its exclusive properties that are intellectually challenging and potential for a wide range of applications. Even if it must address many more issues we can surely say that it is one of the discoveries which changed the progress of mankind.

1.2 Machine Learning

The Machine Learning field evolved from the broad field of Artificial Intelligence, which aims to mimic intelligent abilities of humans by machines. In the field of Machine learning one considers the important question of how to make machines able to ―learn‖. Learning in this context is understood as inductive inference, where one observes examples that represent incomplete information about some ―statistical phenomenon‖. In unsupervised learning one typically tries to uncover hidden regularities (e.g. clusters) or to detect anomalies in the data (for instance some unusual machine function or a network intrusion). In supervised learning, there is a label associated with each example. It is supposed to be the answer to a question about the example. If the label is discrete, then the task is called classification problem – otherwise, for real valued labels we speak of a regression problem. Based on these examples (including the labels), one is particularly interested to predict the answer for other cases before they are explicitly observed. Hence, learning is not only a question of remembering but also of generalization to unseen cases.

1.3 Regression

Regression analysis is a form of predictive modelling technique which investigates the relationship between a dependent (target) and independent variable(s) . This technique is used for forecasting, time series modelling and finding the causal effect relationship between the variables. For example, relationship between rash driving and number of road accidents by a driver is best studied through regression.

Regression analysis is an important tool for modelling and analyzing data. Here, we fit a curve / line to the data points, in such a manner that the differences between the distances of data points from the curve or line is minimized.

There are multiple benefits of using regression analysis. They are as follows:
It indicates the significant relationships between dependent and independent variable.
It indicates the strength of impact of multiple independent variables on a dependent one.
Regression analysis also allows us to compare the effects of variables measured on different scales, such as the effect of price changes and the number of promotional activities. These benefits help market researchers / data analysts / data scientists to eliminate and evaluate the best set of variables to be used for building predictive models.
Types of Regression

- Simple Linear Regression
- Polynomial Regression
- Support Vector Regression
- Decision Tree Regression
- Random Forest Regression

1.3.1 Simple Linear Regression

This is one of the most common and interesting type of Regression technique. Here we predict a target variable Y based on the input variable X. A linear relationship should exist between target variable and predictor and so comes the name Linear Regression. Consider predicting the salary of an employee based on his/her age. We can easily identify that there seems to be a correlation between employee's age and salary (more the age more is the salary). The hypothesis of linear regression is $Y = a + b X$.

Y represents salary, X is employee's age and a and b are the coefficients of equation. So, to predict Y (salary) given X (age), we need to know the values of a and b (the model's coefficients).

While training and building a regression model, it is these coefficients which are learned and fitted to training data. The aim of training is to find a best fit line such that cost function is minimized. The cost function helps in measuring the error. During training process, we try to minimize the error between actual and predicted values and thus minimizing cost function.

To summarize, our aim is to find such values of coefficients which will minimize the cost function. The most common cost function is Mean Squared Error (MSE) which is equal to average squared difference between an observation actual and predicted values. The coefficient values can be calculated using Gradient Descent approach which will be discussed in detail in later articles. To give a brief understanding, in Gradient descent we start with some random values of coefficients, compute gradient of cost function on these values, update the coefficients and calculate the cost function again. This process is repeated until we find a minimum value of cost function.

1.3.2 Polynomial Regression

In statistics, polynomial regression is a form of regression analysis in which the relationship between the independent variable x and the dependent variable y is modelled as an nth degree polynomial in x. Polynomial regression fits a nonlinear relationship between the value of x and the corresponding conditional mean of y, denoted E(y |x), and has been used to describe nonlinear phenomena such as the growth rate of tissues, the distribution of carbon isotopes in lake sediments, and the progression of disease epidemics. Although polynomial regression fits a nonlinear model to the data, as a statistical estimation problem it is linear, in the sense that the regression function E (y | x) is linear in the unknown parameters that are estimated from the data. For this reason, polynomial regression is considered to be a special case of multiple linear regression.

The explanatory (independent) variables resulting from the polynomial expansion of the "baseline" variables are known as higher-degree terms. Such variables are also used in classification settings.

Polynomial regression models are usually fit using the method of least squares. The least-squares method minimizes the variance of the unbiased estimators of the coefficients, under the conditions of the Gauss-Markov theorem. The least-squares method was published in 1805 by Legendre and in 1809 by Gauss. The first design of an experiment for polynomial regression appeared in an 1815 paper of Gergonne. In the twentieth century, polynomial regression played an important role in the development of regression analysis, with a greater emphasis on issues of design and inference. More recently, the use of polynomial models has been complemented by other methods, with non-polynomial models having advantages for some classes of problems.

The goal of regression analysis is to model the expected value of a dependent variable y in terms of the value of an independent variable (or vector of independent variables) x. In simple linear regression, the model is used, where ε is an unobserved random error with mean zero conditioned on a scalar variable x. In this model, for each unit increase in the value of x, the conditional expectation of y increases by β1 units.

In many settings, such a linear relationship may not hold. For example, if we are modeling the yield of a chemical synthesis in terms of the temperature at which the synthesis takes place, we may find that the yield improves by increasing amounts for each unit increase in temperature. In this case, we might propose a quadratic model of the form.

Conveniently, these models are all linear from the point of view of estimation, since the regression function is linear in terms of the unknown parameters β0, β1,......Therefore, for least squares analysis, the computational and inferential problems of polynomial regression can be completely addressed using the techniques of multiple regression. This is done by treating x, x2, ...as being distinct independent variables in a multiple regression model.

1.3.3 Support Vector Regression

In SVR, we identify a hyperplane with maximum margin such that maximum number of data points are within that margin. SVRs are almost like SVM classification algorithm. We will discuss SVM algorithm in detail in my next article. Support Vector Machine can also be used as a regression method, maintaining all the main features that characterize the algorithm (maximal margin). The Support Vector Regression (SVR) uses the same principles as the SVM for classification, with only a few minor differences. First of all, because output is a real number it becomes very difficult to predict the information at hand, which has infinite possibilities. In the case of regression, a margin of tolerance (epsilon) is set in approximation to the SVM which would have already requested from the problem. But besides this fact, there is also a more complicated reason, the algorithm is more complicated therefore to be taken in consideration. However, the main idea is always the same: to minimize error, individualizing the hyperplane which maximizes the margin, keeping in mind that part of the error is tolerated.

Instead of minimizing the error rate as in simple linear regression, we try to fit the error within a certain threshold. Our objective in SVR is to basically consider the points that are within the margin. Our best fit line is the hyperplane that has maximum number of points.

1.3.4 Decision Tree Regression

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. Decision trees learn from data to approximate a sine curve with a set of if-then-else decision rules. The deeper the tree, the more complex the decision rules and the fitter the model.

Decision tree builds classification or regression models in the form of a tree structure. It breaks down a data set into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node has two or more branches. Leaf node represents a classification or decision. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data.

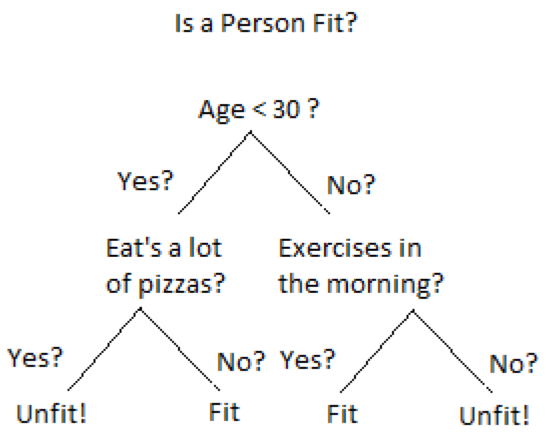


Fig 1.1 Working of a decision tree

A decision tree is built by partitioning the data into subsets containing instances with similar values (homogenous). Standard deviation is used to calculate the homogeneity of a numerical sample. If the numerical sample is completely homogeneous, its standard deviation is zero.

The steps for finding splitting node is briefly described as below:

- Calculate standard deviation of target variable
- Split the dataset on different attributes and calculate standard deviation for each branch (standard deviation for target and predictor). This value is subtracted from the standard deviation before the split. The result is the standard deviation reduction.
- The attribute with the largest standard deviation reduction is chosen as the splitting node.

- The dataset is divided based on the values of the selected attribute. This process is run recursively on the non-leaf branches, until all data is processed.

To avoid overfitting, Coefficient of Deviation (CV) is used which decides when to stop branching. Finally, the average of each branch is assigned to the related leaf node (in regression mean is taken where as in classification mode of leaf nodes is taken).

1.3.5 Random Forest Regression

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

The first algorithm for random decision forests was created by Tin Kam Ho using the random subspace method, which, in Ho's formulation, is a way to implement the "stochastic discrimination" approach to classification proposed by Eugene Kleinberg.

An extension of the algorithm was developed by Leo Breiman and Adele Cutler, who registered "Random Forests" as a trademark. The extension combines Breiman's "bagging" idea and random selection of features, introduced first by Ho and later independently by Amit and Geman in order to construct a collection of decision trees with controlled variance. Decision trees are a popular method for various machine learning tasks. Tree learning "come[s] closest to meeting the requirements for serving as an off-the-shelf procedure for data mining", say Hastie et al., "because it is invariant under scaling and various other transformations of feature values, is robust to inclusion of irrelevant features, and produces inspectable models. However, they are seldom accurate".

In particular, trees that are grown very deep tend to learn highly irregular patterns: they overfit their training sets, i.e. have low bias, but very high variance. Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance. This comes at the expense of a small increase in the bias and some loss of interpretability, but generally greatly boosts the performance in the final model.

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set X = x1, ..., xn with responses Y

= y1, ..., yn, bagging repeatedly (B times) selects a random sample with replacement of the training set and fits trees to these samples:

For b = 1, ..., B:

Sample, with replacement, n training examples from X, Y; call these Xb, Yb.

Train a classification or regression tree fb on Xb, Yb.

After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x' or by taking the majority vote in the case of classification trees.

This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, if the trees are not correlated. Simply training many trees on a single training set would give strongly correlated trees (or even the same tree many times, if the training algorithm is deterministic); bootstrap sampling is a way of de-correlating the trees by showing them different training sets.

Additionally, an estimate of the uncertainty of the prediction can be made as the standard deviation of the predictions from all the individual regression trees on x':

The number of samples/trees, B, is a free parameter. Typically, a few hundred to several thousand trees are used, depending on the size and nature of the training set. An optimal number of trees B can be found using cross-validation, or by observing the out-of-bag error: the mean prediction error on each training sample x_i, using only the trees that did not have x_i in their bootstrap sample. The training and test error tend to level off after some number of trees have been fit.

The above procedure describes the original bagging algorithm for trees. Random forests differ in only one way from this general scheme: they use a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features. This process is sometimes called "feature bagging". The reason for doing this is the correlation of the trees in an ordinary bootstrap sample: if one or a few features are very strong predictors for the response variable (target output), these features will be selected in many of the B trees, causing them to become correlated. An analysis of how bagging and random subspace projection contribute to accuracy gains under different conditions is given by Ho.

Typically, for a classification problem with p features, √p (rounded down) features are used in each split. For regression problems the inventors recommend p/3 (rounded down) with a minimum node size of 5 as the default.

Adding one further step of randomization yields extremely randomized trees, or Extra Trees. While similar to ordinary random forests in that they are an ensemble of individual trees, there are two main differences: first, each tree is trained using the whole learning sample (rather than a bootstrap sample), and second, the top-down splitting in the tree learner is randomized. Instead of computing the locally optimal cut-point for each feature under consideration (based on, e.g., information gain or the Gini impurity), a random cut-point is selected. This value is selected from a uniform distribution within the feature's empirical range (in the tree's training set). Then, of all the randomly generated splits, the split that yields the highest score is chosen to split the node.

1.4 AdaBoost

AdaBoost, short for Adaptive Boosting, is a machine learning meta-algorithm formulated by Youv Freund. It can be used in conjunction with many other types of learning algorithms to improve performance. The output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. AdaBoost is sensitive to noisy data and outliers. In some problems it can be less susceptible to the overfitting problem than other learning algorithms. The individual learners can be weak, but as long as the performance of each one is slightly better than random guessing, the final model can be proven to converge to a strong learner.

Every learning algorithm tends to suit some problem types better than others, and typically has many different parameters and configurations to adjust before it achieves optimal performance on a dataset, AdaBoost (with decision trees as the weak learners) is often referred to as the best out-of-the-box classifier. When used with decision tree learning, information gathered at each stage of the AdaBoost algorithm about the relative 'hardness' of each training sample is fed into the tree growing algorithm such that later trees tend to focus on harder-to-classify examples.

Problems in machine learning often suffer from the curse of dimensionality — each sample may consist of a huge number of potential features (for instance, there can be 162,336

Haar features, as used by the Viola-Jones object detection framework, in a 24×24-pixel image window), and evaluating every feature can reduce not only the speed of classifier training and execution, but in fact reduce predictive power. Unlike neural networks and SVMs, the AdaBoost training process selects only those features known to improve the predictive power of the model, reducing dimensionality and potentially improving execution time as irrelevant features need not be computed.

Boosting can be seen as minimization of a convex loss function over a convex set of functions. Specifically, the loss being minimized by AdaBoost is the exponential loss $\sum_i \phi(i, y, f) = \sum_i e^{-y_i f(x_i)}$, whereas Logit Boost performs logistic regression, minimizing $\sum_i \phi(i, y, f) = \sum_i \ln\left(1 + e^{-y_i f(x_i)}\right)$ In the gradient descent analogy, the output of the classifier for each training point is considered a point $(F_t(x_1), \ldots, F_t(x_n))$ in n-dimensional space, where each axis corresponds to a training sample, each weak learner $h(x)$ corresponds to a vector of fixed orientation and length, and the goal is to reach the target point $(y_1, \ldots, y_n)$ (or any region where the value of loss function $E_T(x_1, \ldots, x_n)$ is less than the value at that point), in the least number of steps. Thus, AdaBoost algorithms perform either Cauchy

(find $h(x)$ with the steepest gradient, choose α to minimize test error) or Newton (choose some target point, find $\alpha h(x)$ that brings F_t closest to that point) optimization of training error.

1.5 Principal component analysis

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of linearly uncorrelated variables called principal components. This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. The resulting vectors (each being a linear combination of the variables and containing n observations) are an uncorrelated orthogonal basis set. PCA is sensitive to the relative scaling of the original variables.

PCA was invented in 1901 by Karl Pearson, as an analogue of the principal axis theorem in mechanics; it was later independently developed and named by Harold Hotelling in the 1930s. Depending on the field of application, it is also named the discrete Karhunen-Loeve transform (KLT) in signal processing, the Hotelling transform in multivariate quality control, proper orthogonal decomposition (POD) in mechanical engineering, singular value decomposition (SVD) of X (Golub and Van Loan, 1983), eigenvalue decomposition (EVD) of XTX in linear algebra, factor analysis.

PCA is mostly used as a tool in exploratory data analysis and for making predictive models. It is often used to visualize genetic distance and relatedness between populations. PCA can be done by eigenvalue decomposition of a data covariance matrix or singular value decomposition of a data matrix, usually after a normalization step of the initial data. The normalization of each attribute consists of mean centering – subtracting each data value from its variable's measured mean so that its empirical mean (average) is zero – and, possibly, normalizing each variable's variance to make it equal to 1; The results of a PCA are usually discussed in terms of component scores, sometimes called factor scores (the transformed variable values corresponding to a particular data point), and loadings (the weight by which each standardized original variable should be multiplied to get the component score). If component scores are standardized to unit variance, loadings must contain the data variance in them (and that is the magnitude of eigenvalues). If component scores are not standardized (therefore they contain the data variance) then loadings must be unit-scaled, ("normalized") and these weights are called eigenvectors; they are the cosines of orthogonal rotation of variables into principal components or back.

1.6 Grid search

The traditional way of performing hyperparameter optimization has been grid search, or a parameter sweep, which is simply an exhaustive searching through a manually specified subset of the hyperparameter space of a learning algorithm. A grid search algorithm must be guided by some performance metric, typically measured by cross-validation on the training set or evaluation on a held-out validation set.

Since the parameter space of a machine learner may include real-valued or unbounded value spaces for certain parameters, manually set bounds and discretization may be necessary before applying grid search.

For example, a typical soft-margin SVM classifier equipped with an RBF kernel has at least two hyperparameters that need to be tuned for good performance on unseen data: a regularization constant C and a kernel hyperparameter γ.

Grid search then trains an SVM with each pair (C, γ) in the Cartesian product of these two sets and evaluates their performance on a held-out validation set (or by internal cross-validation on the training set, in which case multiple SVMs are trained per pair). Finally, the grid search algorithm outputs the settings that achieved the highest score in the validation procedure.

Grid search suffers from the curse of dimensionality but is often embarrassingly parallel because typically the hyperparameter settings it evaluates are independent of each other.

1.7 Motivation and Scope

As important functional materials, high-transition temperature (high-TC) superconductors have some typical physical parameters, such as transition temperature Tc, magnetic susceptibility and critical current density (Jc), which make them very useful in many practical applications like magnetically levitated trains and power transmission. Previous researches showed that the high-Tc superconductors are generally characterized by a two-dimensional layered superconducting condensate with unique features that are not traditional superconducting metals. Their important property, Tc, is determined by their layered crystals, bond lengths, valency properties of the ions, and Coulomb coupling between electronic bands in adjacent, spatially separated layers.

It is clear that Tc (critical temperature) of superconductors depend on its other chemical properties. In this project we utilize the already available data about superconductors to estimate the critical temperature of new potential materials. The developed model can be used to gain confidence on a new material to conduct further research on it regarding its superconducting behavior. Also, experimental determination of critical temperature is a laborious process which is made easy when an estimate of the value is provided by our model.

1.8 Need for study

Superconductors are used in MRI scans, particle accelerators, plasma fusion reactors, power cables and electronic devices. They will also be a part of futuristic projects like magnetic levitated trains and hyperloops. So rapid experimentation is needed to produce new superconductors with different properties.

Understanding the relationship between superconductivity and materials' chemistry and structure presents significant theoretical and experimental challenges. In particular, despite focused research efforts in the last 30 years, the mechanisms responsible for high-temperature superconductivity in cuprite and iron-based families remain elusive. Recent developments, however, allow a different approach to investigate what ultimately determines the superconducting critical temperatures (Tc) of materials. This project tries to eliminate the laborious task of going through the entire experimental procedure of determining the critical temperature and gives an estimate of critical temperature of a superconductor based on its chemical properties.

1.9 Literature survey

Several researches have been done from past three decades on machine learning modelling of superconducting critical temperature. They concluded that machine-learning methods can be applied to some domains of materials and the PCA-PSO-SVR ensemble method may be used to predict the Tc of new high-Tc superconductors C from structural and correlative electronic parameters.

In [1] the author made a point that 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. 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 in [3] the author suggests that in particular, despite focused research efforts in the last 30 years, the mechanisms responsible for high-temperature superconductivity in cuprite and iron-based families remain elusive.

In [2] the author shows that there is even a profound connection between phenomena in the superconducting state and the Higgs mechanism in particle physics. However, understanding the relationship between superconductivity and materials' chemistry and structure presents significant theoretical and experimental challenges. In [4] the writer mentions extensive databases covering various measured and calculated materials properties have been created over the years. The sheer quantity of accessible information also makes possible, and even necessary, the use of data-driven approaches.

In [5][6] it is suggested that statistical and machine learning (ML) methods 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, the likely crystal structure at a given composition, band gap energies, and density of states of certain classes of materials.

In [7] we learn that to further improve the predictive power of the models, as well as the ability to extract useful information out of them, another set of features are constructed based on crystallographic and electronic information. Such information is not generally available in SuperCon, we employ data from the AFLOW Online Repositories. The materials database houses nearly 170 million properties calculated. In [12] we came to know that the materials database houses nearly 170 million properties calculated with the software package AFLOW. It contains information for the vast majority of compounds in the ICSD. And in [11] the author explains that. To convert this information into meaningful features/predictors (used interchangeably), we employ the Materials Agnostic Platform for Informatics and Exploration (Magpie). Magpie computes a set of attributes for each material, including elemental property statistics like the mean and the standard deviation of 22 different elemental properties. Despite the success of Magpie predictors in modeling materials properties, interpreting their connection to superconductivity presents a serious challenge.

[8] talks about application of statistical methods in the context of superconductivity began in the early eighties with simple clustering methods. 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. On the other hand [9] exposes the limitations of relying on random-guess (trial-and-error) approaches for breakthrough discoveries. Subsequently, this study also highlights the impact machine learning can have on this particular field.

In another early work [10], statistical methods were used to find correlations between normal state properties and Tc of the metallic elements in the first six rows of the periodic table. Other contemporary works hone in on specific materials and families of superconductors. According to [13] [14] once we have a list of relevant predictors, various ML models can be applied to the data. All ML algorithms in this work are variants of the random forest method. In the random forest method, features can be ordered by their importance quantified via the so-called Gini importance or –mean decrease in impurity‖. For a given feature, it is the sum of the Gini impurity (calculated as ∑ i p i (1 - p i), where p i is the probability of randomly chosen data point from a given decision tree leaf to be in class i) over the number of splits that include the feature, weighted by the number of samples it splits, and averaged over the entire forest. Also [15] states that Random forest is one of the most powerful, versatile, and widely used ML methods. There are several advantages that make it especially suitable for this problem.

In [16] the author carries out the data cleaning and processing using the Python Pandas package for data analysis. And from [17] the random forest models above are developed using scikit-learn—a powerful and efficient machine learning Python library. 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.

1.10 Research Gap

There have been studies on how a Random forest model is suitable for estimation of superconducting critical temperature of material based on its chemical properties. Previous research has shown how a PCA-PSO-SVR ensemble model works on this task. But no research has mentioned how a PCA version of Random forest will perform this specific task. There are also new ensemble methods which use adaptive boosting to boost the model performance. This project focuses on the using a PCA-PSO-RandomForest ensemble model and improves with the usage of Adaptive boosting. This is a relatively new approach and produces better results compared to its predecessor models.

Chapter 2

2.1 Problem statement

The goal of the projects is to develop a model, taking advantage of immense increase in data available about superconductors to predict the critical temperature of a superconductors based on the chemical properties of elements in its composition. A regression model developed based on supervised learning algorithms can achieve this task.

2.2 Objectives

- To eliminate the laborious task of experimentation to find the critical temperature of a superconductor
- To find whether a material acts as a superconductor or not
- To provide an estimate of superconducting critical temperature of a material

2.3 Hypothesis

The critical temperature of a superconducting material can be estimated by a regression model trained by supervised learning algorithms using chemical characteristics of materials.

Chapter 3

3.1 Research methodology

The problem to estimate the superconducting critical temperature based on its chemical properties is a data driven approach. The research which is already done covers some vintage data driven statistical approaches. But with availability of more powerful machine learning algorithms, this project introduces an efficient and better approach to solve this regression problem.

The research methodology in this project include,

- Visualizing and understanding the data
- Exploring different suitable models
- Agreeing on a common evaluation metric
- Training and Testing the models
- Deciding on the final model
- Refining the final model
- Analyzing the result

3.1.1 Data visualization

The data visualization is carried out using the matplotlib.pyplot package. Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+. There is also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged. SciPy makes use of Matplotlib. Matplotlib was originally written by John D. Hunter, has an active development community, and is distributed under a BSD-style license. Michael Droettboom was nominated as matplotlib's lead developer shortly before John Hunter's death in August 2012, and further joined by Thomas Caswell.

As of 23 June 2017, matplotlib 2.0.x supports Python versions 2.7 through 3.6. Matplotlib 1.2 is the first version of matplotlib to support Python 3.x. Matplotlib 1.4 is the last version of Matplotlib to support Python 2.6.

3.1.2 Learning models

The different regression model were chosen from scikit learn. Scikit-learn (formerly scikits.learn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

Scikit-learn is largely written in Python, with some core algorithms written in Cython to achieve performance. Support vector machines are implemented by a Cython wrapper around LIBSVM; logistic regression and linear support vector machines by a similar wrapper around LIBLINEAR.

3.2 Related work

A similar research work has been done by Yao Liu and his team, which has been publish under the name ―PREDICTION OF SUPERCONDUCTING TRANSITION TEMPERATURE USING A MACHINE-LEARNING METHOD‖. In this work, to predict TC of various high-TC superconductors, they established a PCA-PSO-SVR model based on a machine-learning method using structural and electronic parameters. These parameters, are related to 31 kinds of high-TC superconductors that form the dataset from the literature.7,9 The dataset has only 31 samples and each sample has only 7 features, which is obviously a small sample set, but the SVR shows many unique advantages of processing small sample sets because of the theory of statistical learning and the minimum principle of structural risk. Hence, they chose the SVR as the regression algorithm of the prediction model. To achieve a higher performance of the model, they adopted automatic optimization with a simple and efficient PSO18 optimization algorithm instead of the manual optimization used in the previous studies when searching for the optimal SVR parameters. Meanwhile, they found that some parameters are interdependent by analyzing the crystal structure and parameters of the high-TC superconductors, so they used PCA19 to reduce dimensions and interdependencies in the data pre-processing for a better accuracy of the prediction model. In addition, they also trained the PSO-SVR model and the back-propagation neural network (BPNN)20 with the dataset for comparison. The corresponding experimental results showed that the PCA-PSO-SVR prediction model is more accurate when predicting TC.

Chapter 4

4.1 Software requirements

Operating system	: Windows 10
IDE	: Jupyter Notebook - Anaconda
Platform	: Python
Packages	: NumPy, Pandas, Sci-kit learn, Matplotlib
Other applications	: Google sheets or MS-Excel

4.2 Hardware requirements

Processor	: Intel Core i3
RAM	: 4 GB
Hard disk	: 128 GB
Monitor	: 15.6‖ color display
Mouse	: Three button optical mouse
Keyboard	: Standard qwerty keyboard

Chapter 5

Implementation

The goal is to create a regression model to predict the critical temperature of a super conductor. Tasks involved in the project implementation are as follows:

- Gathering, analyzing and preprocessing the data (data exploration)
- Set a benchmark model and evaluation metric
- Training different regression models
- Evaluate the models on a metric and compare them to the benchmark
- Create a well-tuned final model

The final model is expected to give an estimate of critical temperature of a super conductor based on its chemical properties. And this is the architecture of a machine learning model,

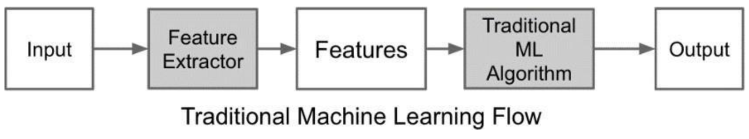


Fig 5.1 Machine Learning work flow

5.1 Data exploration

Some of the features provided in the dataset are

- Number of elements
- Mean atomic mass
- Entropy atomic mass
- Mean atomic radius
- Mean density
- Mean electron affinity
- Mean fusion heat
- Mean thermal conductivity
- Mean valence

These are some properties of the elemental composition of the superconductors provided in the data set as real valued numbers.

The data set is taken from the UCI data repository at:
https://archive.ics.uci.edu/ml/datasets/Superconductivty+Data

The characteristics of the dataset are:

Data Set Characteristics:	Multivariate	Number of Instances:	21263
Attribute Characteristics:	Real	Number of Attributes:	81
Associated Tasks:	Regression	Missing Values?	N/A

Table 5.1 characteristics of the dataset

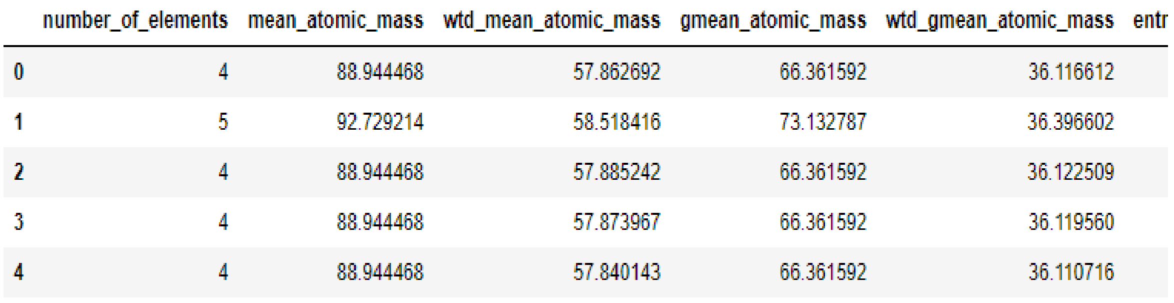
The above table describes the dataset features. As stated above the data set is multivariate with 80 input variables and one output variable. All the values in the dataset are real valued numbers. The dataset comes with no missing values and there are 21263 instances of the values. The task associated with the data is regression.

The data came in the form of a excel sheet. From this format the data is extracted into a pandas' dataframe. The input data is stored in the x variable and the output data in y variable. Both x and y are pandas' dataframes with dimensions 21263*80 and 21263*1 respectively.

Here is a glimpse at the data and the code leading it,

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import time
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
#reading from csv file
data = pd.read_csv('dataset.csv')
```

```
#seperating target variable into y
y = pd.DataFrame(data['critical_temp'], columns=['critical_temp'])
#all features are stored in x
x = data.drop(columns='critical_temp', axis=1)
#display first 5 rows in dataset
x.head()
```

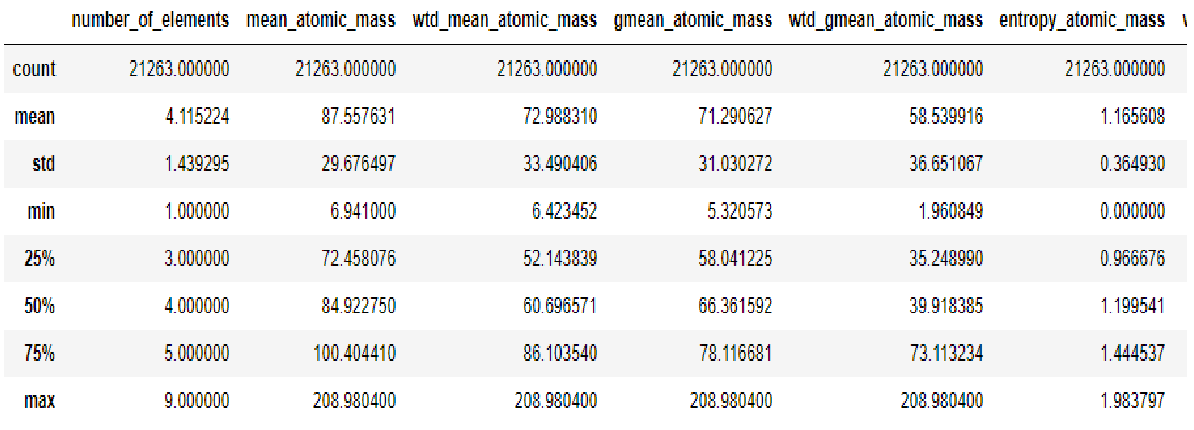


	number_of_elements	mean_atomic_mass	wtd_mean_atomic_mass	gmean_atomic_mass	wtd_gmean_atomic_mass	entr
0	4	88.944468	57.862692	66.361592	36.116612	
1	5	92.729214	58.518416	73.132787	36.396602	
2	4	88.944468	57.885242	66.361592	36.122509	
3	4	88.944468	57.873967	66.361592	36.119560	
4	4	88.944468	57.840143	66.361592	36.110716	

Table 5.2 Initial look at data

To analyze data, we need to understand the input variables so their statistical properties like mean, standard deviation, minimum values, etc. are studies. The description of the input variables is gathered with

```
#description of input variables
x.describe()
```

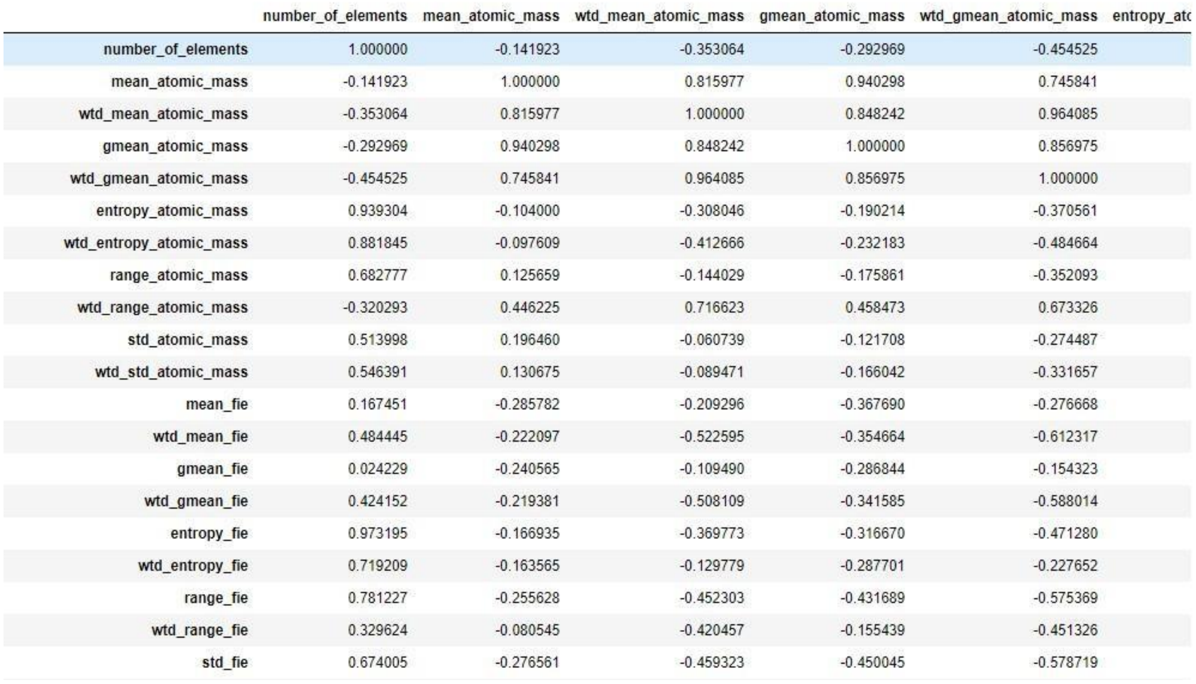


	number_of_elements	mean_atomic_mass	wtd_mean_atomic_mass	gmean_atomic_mass	wtd_gmean_atomic_mass	entropy_atomic_mass
count	21263.000000	21263.000000	21263.000000	21263.000000	21263.000000	21263.000000
mean	4.115224	87.557631	72.988310	71.290627	58.539916	1.165608
std	1.439295	29.676497	33.490406	31.030272	36.651067	0.364930
min	1.000000	6.941000	6.423452	5.320573	1.960849	0.000000
25%	3.000000	72.458076	52.143839	58.041225	35.248990	0.966676
50%	4.000000	84.922750	60.696571	66.361592	39.918385	1.199541
75%	5.000000	100.404410	86.103540	78.116681	73.113234	1.444537
max	9.000000	208.980400	208.980400	208.980400	208.980400	1.983797

Table 5.3 Description of data

Since working with too many features puts so much burden on the learning model, a feature extraction method is employed to see that the number features can be reduced. So, find out if it possible or not the correlations between the features are calculated with,

```
#checking for correlaitons
corr = x.corr()
#display first five rows of correlation table
corr.head()
```



	number_of_elements	mean_atomic_mass	wtd_mean_atomic_mass	gmean_atomic_mass	wtd_gmean_atomic_mass	entropy_atc
number_of_elements	1.000000	-0.141923	-0.353064	-0.292969	-0.454525	
mean_atomic_mass	-0.141923	1.000000	0.815977	0.940298	0.745841	
wtd_mean_atomic_mass	-0.353064	0.815977	1.000000	0.848242	0.964085	
gmean_atomic_mass	-0.292969	0.940298	0.848242	1.000000	0.856975	
wtd_gmean_atomic_mass	-0.454525	0.745841	0.964085	0.856975	1.000000	
entropy_atomic_mass	0.939304	-0.104000	-0.308046	-0.190214	-0.370561	
wtd_entropy_atomic_mass	0.881845	-0.097609	-0.412666	-0.232183	-0.484664	
range_atomic_mass	0.682777	0.125659	-0.144029	-0.175861	-0.352093	
wtd_range_atomic_mass	-0.320293	0.446225	0.716623	0.458473	0.673326	
std_atomic_mass	0.513998	0.196460	-0.060739	-0.121708	-0.274487	
wtd_std_atomic_mass	0.546391	0.130675	-0.089471	-0.166042	-0.331657	
mean_fie	0.167451	-0.285782	-0.209296	-0.367690	-0.276668	
wtd_mean_fie	0.484445	-0.222097	-0.522595	-0.354664	-0.612317	
gmean_fie	0.024229	-0.240565	-0.109490	-0.286844	-0.154323	
wtd_gmean_fie	0.424152	-0.219381	-0.508109	-0.341585	-0.588014	
entropy_fie	0.973195	-0.166935	-0.369773	-0.316670	-0.471280	
wtd_entropy_fie	0.719209	-0.163565	-0.129779	-0.287701	-0.227652	
range_fie	0.781227	-0.255628	-0.452303	-0.431689	-0.575369	
wtd_range_fie	0.329624	-0.080545	-0.420457	-0.155439	-0.451326	
std_fie	0.674005	-0.276561	-0.459323	-0.450045	-0.578719	

Table 5.4 Correlation table

It is clear from the above table that there are some features with high correlation. This allows us to do Principal component analysis on dataset and project the data to lower dimensions. After doing the PCA the 80 features in the input data are projected to 3 features and the variance explained by these three new dimensions and the corresponding code is,

```
good_data = x
dimensions = dimensions = ['Dimension {}'.format(i) for i in
range(1,len(pca.components_)+1)]
```

```
# PCA explained variance
ratios = pca.explained_variance_ratio_.reshape(len(pca.components_), 1)
variance_ratios = pd.DataFrame(np.round(ratios, 4), columns = ['Explained Variance'])
variance_ratios.index = dimensions
# Return a concatenated DataFrame
variance_ratios
```

	Explained Variance
Dimension 1	0.6240
Dimension 2	0.2990
Dimension 3	0.0492

Table 5.5 Values of explained variance

The total variance explained by these 3 dimensions is 97.22%. Any new dimensions would not contribute much in terms of explained variance and this is evident by the graph below,

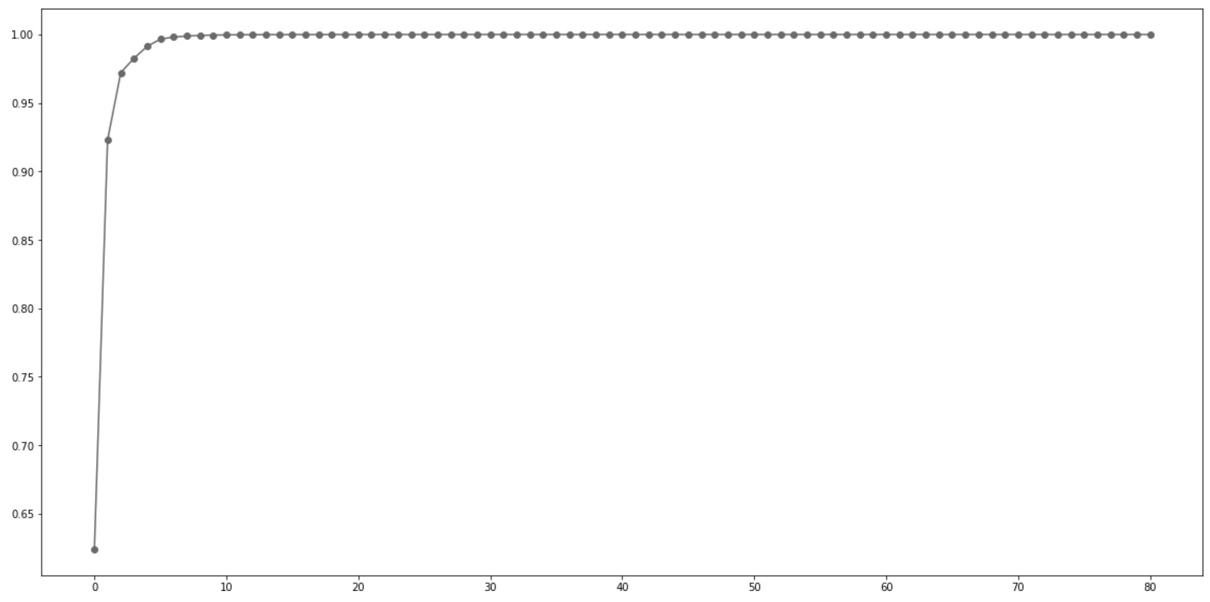
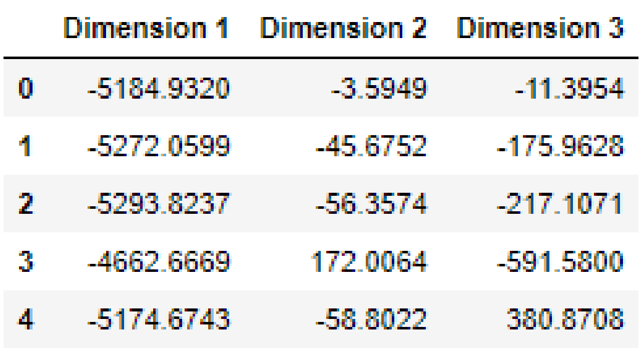


Fig 5.2 Plot of explained variances

The new transformed data after doing principal component analysis looks like,



	Dimension 1	Dimension 2	Dimension 3
0	-5184.9320	-3.5949	-11.3954
1	-5272.0599	-45.6752	-175.9628
2	-5293.8237	-56.3574	-217.1071
3	-4662.6669	172.0064	-591.5800
4	-5174.6743	-58.8022	380.8708

Table 5.6 Values of transformed data

And the code accompanying it is,

```
#performing pca with 3 dimensions
pca = PCA(n_components=3)
pca.fit(x)

#transfroming 81 features into 3 dimensional data
reduced_data = pca.transform(x)
new_x = pd.DataFrame(reduced_data, columns = ['Dimension '+str(i) for i in range(3)])

#display new data
new_x.head()
```

Here that data is reduced to three dimensions instead being in 80 dimensions, but the information itself is not compromised as the new dataset can explain over 97% of trends in the original data.

5.2 Benchmark model and Evaluation metric

5.2.1 Evaluation metric

Since the task associated with the project is regression, there are not many metrics for evaluating the performance of a regression model. Among the few, the best suited evaluation metric in this scenario would be the R2 score or the coefficient of determination.

R2_score (Coefficient of determination) is a common metric for a regression model; it is a statistical measure of how well the regression predictions approximate the real data points.

$$R^2 \equiv 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$

R2_score is better for this project than other regression metrics like Root_Mean_Squarred error or Mean_Absolute error because, R^2 does a better job than RMSE or MAE whose scope is limited to comparing predicted values with actual values. Also, the absolute value of RMSE does not actually tell how bad a model is. It can only be used to compare across two models whereas R^2 easily does that.

R2_score can be problematic if variables that are the result of the dependent variable are included as predictors. But the features used to predict the critical temperature are not dependent on the output variable, because they are the chemical properties of the elements in the superconductor.

5.2.2 Benchmark model

The benchmark r2_score, training time and predicting time are created based on the linear regression model's performance. Linear regression is the simplest regression model. As the Occam's Razor suggests going with the simplest model, it is first checked whether the data follows a linear trend or not.

Further steps in project implementation include,

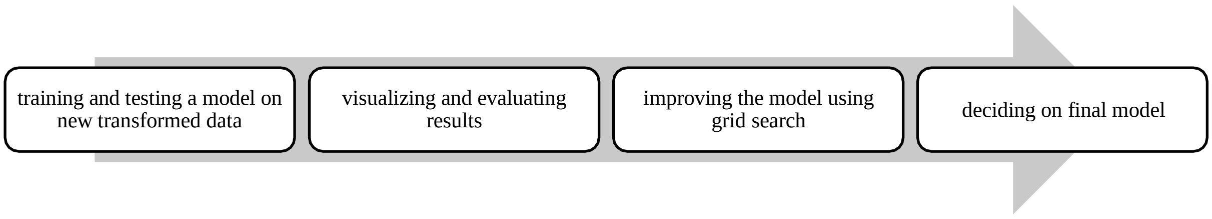

training and testing a model on new transformed data
visualizing and evaluating results
improving the model using grid search
deciding on final model

Fig 5.3 Steps involved in the project

5.3 Training models

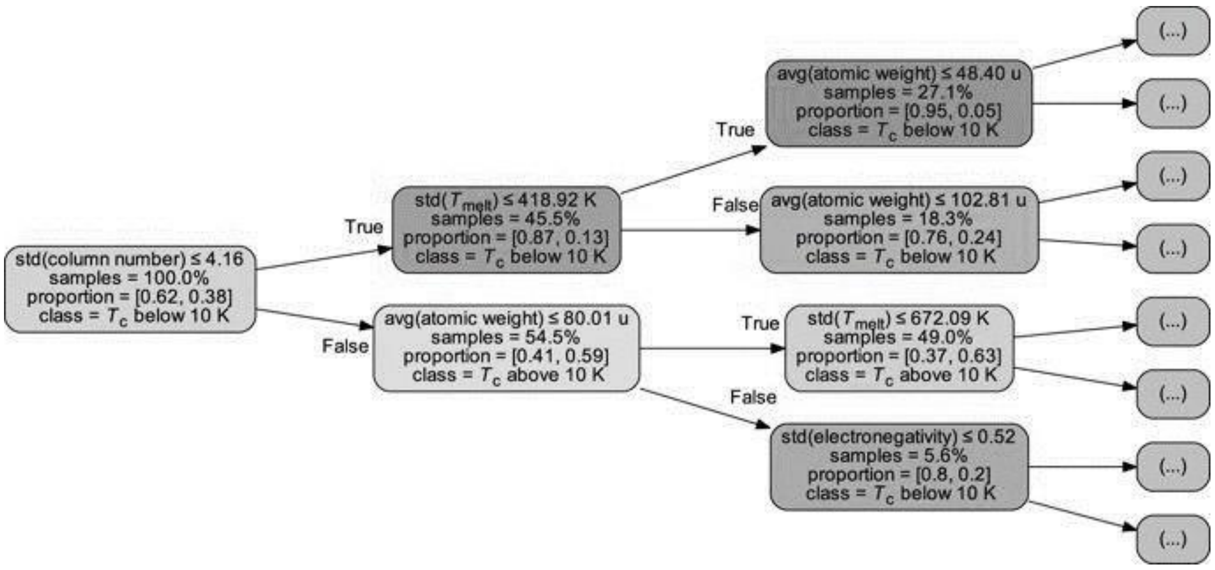
An initial look at the output variable suggests that it mimics stepwise data. Amongst the different regression models the decision tree regression model comes first to mind when dealing with step wise data. So initially a decision tree model is trained to fit the data.
The steps included in training models are,

- Splitting the data into train and test sets

- Creating a regression model
- Training the model on the train data
- Evaluating the model on test data
- Evaluating the model performance with the metric chosen

The data is split into train and test datasets in the ration of 70:30 using a train_test_split model from the model selection package of scikit learn. Then a Decision tree regressor model is created and trained on train data set. This model is then evaluated on test data set. The performance of the model is then measured using the evaluation model.

This is how a decision tree works,



std(column number) ≤ 4.16
samples = 100.0%
proportion = [0.62, 0.38]
class = T_c below 10 K
True
False
std(T_{melt}) ≤ 418.92 K
samples = 45.5%
proportion = [0.87, 0.13]
class = T_c below 10 K
avg(atomic weight) ≤ 80.01 u
samples = 54.5%
proportion = [0.41, 0.59]
class = T_c above 10 K
avg(atomic weight) ≤ 48.40 u
samples = 27.1%
proportion = [0.95, 0.05]
class = T_c below 10 K
avg(atomic weight) ≤ 102.81 u
samples = 18.3%
proportion = [0.76, 0.24]
class = T_c below 10 K
std(T_{melt}) ≤ 672.09 K
samples = 49.0%
proportion = [0.37, 0.63]
class = T_c above 10 K
std(electronegativity) ≤ 0.52
samples = 5.6%
proportion = [0.8, 0.2]
class = T_c below 10 K
(...)

Fig 5.4 Decision tree working

The decision process is taken by choosing the criteria or attribute with highest entropy value and tree is constructed until the threshold of minimum entropy reaches. Finally, the leaf nodes are the classes in this case the estimate of the output variable.

The code of the above process is,

```
results = {}

#splitting the data into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=42)
```

```
#training the learner
start = time.time()
learner = DecisionTreeRegressor().fit(x,y)
end = time.time()

results['train_time'] = end - start

#predicting the output variable
start = time.time()
predictions_test = learner.predict(x_test)
predictions_train = learner.predict(x_train)
end = time.time()

results['pred_time'] = end - start

#calculating the score of the learner
results['score_train'] = learner.score(x_train, y_train)
results['score_test'] = learner.score(x_test, y_test)

print("{} trained.".format(learner))
```

The results after performing the decision tree regression are as follows,

```
DecisionTreeRegressor
{
        'train_time': 0.17719674110412598 seconds,
        'pred_time': 0.00800633430480957 seconds,
        'score_train': 0.7845250110933649,
        'score_test': 0.7835929355566275
}
```

After the training the decision tree model the results were moderate in terms of fitting the data with 78% r2_score on the data. But the model does very well when it comes to training and prediction time. This suggests that a more powerful model can be used for this regression task.

To improve the model performance an ensemble method is used. The Random forest is an ensemble method which combines multiple decision trees. The code used to train and evaluate a Random forest model is,

```
results = {}

#splitting the data into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=42)

#training the learner
start = time.time()
learner = RandomForestRegressor().fit(x,y)
end = time.time()

results['train_time'] = end - start

#predicting the output variable
start = time.time()
predictions_test = learner.predict(x_test)
predictions_train = learner.predict(x_train)
end = time.time()

results['pred_time'] = end - start

#calculating the score of the learner
results['score_train'] = learner.score(x_train, y_train)
results['score_test'] = learner.score(x_test, y_test)
```

```
    print("{} trained.".format(learner))
```

After training the random forest model the results were as follows,

```
RandomForestRegressor
{
        'train_time': 0.8048872947692871 seconds,
        'pred_time': 0.08109617233276367 seconds,
        'score_train': 0.9230448962888124,
        'score_test': 0.9226525108509567
}
```

It is discussed in the previous papers how a random forest model can do well in this kind of project and it is evident by the results shown here. The model does well in terms of r2_score with over 92% and it does in less than a second. The random forest algorithm's working is explained in the below diagram,

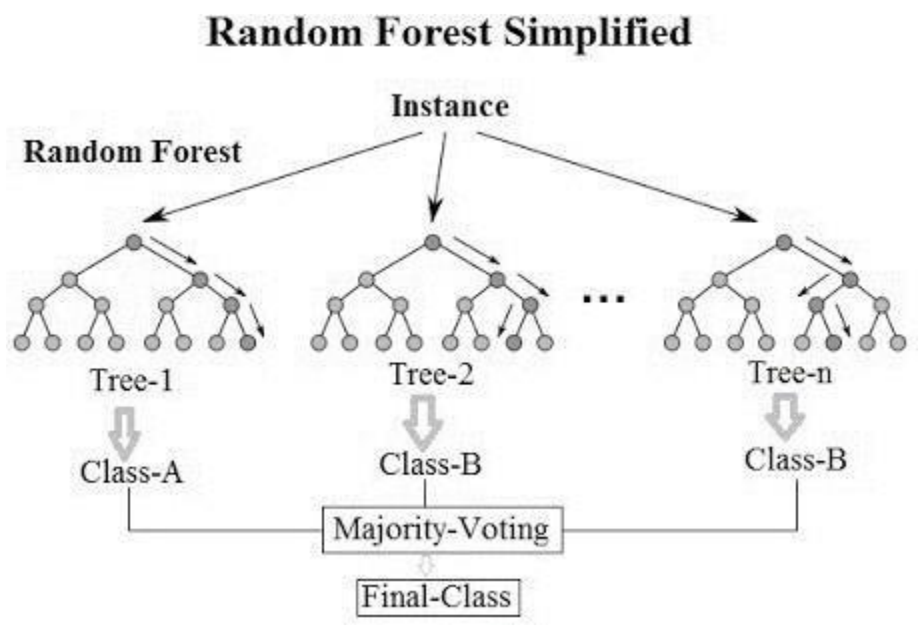


Fig 5.5 Working of Random Forest

There is still room for improving the model. Since, it the random forest generates trees randomly and those trees are essentially weak leaners. The algorithm combines the weak learners to produce the end result. This approach of random tree generation can be replaced with a comparatively new and efficient ensemble method called Adaboost. Adaboost generates trees which will perform well on the areas its predecessors couldn’t. The code for the new adaboost algorithm is,

```
results = {}

#splitting the data into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=42)

#training the learner
start = time.time()
learner = AdaBoostRegressor(DecisionTreeRegressor()).fit(x,y)
end = time.time()

results['train_time'] = end - start

#predicting the output variable
start = time.time()
predictions_test = learner.predict(x_test)
predictions_train = learner.predict(x_train)
end = time.time()

results['pred_time'] = end - start

#calculating the score of the learner
results['score_train'] = learner.score(x_train, y_train)
results['score_test'] = learner.score(x_test, y_test)

print("{} trained.".format(learner))
```

A Decision tree regressor is passed to the adaboost algorithm to boost it. Then all the estimators are combined to produce the best results and here they are,

```
AdaBoostRegressor
{
        'train_time': 1.5947909355163574 seconds,
        'pred_time': 0.20222735404968262 seconds,
        'score_train': 0.9684746229251533,
        'score_test': 0.9671931187916423
}
```

The working description of an Adaboost algorithm is shown below,

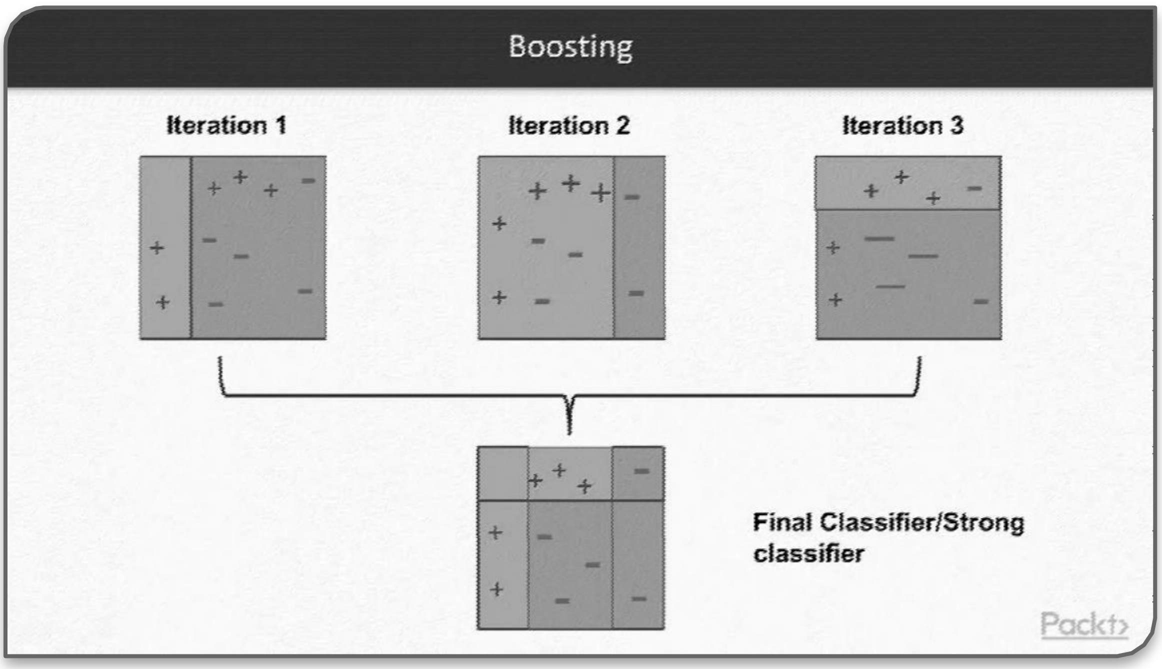


Fig 5.6 Working of Adaboost

The adaboost model performs better than other primitive models so its performance is compared to other models and chosen to be the best model for this particular regression task.

5.4 Comparison with benchmark model

As discussed earlier, the benchmark model is a linear regression model, the performance of other models when compared to the training time, prediction time, training score, and test score of the linear regression model is depicted in the below plot and the code for it is,

```
train_time = ( 0.04 , 0.17 , 0.80 , 1.60 )
pred_time = ( 0.03 , 0.01 , 0.08 , 0.07 )
train_score = ( 0.3 , 0.78 , 0.92 , 0.97 )
test_score = ( 0.3 , 0.78 , 0.92 , 0.97 )

fig, ax = plt.subplots(figsize=(10,5))

index = np.arange(4)
bar_width = 0.2

rects1 = ax.bar(index, train_time, bar_width, alpha=0.4, color='b', label='train_time')

rects2 = ax.bar(index + bar_width, pred_time, bar_width, alpha=0.4, color='r',
label='pred_time')

rects3 = ax.bar(index+ 2*bar_width, train_score, bar_width, alpha=0.4, color='y',
label='train_score')

rects4 = ax.bar(index + 3*bar_width, test_score, bar_width, alpha=0.4, color='g',
label='test_score')

ax.set_title('comparision of different models')
ax.set_xticks(index+bar_width*1.5)
ax.set_xticklabels(('Linear Regression', 'Decision Tree', 'Random Forest', 'Adaboost'))
ax.legend()
```

```
fig.tight_layout()
plt.show()
```

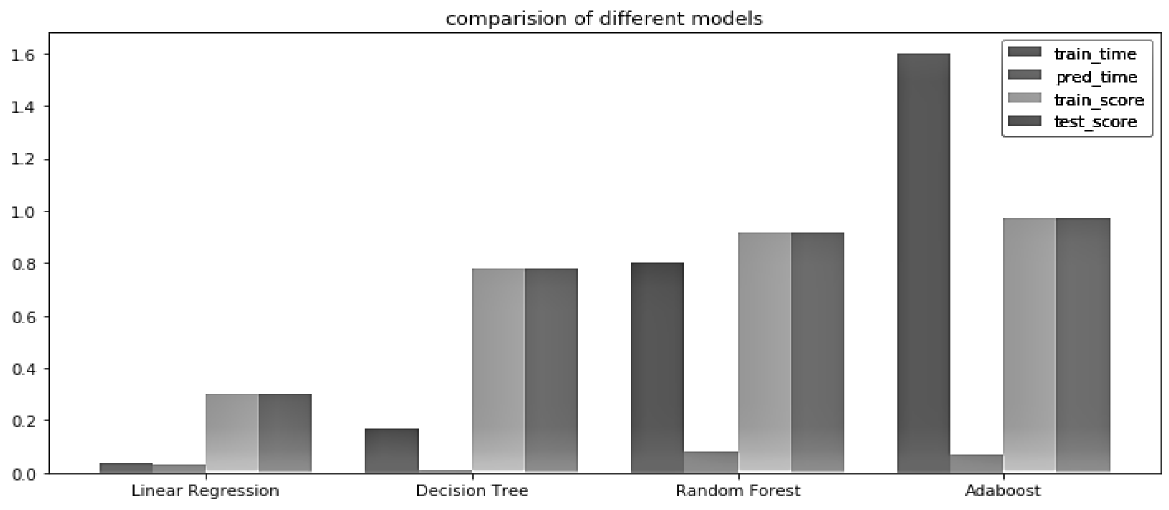


Fig 5.7 comparison of models

From the above graph it is clear that the adaboost model outperforms other models in terms of prediction power and does it in a reasonable amount of time. Hence, it is chosen as the final model to do the said regression task.

5.5 Tuning the final model

The final model must be tuned to improve its performance and one way to do so is using a grid search method. In grid search the parameters passed to the model are tuned by using cross validation. The parameters of the adaboost model which are tuned are
Number of estimators
Max depth of the tree

The number of estimators is the number of trees used for boosting and the max depth is the depth of the tree used for regression.
The code for the grid search is,

```
reg = AdaBoostRegressor(DecisionTreeRegressor())


 #training two best models with grid search
 parameters={}
```

```
  parameters[reg._class_. name ] =
{'n_estimators':[5,10,20,25],'max_depth':[5,15,20,30]}
  results = {}
  reg_name = reg.__class__.__name___
  results[reg_name] = {}
  results[reg_name] = train_predict_search(reg,parameters[reg_name], new_x,
y['critical_temp'])
```

The result of the grid search is as follows,

```
AdaBoostRegressor(base_estimator=DecisionTreeRegressor(criterion='mse', max_depth=25,
max_features=None,
max_leaf_nodes=None, min_impurity_decrease=0.0,
min_impurity_split=None, min_samples_leaf=1,
min_samples_split=2, min_weight_fraction_leaf=0.0,
presort=False, random_state=None, splitter='best'),
learning_rate=1.0, loss='linear', n_estimators=30,
random_state=None) trained.
```

Through grid search the best parameters for the adaboost model are found out to be, Number of estimators is 30 and maximum depth is 25. With this fine tuned model, we can produce the best result possible.

Chapter 6

Result Analysis

The final model is evaluated using the R2_score. The input data is split into train and test dataset. The train dataset is used to used to train the model and then the test dataset is used to see how the model does on previously unseen data.

6.1 Models summary

The summary of the performance of all models is

Model name	**R2_score**
Linear regression	30%
Decision tree regression	78%
Random forest regression	92%
Adaboost regression	96%

Table 6.1 summary of models performance

Since, the data doesn't have a linear trend, it is expected that the linear regression model doesn't perform well. So, it ended with 30% r2_score. The data is in a stepwise trend, so the decision tree model does pretty well with a 78% r2_score. The improvement for a simple decision tree model is an ensemble method. Random forest is introduced to improve the r2_score of the decision tree model. With the random forest model, the r2_score is increased to 92%. The final model choice would be Adaboost which is also a ensemble method. The best it could do is 96%.

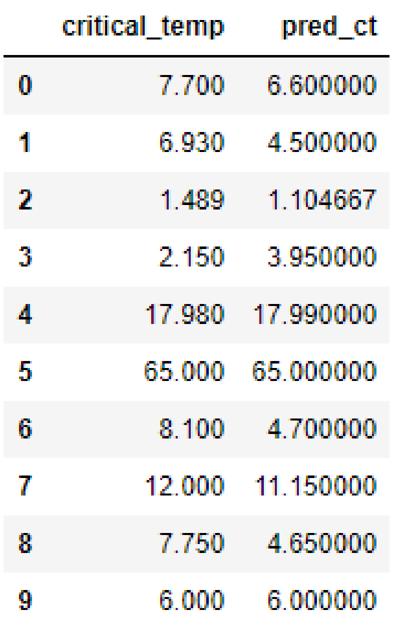
6.2 Final model analysis

It is clear that the adaboost outclasses other models in terms of r2_score. But this adaboost is further improved through gridsearch the final improved model's performance is noted as follows,

```
train_time: 0.4915473461151123
pred_time: 0.1772010326385498
score_train: 0.982052608960896
```

```
score_test: 0.9833312158619923
```

The final model is seen to do well with 98.33% r2_score, that means the final model can explain 98% of variance in the data. To see the final model in work, some actual and predicted values are,



	critical_temp	pred_ct
0	7.700	6.600000
1	6.930	4.500000
2	1.489	1.104667
3	2.150	3.950000
4	17.980	17.990000
5	65.000	65.000000
6	8.100	4.700000
7	12.000	11.150000
8	7.750	4.650000
9	6.000	6.000000

Table 6.2 actual and predicted values

And this is a result of the code,

```
from random import randint
indices = [randint(1,21000) for i in range(10)]
x_samples = pd.DataFrame(new_x.loc[indices], columns = new_x.keys()).reset_index(drop
= True)
y_samples = pd.DataFrame(y.loc[indices], columns = y.keys()).reset_index(drop = True)
pred = learner.predict(x_samples)
pred = pd.DataFrame(pred, columns = ['pred_ct']).reset_index(drop = True)
results = pd.concat([y_samples,pred], axis=1)
display(results)
#this is a table of actual and predicted critical temperature values
```

The actual and predicted values are quite near, this means the model works well. To take a deeper look a plot is drawn on actual and predicted values. The code and plot is,

```
from random import randint
indices = [randint(1,21000) for i in range(1000)]

x_samples = pd.DataFrame(new_x.loc[indices], columns = new_x.keys()).reset_index(drop
= True)
y_samples = pd.DataFrame(y.loc[indices], columns = y.keys()).reset_index(drop = True)
pred = learner.predict(x_samples)
pred = pd.DataFrame(sorted(pred), columns = ['pred_ct']).reset_index(drop = True)

plt.figure(figsize=(20,10))
plt.plot(sorted(y_samples['critical_temp']))
plt.plot(pred, 'r')
plt.show()
```

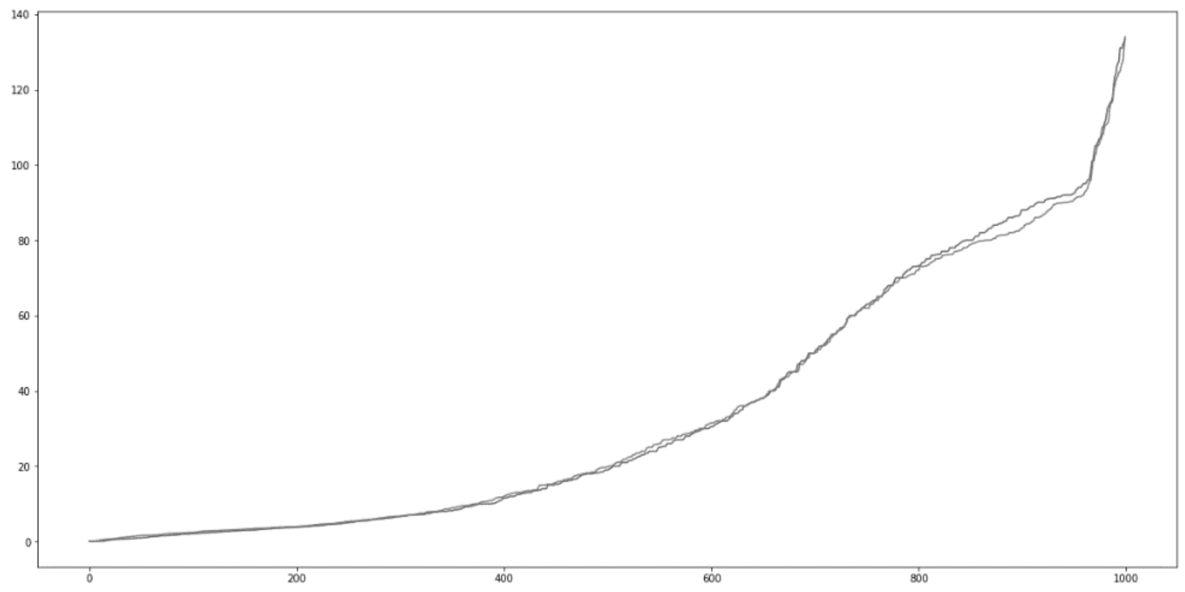


Fig 6.1 Plot of actual and predicted values

The two curves are almost overlapped suggesting that the model is performing well by predicting values near to actual values.

6.3 Comparison with Benchmark

To look at the final model in comparison to the benchmark model, here is the code and the plot,

```
train_time = ( 0.04 , 0.49 )
pred_time = ( 0.03 , 0.17 )
train_score = ( 0.3 , 0.98 )
test_score = ( 0.3 , 0.98 )

fig, ax = plt.subplots(figsize=(10,8))

index = np.arange(2)
bar_width = 0.2

rects1 = ax.bar(index, train_time, bar_width, alpha=0.4, color='b', label='train_time')

rects2 = ax.bar(index + bar_width, pred_time, bar_width, alpha=0.4, color='r',
label='pred_time')

rects3 = ax.bar(index+ 2*bar_width, train_score, bar_width, alpha=0.4, color='y',
label='train_score')

rects4 = ax.bar(index + 3*bar_width, test_score, bar_width, alpha=0.4, color='g',
label='test_score')

ax.set_title('comparision of different models')
ax.set_xticks(index+bar_width*1.5)
ax.set_xticklabels(('Linear Regression', 'Adaboost'))
ax.legend()
```

```
fig.tight_layout()
plt.show()
```

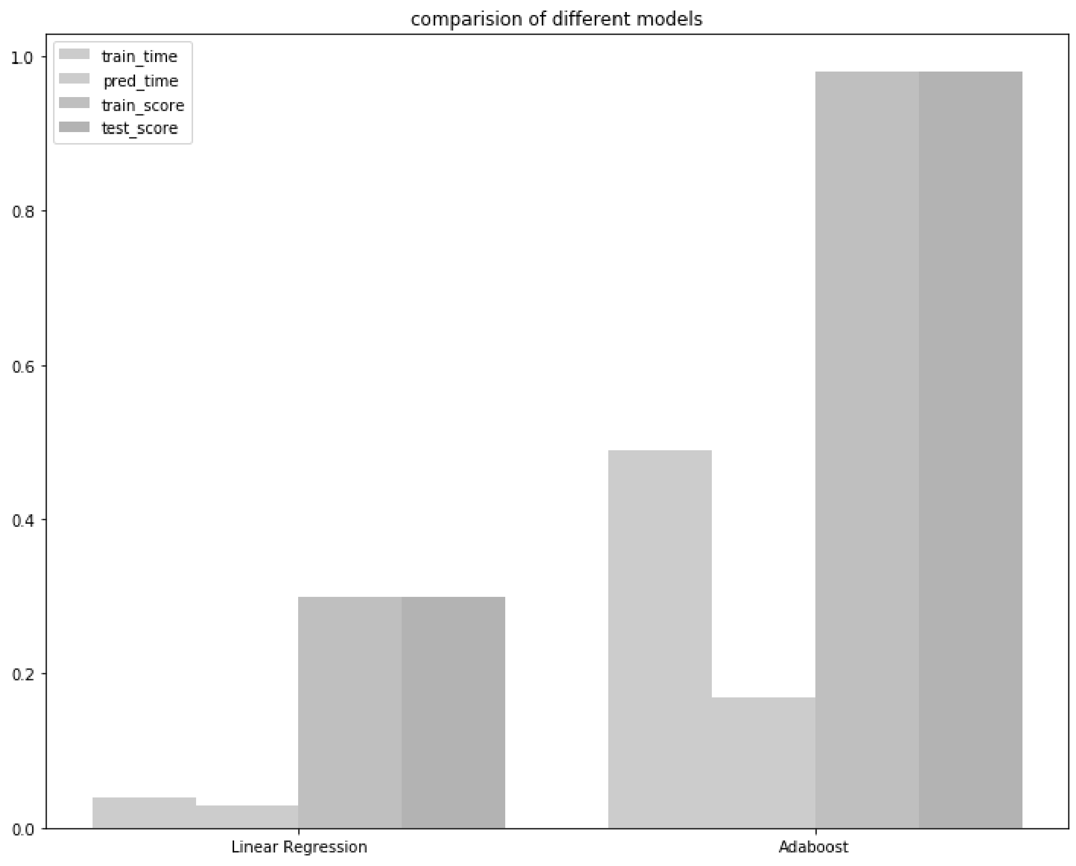


Fig 6.2 Comparison with benchmark model

With 98% variance explained the final model reaches the goal of precisely estimating the critical temperature of the superconductor.

Chapter 7

7.1 Conclusion

The steps involved in this project can be summarized as,

- A problem is defined, and a relevant data set is gathered.
- The data is preprocessed and gone through PCA for dimensionality reduction.
- A benchmark regression model is defined.
- Some regression models were trained and tested
- They are evaluated using a metric and determined which one of them is best.
- Best final model is chosen, and the results are observed.

The boosted decision tree turns out to be the best model, one of the reasons for this could be that the output variable is analogous to step wise data and decision trees tend to work well for such values.

A PCA-PSO-ADA model for predicting TC from structural and correlative electronic parameters of high-TC superconductors. Adaboost was adopted to deal with the dataset, which was a small sample set, and the PSO (Grid search) algorithm was utilized to search for its optimal parameters to achieve a good performance. The PCA was employed to reduce dimensions and interdependencies between the parameters, and the selected optimal dimensions of the parameters were subsequently utilized in PSO-Adaboost to train and validate the regression model. In addition, we also trained a PSO-ADA model without the PCA, with the dataset used for comparison. According to the assessment results and comparison, the PCA-PSO-ADA model provided a better accuracy of prediction than the other models for the dataset. At last, additional data was used to validate the prediction, and the results were also reasonable. In a word, machine-learning methods can be applied to some domains of materials and the PCA-PSO-ADA ensemble method may be used to predict the TC of new high-TC superconductors.

The final the adaboost model estimates the critical temperature of the super conductor with 98% explained variance, which mean it would give a very good estimate.

7.2 Objects justification

Working with many regression models was interesting. But the most interesting thing is that the boosted decision tree model turns out to be the best model for this regression task. Despite having more powerful models like, Random Forest and XG boosting, a single tree with adaboost out classed them by performing as good as those models and takes less training time than them. One reason might be that a single decision tree is good enough for this task and introducing bagging methods like Random forest is increasing the complexity of the model but not improving its quality.

7.3 Future enhancement

- To further improve the predictive power of the models, another set of features can be constructed based on crystallographic and electronic information
- By building a model which can extract the required features for our model from different combinations of elements it might be possible to find new superconductors
- Also instead of using the PCA to reduce the dimensionality, when a powerful configuration of firmware is available, all the features can be used to train the estimator.

Chapter 8

References/Bibliography

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. Bergerhoff, G., Hundt, R., Sievers, R. & Brown, I. D. The inorganic crystal structure data base. J. Chem. Inf. Comput. Sci. 23, 66–69 (1983).
5. Agrawal, A. & Choudhary, A. Perspective: materials informatics and big data: realization of the ―fourth paradigm‖ of science in materials science. APL Mater. 4, 053208 (2016).
6. 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).
7. Curtarolo, S. et al. AFLOWLIB.ORG: a distributed materials properties repository from high-throughput ab initio calculations. Comput. Mater. Sci. 58, 227–235 (2012).
8. Villars, P. & Phillips, J. C. Quantum structural diagrams and high-Tc superconductivity. Phys. Rev. B 37, 2345–2348 (1988).
9. Ling J., Hutchinson M., Antono E., Paradiso S., and Meredig B. High-dimensional materials and process optimization using data-driven experimental design with well-calibrated uncertainty estimates. Integr. Mater. Manuf. Innov. 6, 207–217 (2017).
10. Hirsch, J. E. Correlations between normal-state properties and superconductivity. Phys. Rev. B 55, 9007–9024 (1997).
11. Ward, L., Agrawal, A., Choudhary, A. & Wolverton, C. A general-purpose machine learning framework for predicting properties of inorganic materials. NPJ Comput. Mater. 2, 16028 (2016).
12. Yang, K., Oses, C. & Curtarolo, S. Modeling off-stoichiometry materials with a high-throughput ab-initio approach. Chem. Mater.28, 6484–6492 (2016).
13. Bishop, C. Pattern Recognition and Machine Learning. (Springer-Verlag, NY, 2006).
14. Breiman, L. Random forests. Mach. Learn. 45, 5–32 (2001).

15. Caruana, R. & Niculescu-Mizil, A. An Empirical Comparison of Supervised Learning Algorithms. In Proceedings of the 23rd International Conference on Machine Learning, ICML ’06, 161–168 (ACM, New York, NY, 2006).
16. McKinney, W. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython (O’Reilly Media, 2012).
17. Pedregosa, F. et al. Scikit-learn: Machine Learning in Python. J. Mach. Learn. Res. 12, 2825–2830 (2011).

Chapter 9
Paper Publication

Regression of Superconducting Critical Temperature: using a PCA-GridSearch-Adaboost Regression Model

Naresh Aketi
Department of Computer Science and Engineering, JNTUACEP, India
pandu5188@gmail.com

Harsha Praneeth Dussa
Department of Computer Science and Engineering, JNTUACEP, India
harshapraneeth10@gmail.com

Suresh Parachuri
Department of Computer Science and Engineering, JNTUACEP, India
psuresh0597@gmail.com

Hanmanthu Uppara
Department of Computer Science and Engineering, JNTUACEP, India
Hanumanth557chandu@gmail.com

Abstract -

Superconductivity is being studied since its discovery more than a century ago. The numerous applications of the superconductors made it a subject of intense research. Despite being studied for so long, some of its properties remain a mystery. One of the interesting properties of a super conductor is its critical temperature. Superconductors exhibit zero electrical resistance when maintained at the critical temperature. The value of critical temperature is different for each superconducting material. This value is experimentally calculated by measuring resistance against the temperature of the material. In this project, by taking advantage of the immense increase of readily accessible and potentially relevant information, we develop several machine learning methods modeling critical temperature of a super conductor based on its chemical properties. The final model will give an estimate of critical temperature of a superconductor. This estimate provides confidence on a newly discovered material to continue further research on it.

Keywords – Superconductivity, Critical temperature, Adaboost regression, Machine learning, Decision tree regression, Supervised learning model.

I. Introduction

As important functional materials, high-transition temperature (high-TC) superconductors have some typical physical parameters, such as transition temperature Tc, magnetic susceptibility and critical current density (Jc), which make them very useful in many practical applications like magnetically levitated trains and power transmission. Previous researches showed that the high-Tc superconductors are generally characterized by a two-dimensional layered superconducting condensate with unique features that are not traditional superconducting metals. Their important property, Tc, is determined by their layered crystals, bond lengths, valency properties of the ions, and Coulomb coupling between electronic bands in adjacent, spatially separated layers.

It is clear that Tc (critical temperature) of superconductors depend on its other chemical properties. In this project we utilize the already available data about superconductors to estimate the critical temperature of new potential materials. The developed model can be used to gain confidence on a new material to conduct further research on it regarding its superconducting behavior. Also, experimental determination of critical temperature is a laborious process which is made easy when an estimate of the value is provided by our model.

II. Proposed Work

The goal is to create a regression model to predict the critical temperature of a super conductor. Tasks involved in the project implementation are as follows:

- Gathering, analyzing and preprocessing the data (data exploration)
- Set a benchmark model and evaluation metric
- Training different regression models
- Tuning the final model

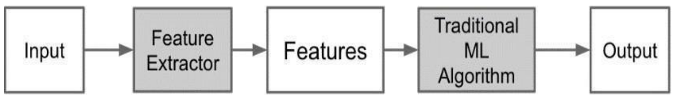


Figure 2.1: Machine learning work flow

the final model is expected to give an estimate of critical temperature of a super conductor based on its chemical properties. And this is the architecture of a machine learning model,

A. Gathering and preprocessing data

The data set is taken from the UCI data repository at:

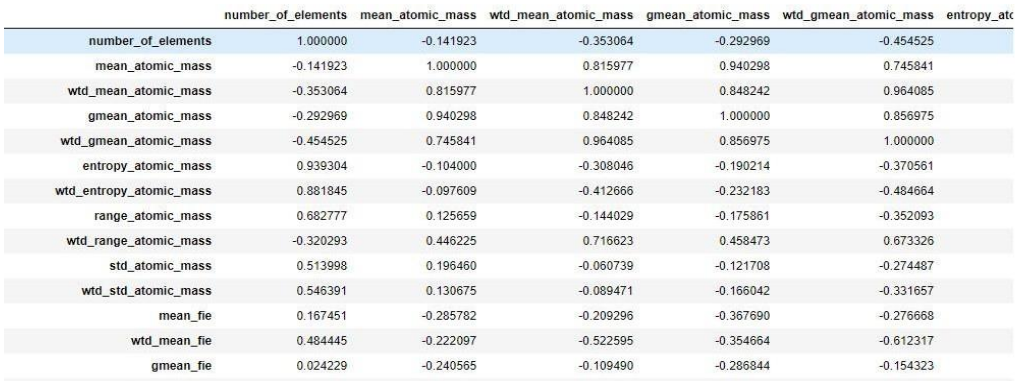
https://archive.ics.uci.edu/ml/datasets/Superconductivty+Data

The characteristics of the dataset are:

Data Set Characteristics:	Multivariate	Number of Instances:	21263
Attribute Characteristics:	Real	Number of Attributes:	81
Associated Tasks:	Regression	Missing Values?	N/A

Table 2.1: Characteristics of the dataset

Since working with too many features puts so much burden on the learning model, a feature extraction method is employed to see that the number features can be reduced.



	number_of_elements	mean_atomic_mass	wtd_mean_atomic_mass	gmean_atomic_mass	wtd_gmean_atomic_mass	entropy_atc
number_of_elements	1.000000	-0.141923	-0.353064	-0.292969	-0.454525	
mean_atomic_mass	-0.141923	1.000000	0.815977	0.940298	0.745841	
wtd_mean_atomic_mass	-0.353064	0.815977	1.000000	0.848242	0.964085	
gmean_atomic_mass	-0.292969	0.940298	0.848242	1.000000	0.856975	
wtd_gmean_atomic_mass	-0.454525	0.745841	0.964085	0.856975	1.000000	
entropy_atomic_mass	0.939304	-0.104000	-0.308046	-0.190214	-0.370561	
wtd_entropy_atomic_mass	0.881845	-0.097609	-0.412666	-0.232183	-0.484664	
range_atomic_mass	0.682777	0.125659	-0.144029	-0.175861	-0.352093	
wtd_range_atomic_mass	-0.320293	0.446225	0.716623	0.458473	0.673326	
std_atomic_mass	0.513998	0.196460	-0.060739	-0.121708	-0.274487	
wtd_std_atomic_mass	0.546391	0.130675	-0.089471	-0.166042	-0.331657	
mean_fie	0.167451	-0.285782	-0.209296	-0.367690	-0.276668	
wtd_mean_fie	0.484445	-0.222097	-0.522595	-0.354664	-0.612317	
gmean_fie	0.024229	-0.240565	-0.109490	-0.286844	-0.154323	

Table 2.2: Correlation values of the attributes

After doing the PCA the 80 features in the input data are projected to 3 features and the total variance explained by these 3 dimensions is 97.22%. Any new dimensions would not contribute much in terms of explained variance and this is evident by the graph below,

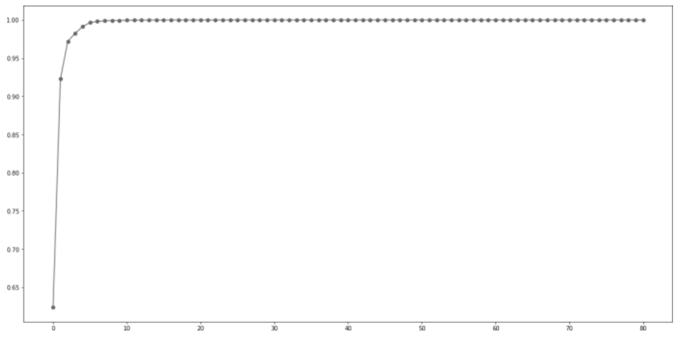
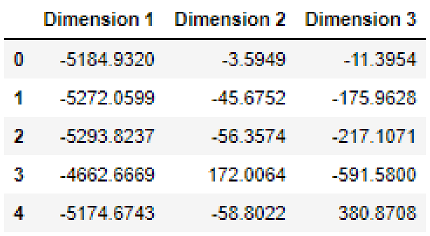


Figure 2.2: Explained variance values against the dimensions of the dataset

The new transformed data after doing principal component analysis looks like,



	Dimension 1	Dimension 2	Dimension 3
0	-5184.9320	-3.5949	-11.3954
1	-5272.0599	-45.6752	-175.9628
2	-5293.8237	-56.3574	-217.1071
3	-4662.6669	172.0064	-591.5800
4	-5174.6743	-58.8022	380.8708

Table 2.3: New transformed data

B. Benchmark model and Evaluation metric

R2_score (Coefficient of determination) is a common metric for a regression model; it is a statistical measure of how well the regression predictions approximate the real data points.

$$R^2 \equiv 1 - \frac{SS_{\mathrm{res}}}{SS_{\mathrm{tot}}}$$

The benchmark r2_score, training time and predicting time are created based on the linear regression model's performance. Linear regression is the simplest regression model. As the Occam's Razor suggests going with the simplest model, it is first checked whether the data follows a linear trend or not.

C. Algorithms

The decision process is taken by choosing the criteria or attribute with highest entropy value and tree is constructed until the threshold of minimum entropy reaches. Finally, the leaf nodes are the classes in this case the estimate of the output variable.

This is how a decision tree works,

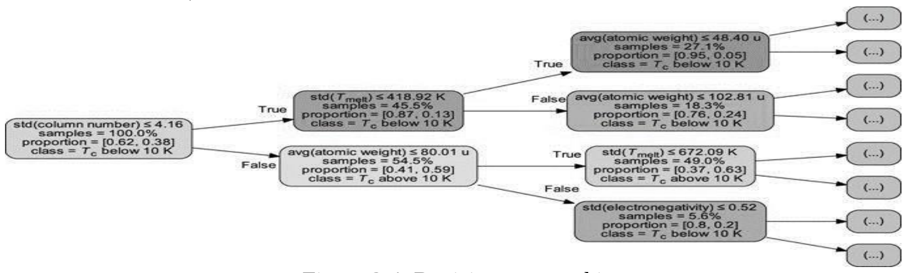

std(column number) ≤ 4.16
samples = 100.0%
proportion = [0.62, 0.38]
class = T_c below 10 K
True
False
std(T_{melt}) ≤ 418.92 K
samples = 45.5%
proportion = [0.87, 0.13]
class = T_c below 10 K
avg(atomic weight) ≤ 80.01 u
samples = 54.5%
proportion = [0.41, 0.59]
class = T_c above 10 K
avg(atomic weight) ≤ 48.40 u
samples = 27.1%
proportion = [0.95, 0.05]
class = T_c below 10 K
avg(atomic weight) ≤ 102.81 u
samples = 18.3%
proportion = [0.76, 0.24]
class = T_c below 10 K
std(T_{melt}) ≤ 672.09 K
samples = 49.0%
proportion = [0.37, 0.63]
class = T_c above 10 K
std(electronegativity) ≤ 0.52
samples = 5.6%
proportion = [0.8, 0.2]
class = T_c below 10 K
(...)

Figure 2.4: Decision tree working

Random forest generates trees randomly and those trees are essentially weak leaners. The algorithm combines the weak learners to produce the end result. This approach of random tree generation can be replaced with a comparatively new and efficient ensemble method called adaboost.

The random forest algorithm's working is explained in the below diagram,

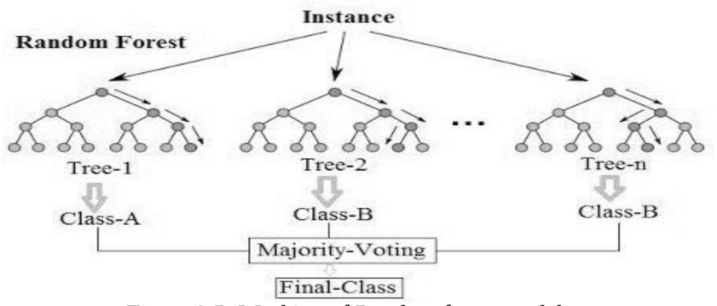


Figure 2.5: Working of Random forest model

Adaboost generates trees which will perform well on the areas its predecessors couldn't. The adaboost model performs better than other primitive models so its performance is compared to other models and chosen to be the best model for this particular regression task.
The working description of an adaboost algorithm is shown below,

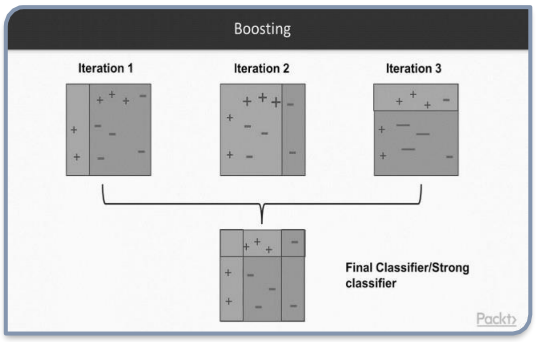


Figure 2.6: Adaboost working

D. Training models

An initial look at the output variable suggests that it mimics stepwise data. Amongst the different regression models the decision tree regression model comes first to mind when dealing with step wise data. So initially a decision tree model is trained to fit the data.

The results after performing the decision tree regression are as follows,

```
DecisionTreeRegressor
{
        'train_time': 0.17719674110412598 seconds,
        'pred_time': 0.00800633430480957 seconds,
        'score_train': 0.7845250110933649,
        'score_test': 0.7835929355566275
}
```

After the training the decision tree model the results were moderate in terms of fitting the data with 78% r2_score on the data. But the model does very well when it comes to training and prediction time. This suggests that a more powerful model can be used for this regression task.

Adaboost generates trees which will perform well on the areas its predecessors couldn't. A Decision tree regressor is passed to the adaboost algorithm to boost it. Then all the estimators are combined to produce the best results and here they are,

```
AdaboostRegressor
{
        'train_time': 1.5947909355163574 seconds,
        'pred_time': 0.20222735404968262 seconds,
        'score_train': 0.9684746229251533,
        'score_test': 0.9671931187916423
}
```

E. Tuning the final model

The final model must be tuned to improve its performance and one way to do so is using a grid search method. In grid search the parameters passed to the model are tuned by using cross validation. The parameters of the adaboost model which are tuned are

- Number of estimators
- Max depth of the tree

The result of the grid search is as follows,

```
AdaBoostRegressor(
base_estimator = DecisionTreeRegressor(criterion='mse', max_depth=25, max_features=None,
max_leaf_nodes = None, min_impurity_decrease = 0.0,
min_impurity_split = None, min_samples_leaf = 1,
min_samples_split = 2, min_weight_fraction_leaf = 0.0,
presort = False, random_state = None, splitter = 'best'),
learning_rate = 1.0, loss = 'linear', n_estimators = 30,
random_state = None) trained.
```

Through grid search the best parameters for the adaboost model are found out to be, Number of estimators is 30 and maximum depth is 25. With this fine tuned model, we can produce the best result possible.

Result Analysis

The final model is evaluated using the R2_score. The input data is split into train and test dataset. The train dataset is used to train the model and then the test dataset is used to see how the model does on previously unseen data.

A. Models summary

The summary of the performance of all models is,

Model name	**R2_score**
Linear regression	30%
Decision tree regression	78%
Random forest regression	92%
Adaboost regression	96%

Table 3.1: models summary

Since, the data doesn't have a linear trend, it is expected that the linear regression model doesn't perform well. So, it ended with 30% r2_score. The data is in a stepwise trend, so the decision tree model does pretty well with a 78% r2_score. The improvement for a simple decision tree model is an ensemble method. Random forest is introduced to improve the r2_score of the decision tree model. With the random forest model, the r2_score is increased to 92%. The final model choice would be Adaboost which is also a ensemble method. The best it could do is 96%.

B. Final model analysis

It is clear that the adaboost outclasses other models in terms of r2_score. But this adaboost is further improved through grid search the final improved model's performance is noted as follows,

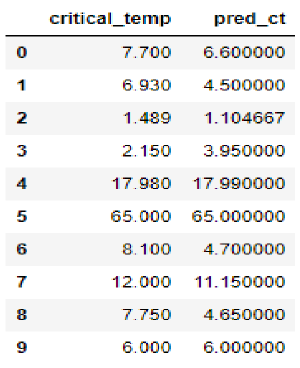
```
train_time: 0.4915473461151123

pred_time: 0.1772010326385498

score_train: 0.982052608960896

score_test: 0.9833312158619923
```

The final model is seen to do well with 98.33% r2_score, that means the final model can explain 98% of variance in the data. To see the final model in work, some actual and predicted values are,



	critical_temp	pred_ct
0	7.700	6.600000
1	6.930	4.500000
2	1.489	1.104667
3	2.150	3.950000
4	17.980	17.990000
5	65.000	65.000000
6	8.100	4.700000
7	12.000	11.150000
8	7.750	4.650000
9	6.000	6.000000

Table 3.2: Comparison of actual and predicted values

The actual and predicted values are quite near, this means the model works well. To take a deeper look a plot is drawn on actual and predicted values.

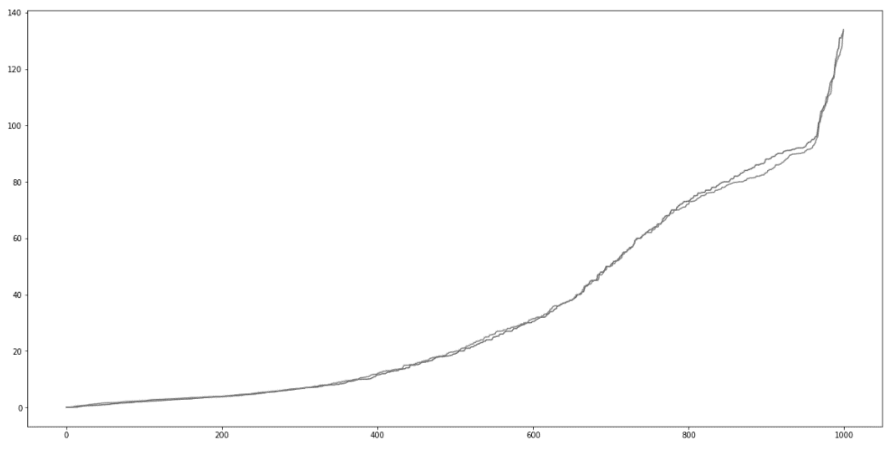


Figure 3.1: Plot of actual and predicted values

The two curves are almost overlapped suggesting that the model is performing well by predicting values near to actual values.

III. Conclusion

A PCA-PSO-ADA model for predicting TC from structural and correlative electronic parameters of high-TC superconductors. Adaboost was adopted to deal with the dataset, which was a small sample set, and the PSO (Grid search) algorithm was utilized to search for its optimal parameters to achieve a good performance. The PCA was employed to reduce dimensions and interdependencies between the parameters, and the selected optimal dimensions of the parameters were subsequently utilized in PSO-Adaboost to train and validate the regression model. According to the assessment results and comparison, the PCA-PSO-ADA model provided a better accuracy of prediction than the other models for the dataset. At last, additional data was used to validate the prediction, and the results were also reasonable. In a word, machine-learning methods can be applied to some domains of materials and the PCA-PSO-ADA ensemble method may be used to predict the TC of new high-TC superconductors.

The final the adaboost model estimates the critical temperature of the super conductor with 98% explained variance, which mean it would give a very good estimate.

IV. Future enhancement

- To further improve the predictive power of the models, another set of features can be constructed based on crystallographic and electronic information
- By building a model which can extract the required features for our model from different combinations of elements it might be possible to find new superconductors
- Also instead of using the PCA to reduce the dimensionality, when a powerful configuration of firmware is available, all the features can be used to train the estimator.

References

[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.

[4] Bergerhoff, G., Hundt, R., Sievers, R. & Brown, I. D. The inorganic crystal structure data base. J. Chem. Inf. Comput. Sci. 23, 66–69 (1983).

[5] Agrawal, A. & Choudhary, A. Perspective: materials informatics and big data: realization of the "fourth paradigm" of science in materials science. APL Mater. 4, 053208 (2016).

[6] 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).

[7] Curtarolo, S. et al. AFLOWLIB.ORG: a distributed materials properties repository from high-throughput ab initio calculations. Comput. Mater. Sci. 58, 227–235 (2012).

[8] Villars, P. & Phillips, J. C. Quantum structural diagrams and high-Tc superconductivity. Phys. Rev. B 37, 2345– 2348 (1988).

[9] Ling J., Hutchinson M., Antono E., Paradiso S., and Meredig B. High-dimensional materials and process optimization using data-driven experimental design with well-calibrated uncertainty estimates. Integr. Mater. Manuf. Innov. 6, 207–217 (2017).

[10] Hirsch, J. E. Correlations between normal-state properties and superconductivity. Phys. Rev. B 55,9007–9024.

[11] Ward, L., Agrawal, A., Choudhary, A. & Wolverton, C. A general-purpose machine learning framework for predicting properties of inorganic materials. NPJ Comput. Mater. 2, 16028 (2016).

[12] Yang, K., Oses, C. & Curtarolo, S. Modeling off-stoichiometry materials with a high-throughput ab-initio approach. Chem. Mater.28, 6484–6492 (2016).

International Journal of Innovative Research in Advanced Engineering
A MONTHLY JOURNAL OF ADVANCED ENGINEERING

ISSN: 2349-2163

Impact Factor Value: Citefactor **2017:1.91** |**Mendeley (Elsevier Indexed)** IndexCopernicus **2016** = 64.35|
Google Scholar Indexed, DOI

Ref: No: IJIRAE/RS/Vol.06/Issue04/APAE10087 Date: 26/04/2019

CERTIFICATE OF ACCEPTANCE LETTER

Certified that research paper titled "REGRESSION OF SUPERCONDUCTING CRITICAL TEMPERATURE: USING A PCA-GRID SEARCH-ADABOOST REGRESSION MODEL" AUTHOR Department of Computer Science and Engineering, JNTUACEP,India, has been accepted to publish in the **International Journal of Innovative Research in Advanced Engineering Volume VI, Issue IV of April 2019**

AFTER REGISTRATION PROCESS IS OVER ✓ITHENTICATE PLAGIARISM REPORT TO BE SEND ALONG WITH CROSS REF - DOI: 10.26562/IJIRAE.2017/

6 009800 461091 >

[illegible] [illegible] [illegible]

International Reviewer National Reviewer Chief Editor - IJIRAE

www.ijirae.com

Email: editor.ampublications@gmail.com, editor.ijirae@gmail.com , editor@ijirae.com , admin@ijirae.com , submit@ijirae.com,

IJIRAE Indexed Organization

http://www.citefactor.org/impact-factor/impact-factor-of-journal-INTERNATIONAL-JOURNAL-OF-INNOVATIVE-RESEARCH-IN-ADVANCED-ENGINEERING.php
https://scholar.google.co.in/citations?hl=en&user=XIGEjOkAAAAJ&view_op=list_works&sortby=pubdate
https://vtu.academia.edu/IJIRAEInternationalJournalofInnovativeResearchinAdvancedEngineering
https://www.scribd.com/user/258778951/IJIRAE-International-Journal-of-Innovative-Research-in-Advanced-Engineering
http://www.bibsonomy.org/user/ijirae
https://www.mendeley.com/profiles/ijirae-journals/ (Mendeley- Elsevier Indexed)
http://www.citeulike.org/user/IJIRAE
https://www.mendeley.com/profiles/ijirae-journals/
https://www.worldcat.org/profiles/ijirae_journal/lists
http://www.publishresearch.com/search
http://isindexing.com/isi/searchedpapers.php
http://www.slideshare.net/IJIRAE
http://www.citefactor.org/myarticles
https://www.linkedin.com/messaging/compose?trk=hb-messages-hdr-item-msg-v2
https://figshare.com/authors/IJIRAE_Journals/1390133
https://www.facebook.com/profile.php?id=100006662446035
https://twitter.com/editorijirae
https://in.pinterest.com/editorijirae/boards/