PERSONAL PROGRAMMING PROJECT REPORT

A data-driven machine learning approach to predicting stacking faulting energy in austenitic steels

By K. Ramya Sri (Mat Nu: 65061)

Professor Incharge:
Dr.-Ing.Arun Prakash and Dr.-Ing. Stefan Prüger

WINTER SEMESTER 2021/22

Contents

1 Introduction **1**

2 Theory **2**
2.1 Data preprocessing 2
2.2 Dividing the data into three classes based on SFE regime . . . 3
2.3 Dimensionality reduction 4
2.3.1 Principal Component Analysis 4
2.3.2 Multidimensional Scaling 5
2.4 Classification Methods 5
2.4.1 Random Forest 5
2.4.2 Neural Networks 8
2.4.3 Confusion Matrix 11

3 Implementation **12**
3.1 Data Pre-processing 12
3.2 Visualization 13
3.3 Dimensionality Reduction 13
3.3.1 Principal Component Analysis 13
3.3.2 Multidimensional Scaling 14
3.4 Classification methods 14
3.4.1 Decision Tree 14
3.4.2 Random Forest 16
3.4.3 Optimizers 17
3.4.4 Layers 18
3.4.5 Neural Network 20
3.4.6 Confusion Matrix metrics 21
3.5 Results 21
3.5.1 Data exploration 21
3.5.2 Dimensionality reduction and visualization 25
3.5.3 Classification 25
3.5.4 Validation and Testing 28
3.5.5 Conclusions 28

4 Git log **28**

1 Introduction

Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed by Arthur Samuel (1959). it can be also said as Well-posed Learning Problem: A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E by Tom Mitchell (1998).

Machine learning is growing rapidly in the field of data science. It is a is a branch of artificial intelligence (AI) and computer science uses data and statistical methods in which algorithms are trained to make classifications/ predictions, to imitate the way that humans learn, gradually improving its accuracy. Machine learning is becoming more ubiquitous every day with greater access to data and computation power and will soon be integrated into many facets of human life.

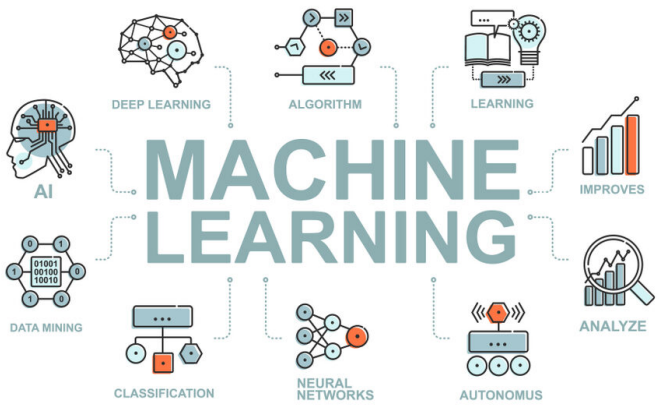

Figure 1: Machine Learning model[1]

In this project we are trying to find out relationship between composition and SFE in austenitic steels which is crucial to alloy design which dictates secondary deformation regime in steels. The machine learning methods are used to study the data available from experimental methods and find out the secondary deformation regimes. After this predicted model is used to predict SFE and consequently deformation regimes for untested compositions in austenitic steels [18].

This is done by:

1. Data preprocessing.
2. Dividing the data into three classes based on SFE regime.
3. Dimensionality reduction for Visualization. The following two methods are used
   (a) Principal Component Analysis
   (b) Multidimensional Scaling
4. After the observation, the data is non-linear. so we are using Classification methods.
   (a) Random Forests
   (b) Artificial Neural Networks

2 Theory

2.1 Data preprocessing

Data preprocessing is the process of transforming raw data into an understandable format. It is also an important step in data mining as we cannot work with raw data. The quality of the data should be checked before applying machine learning or data mining algorithms.

Figure 2: data-preprocessing Tasks[9]

Tasks in Data Preprocessing:

1. Data cleaning: Data cleaning is the process to remove incorrect data, incomplete data and inaccurate data from the datasets, and it also replaces the missing values. There are some techniques in data cleaning. Example, Noisy data

2. Data integration: The process of combining multiple sources into a single dataset. The Data integration process is one of the main components in data management. There are some problems to be considered during data integration. Example, Schema integration

3. Data reduction: This process helps in the reduction of the volume of the data which makes the analysis easier yet produces the same or almost the same result. This reduction also helps to reduce storage space. Example, Dimensionality reduction.

4. Data Transformation: The change made in the format or the structure of the data is called data transformation. This step can be simple or complex based on the requirements. Example, Normalization.[9]

2.2 Dividing the data into three classes based on SFE regime

An Austenitic steel with SFE value below 20 mJ/m2 deforms by martensitic transformation of TRIP-like behavior, with SFE value in between 20 and 45 mJ/m2 deforms primarily by deformation twinning leading to a TWIP-like behavior while SFE values above 45 mJ/m2 deforms majorly by slip. These regimes can be termed as Low, Medium and High,respectively[18].

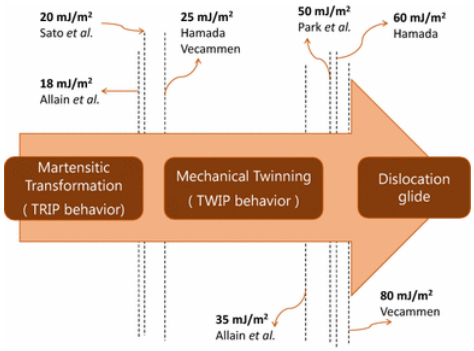

Figure 3: SFE regimes[18]

2.3 Dimensionality reduction

Converting the data from high-dimensional space into a low-dimensional space, which makes data easier to visualize. The data is 10D which is reduced inrto 2D/3D.

2.3.1 Principal Component Analysis

Using the method we are transforming the data to comparable scales. It is done by using the following steps:

1. Standardization
$$z = \frac{\text{value} - \text{mean}}{\text{standard deviation}} \tag{2.1}$$

2. Computation of Covariance Matrix:

   The covariance matrix is a m × m symmetric matrix (where p is the number of dimensions) that has as entries the covariances associated with all possible pairs of the initial variables. It is symmetric with respect to the main diagonal, which means that the upper and the lower triangular portions are equal. Example,

$$\begin{bmatrix} \text{Cov}(x,x) & \text{Cov}(x,y) & \text{Cov}(x,z) \\ \text{Cov}(y,x) & \text{Cov}(y,y) & \text{Cov}(y,z) \\ \text{Cov}(z,x) & \text{Cov}(z,y) & \text{Cov}(z,z) \end{bmatrix} \tag{2.2}$$

   The sign of the covariance matrix is important here. It summaries the correlations between all the possible pairs of variables.

3. Eigenvectors and Eigenvalues:

   Eigenvectors and eigenvalues from the covariance matrix, which are later arranged in ordering descending order are used to determine the principal components of the given data.

$$Av = \lambda v \tag{2.3}$$

$$|A - \lambda I| = 0 \tag{2.4}$$

4. Feature Vector:

   Feature vector is simply a matrix that has as columns the eigenvectors of the components that we decide to keep. This makes it the first step towards dimensionality reduction, because if we choose to keep only p eigenvectors (components) out of n, the final data set will have only p dimensions[15].

5. Recast the data along the principal components axes:

$$\text{FinalDataSet} = \text{FeatureVector}^{T} * \text{StandardizedOriginalDataSet}^{T} \tag{2.5}$$

2.3.2 Multidimensional Scaling

Multidimensional Scaling is a family of statistical methods that focus on creating mappings of items based on distance.it is a distance-preserving manifold learning method. All manifold learning algorithms assume the dataset lies on a smooth, non linear manifold of low dimension and that a mapping f: RD $->$ Rd (D>>d) can be found by preserving one or more properties of the higher dimension space. Distance preserving methods assume that a manifold can be defined by the pairwise distances of its points. In distance preserving methods, a low dimensional embedding is obtained from the higher dimension in such a way that pairwise distances between the points remain same. Some distance preserving methods preserve spatial distances(MDS) while some preserve graph distances[19].

MDS takes a dissimilarity matrix D where Dij represents the dissimilarity between points i and j and produces a mapping on a lower dimension, preserving the dissimilarities as closely as possible. The dissimilarity matrix could be observed or calculated from the given dataset[19].

2.4 Classification Methods

2.4.1 Random Forest

Random forest is a supervised learning algorithm. The random forest is a Machine Learning classification algorithm consisting of many decisions trees. It combines the output of multiple decision trees to reach a single result.

Random forest algorithms have three main hyper parameters, which need to be set before training. These include node size, the number of trees, and the number of features sampled[13]. Random forest classifier can be used to solve for regression or classification problems. The "forest" it builds, is an ensemble of decision trees, usually trained with the bagging method. The general idea of the bagging method is that a combination of learning models increases the overall result[12].

Decision Trees: The goal of using a Decision Tree is to create a training model that can use to predict the class or value of the target variable by learning simple decision rules inferred from prior data(training data). In Decision Trees, for predicting a class label for a record we start from the root

of the tree. We compare the values of the root attribute with the records attribute. On the basis of comparison, we follow the branch corresponding to that value and jump to the next node[11][14].

1. Root Node: It represents the entire population or sample and this further gets divided into two or more homogeneous sets.
2. Splitting: It is a process of dividing a node into two or more sub-nodes.
3. Decision Node: When a sub-node splits into further sub-nodes, then it is called the decision node.
4. Leaf / Terminal Node: Nodes do not split is called Leaf or Terminal node.
5. Pruning: When we remove sub-nodes of a decision node, this process is called pruning. You can say the opposite process of splitting.
6. Branch / Sub-Tree: A subsection of the entire tree is called branch or sub-tree.
7. Parent and Child Node: A node, which is divided into sub-nodes is called a parent node of sub-nodes whereas sub-nodes are the child of a parent node.

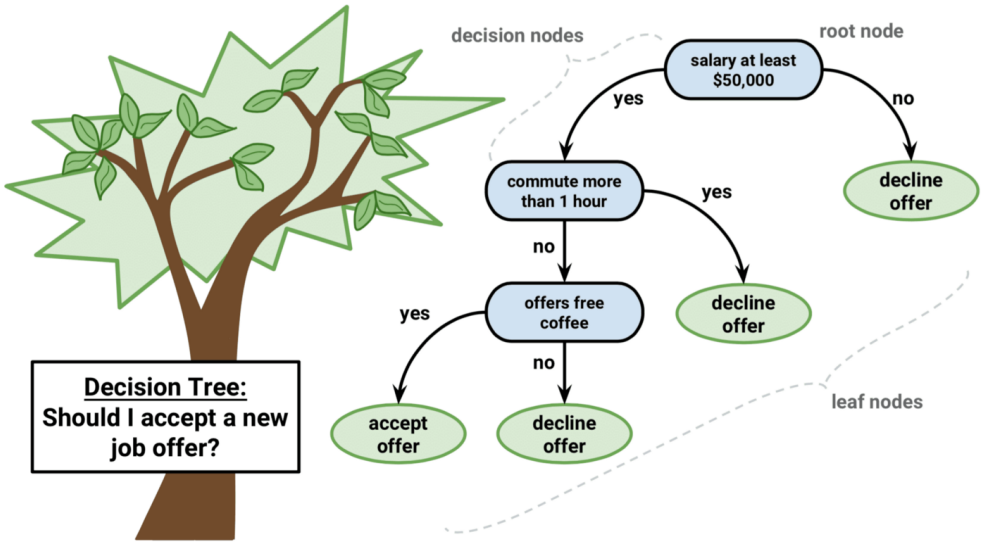

Figure 4: Decision Tree[6]

In the beginning, the whole training set is considered as the root. Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model. Records are distributed recursively on the basis of attribute values. Order to placing attributes as root or internal node of the tree is done by using some statistical approach[11].

Steps performed:

1. It begins with the original set S as the root node.
2. On each iteration of the algorithm, it iterates through the very unused attribute of the set S and calculates Entropy(H) and Information gain(IG) of this attribute.
3. It then selects the attribute which has the smallest Entropy or Largest Information gain.
4. The set S is then split by the selected attribute to produce a subset of the data.
5. The algorithm continues to recur on each subset, considering only attributes never selected before[11].

Entropy: Entropy is a measure of the randomness in the information being processed. The higher the entropy, the harder it is to draw any conclusions from that information.

Entropy for single attribute:

$$E(S) = \sum_{i=1}^{c} -p_i \log_2 p_i \tag{2.6}$$

Entropy for multiple attributes:

$$E(T, X) = \sum_{c \in X} P(c)E(c) \tag{2.7}$$

S $\rightarrow$ Current state, Pi $\rightarrow$ Probability of an event i of state S or Percentage of class i in a node of state S, T $\rightarrow$ Current state and X $\rightarrow$ Selected attribute[11].

Information gain: Information gain is a decrease in entropy. It computes the difference between entropy before split and average entropy after split of the dataset based on given attribute values[11].

$$\text{Information Gain(T,X)} = \text{Entropy (T)} - \text{Entropy(T, X)} \tag{2.8}$$

Gini impurity: Cost function used to evaluate splits in the dataset.

$$\text{Gini} \ = 1 - \sum_{i=1}^{C} (p_i)^2 \tag{2.9}$$

2.4.2 Neural Networks

Neural networks reflect the behaviour of the human brain, allowing computer programs to recognize patterns and solve common problems in the fields of AI, machine learning, and deep learning[7].

Artificial Neural Networks: Multilayer perceptrons (MLPs) is simple artificial Neural Network. A simple neural network includes an input layer, an output (or target) layer and, in between, a hidden layer. The layers are connected via nodes, and these connections form a "network" – the neural network – of interconnected.

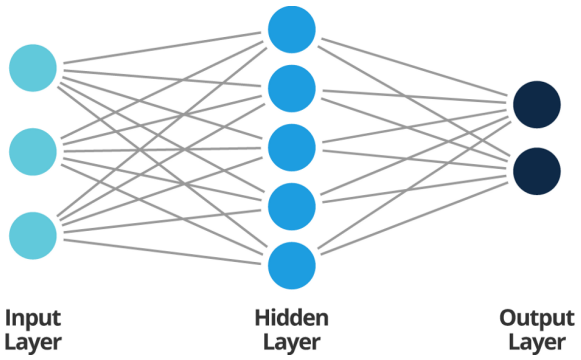

Figure 5: Multilayer perceptrons[8]

Activation functions: An activation function in a neural network defines how the weighted sum of the input is transformed into an output from a node or nodes in a layer of the network.The output layer will typically use a different activation function from the hidden layers and is dependent upon the type of prediction required by the model[4].

Loss Functions: Neural networks are trained using an optimization process that requires a loss function to calculate the model error[5].

Types of loss functions

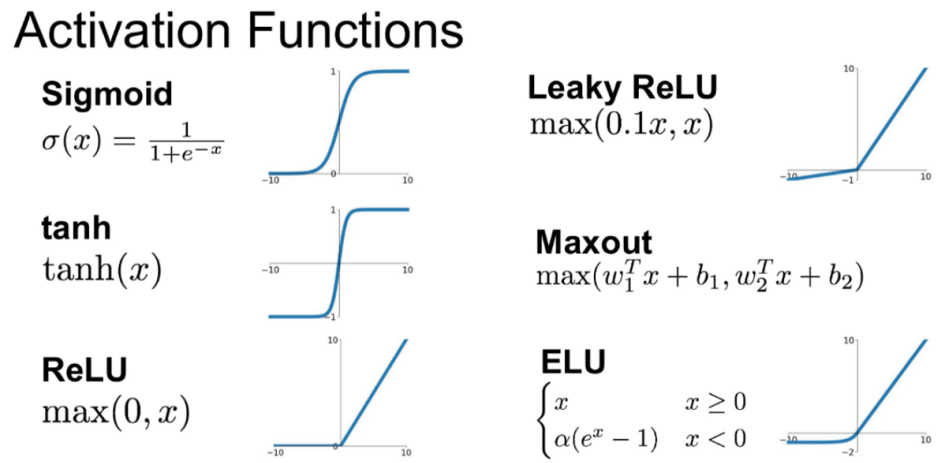

Figure 6: Activation functions[21]

1. For regression Problem we prefer Mean Squared Error (MSE).

$$MSE = \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{n} \tag{2.10}$$

2. For binary/multi-class classification Problem we prefer Cross-Entropy.

$$\text{CrossEntropyLoss } = -\left(y_i \log\left(\hat{y}_i\right) + (1 - y_i) \log\left(1 - \hat{y}_i\right)\right) \tag{2.11}$$

Optimizers: Optimization refers to the process of minimizing the loss function by systematically updating the network weights. Mathematically this is expressed as

$$w' = \text{argmin}\, wL(w) \tag{2.12}$$

L is loss function and w is weights. Intuitively, it can be thought of as descending a high-dimensional landscape. If we could project it in 2D plot, the height of the landscape would be the value of the loss function and the horizontal axis would be the values of our weights w. Ultimately, the goal is to reach the bottom of the landscape by iteratively exploring the space around us[2].

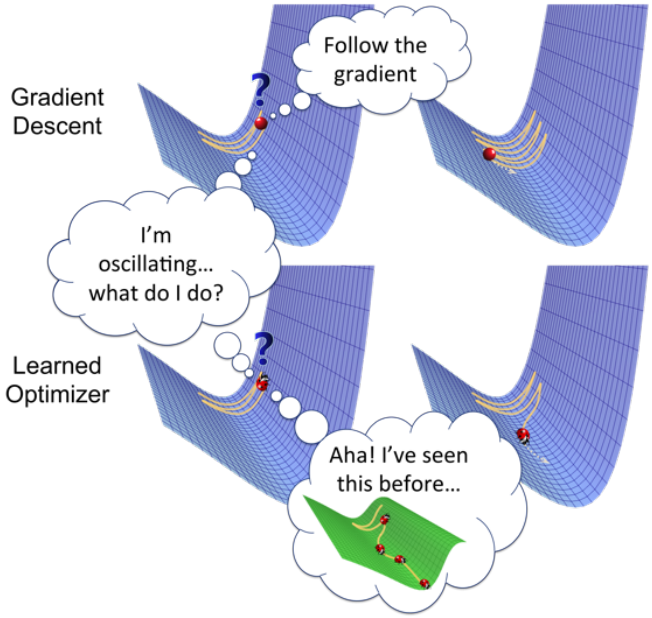

Figure 7: Optimizers help to get results faster[3]

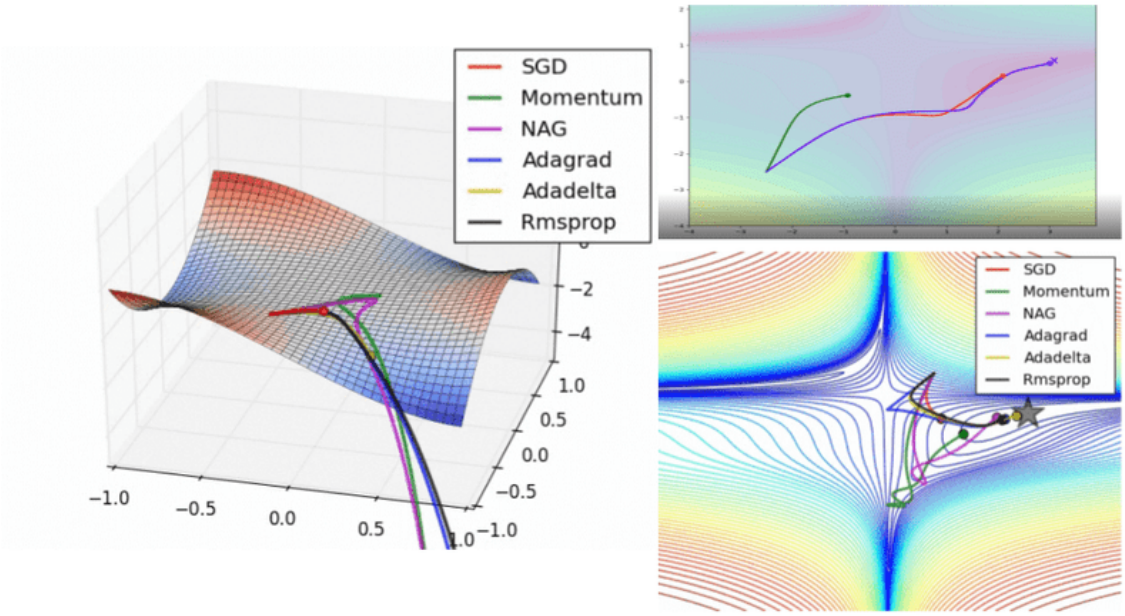

Figure 8: Types of optimizers[2]

2.4.3 Confusion Matrix

A confusion matrix is a matrix (table) that can be used to measure the performance of an machine learning algorithm, usually a supervised learning one. Each row of the confusion matrix represents the instances of an actual class and each column represents the instances of a predicted class[16].

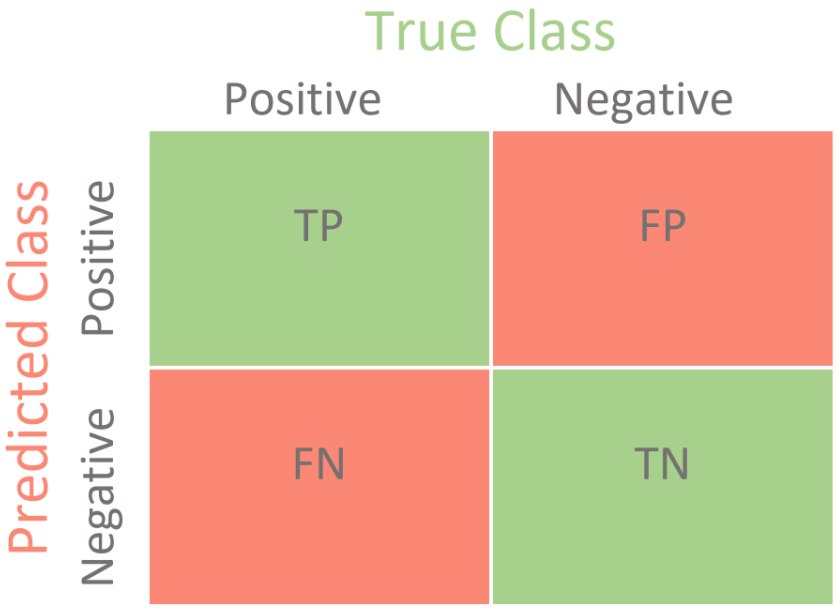

Figure 9: Confusion Matrix[17]

1. true positives (TP): These are cases in which we predicted yes (they are in the particular class), and they do are in given class.

2. true negatives (TN): We predicted no, and they are not in the given class.

3. false positives (FP): We predicted yes, but they are not actually in given class. (Also known as a "Type I error.")

4. false negatives (FN): We predicted no, but they do are in given class. (Also known as a "Type II error.")

We can find the following important terms using Confusion Matrix:

Accuracy: Accuracy simply measures how often the classifier makes the correct prediction. Its the ratio between the number of correct predictions and the total number of predictions.

$$AC = \frac{TN + TP}{TN + FP + FN + TP} \tag{2.13}$$

Precision: predictions are actually positive out of all the total positive predicted.

$$\text{Precision } = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{\text{Predictions Actually Positive}}{\text{Total Predicted positive}} \tag{2.14}$$

Recall: how many observations of positive class are actually predicted as positive

$$\text{Recall } = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{Predictions Actually Positive}}{\text{Total Actual positive}} \tag{2.15}$$

3 Implementation

The Programming part was implemented in python using Jupyter extension Visual Studio Code editor.

3.1 Data Pre-processing

The data used preprocessing is taken from repository provided by the author of research paper[10]. This data is collected from experimental methods, the journal papers from which data is collected has been mentioned in raw data. the data has around 500 data points. we cleared all the noisy and unnecessary data by using pandas library. We removed the alloys which have more zeros values. Now we have final set of data with 374 points and 10 features. Now the sorted data is arranged in classes based on SFE regime. It is a Multi-class Classifier since it has more than two classes.

classes[data <= 20] = 0
classes[data > 20)and($data$ <= 45)] = 1
classes[data > 45)] = 2

3.2 Visualization

The data available is visualized between each element and its SFE, also between two elements and its SFE, and each element with all other elements.

3.3 Dimensionality Reduction

3.3.1 Principal Component Analysis

The overview of implementation of PCA is given below

```
def PCA(data , n_components)

data = we use the data which is normalized
n_components = required dimension as output

 Normalization

 def dataset_MinMax(dataset):
     Min_Max = list()

  "dataset: preprocessed data
   dataset_MinMax: calculates the min and max value
   for each attribute in a dataset.
   Min_val: minimum Value
   Min_val: maximum Value"

   def normalization(dataset, Min_Max):
   "normalization = normalize the dataset to range 0 or 1"

   which iterates over
       value = (value - minimum Value)
                      / (maximum Value - minimum Value)

     Used terms for calculation of
     m:mean
     cm:covariance matrix
     eigenvalues , eigenvectors
     eigenvector_ncom : select the first n
                     eigenvectors, n is desired dimension
     reduced_dim: finally reducing the dimension
```

3.3.2 Multidimensional Scaling

3.4 Classification methods

Before implementing we are splitting the data into training and testing set. 70per is taken as training and remaining 20per is taken as testing data test.

3.4.1 Decision Tree

In this implementation we have root nodes, internal node and leaf nodes

```
class Decision_Node():
    #It is a root node

    def __init__(self, feature_index=None, threshold_val=None,
                 value=None, left_branch=None, right_branch=None):

        self.feature_index = feature_index
        #Feature index helps us to measure threshold.

        self.threshold_val = threshold_val
        #gives access to Decision function output
        which we use to make the prediction.

        self.value = value
        #The prediction, if its classification tree or not.

        self.left_branch = left_branch
        # left branch/subtree/child of a tree

        self.right_branch = right_branch
        # right branch/subtree/child of a tree

class Decision_Tree(object):

    def __init__(self, min_samples=2,
        min_imp=1e-7,maxi_depth=float("inf"), loss_fun=None):

        self.root = None
        # initialize the root of the tree t

        self.min_samples = min_samples
```

```
   #the total number of splits we are using to build tree

   self.min_imp = min_imp
   #The minimum impurity that should be used to
   split the tree further.

   self.maxi_depth = maxi_depth
   #The maximum depth to stop splitting.

   self._impurity_cal = None
   # to decide optimal split required for node

   self._leaf_value_cal = None
   #to predict y al leaf

   self.dimension = None
   # dimension of y(multi-dim)

   self.loss_fun = loss_fun
   #It is a function to calculate impurity
   for Gradient Boosting models.
def fit(self, X, y, loss_fun=None):
   #function to train/build decision tree

def _grow_tree(self, X, y_f, curr_depth=0):

   Max_imp = 0
   BestMeasure = None
    # to represent threshold values and Feature index

   BestSplit = None
    # to represent subtress

def prediction_value(self, x, tree=None):
   # prediction of sample by the value of leaf
   using recursive search

 def predict(self, X):
   #each single sample check and return labels

 def print_tree(self, tree=None, indent=" "):
```

```
        #function to print the tree

class Classification_Tree(Decision_Tree):
 #implementing tree for classification

    def information_gain_cal(self, y_f, y_1, y_2):
        # Calculate information gain

     def calculateEntropy(self, E):
    #function to calculate Entropy
```

3.4.2 Random Forest

we implement random forest based on decision tree, random forest gives better pruning and number of observations.

```
class Random_Forest():
    #We will use the classification_tree that train and
    use on random subsets of data and features.

    def __init__(self, num_estimators=100,
                maxi_features=None,
                min_samples=2,
                 mini_gain=0,
                 maxi_depth=float("inf")):

        self.num_estimators = num_estimators
        # The number of classification trees

        self.maxi_features = maxi_features
        # Maxmimum number of features on classification tree

        self.min_samples = min_samples
        #the total number of splits we are using to build tree

        self.mini_gain = mini_gain
        # Minimum information gain req. to split tree further

        self.maxi_depth = maxi_depth
        # Maxi depth of tree
```

```
        self.progressbar = progressbar.ProgressBar
                        (widgets=bar_custom_widgets)

        bar_custom_widgets = [ "Training Model: ",
                  progressbar.Percentage()," ",
                  progressbar.Bar(marker="-"),
                  ' ', progressbar.ETA()
                  ETA:  estimated time to completion

    # set decision_trees
        self.deci_trees = []
        for _ in range(num_estimators):
            self.deci_trees.append(
                Classification_Tree(
                    min_samples=self.min_samples,
                    min_imp=mini_gain,
                    maxi_depth=self.maxi_depth))

     def fit(self, X, y_f):
     def predict(self, X)
           return y_predi
```

3.4.3 Optimizers

Deep Learning ultimately is about finding a minimum that generalizes well – with bonus points for finding one fast and reliably.stochastic gradient descent (SGD) is used from long. Different optimization algorithms have been proposed in recent years, which use different equations to update a model's parameters. Adam is one of the mostly commonly used algorithm[20].

```
# An optimizer is a function/an algorithm that modifies
the attributes of the neural network, such as weights and
learning rate
# weights: minimizes the loss.
#reduces overall loss and improve the accuracy

class StochasticGradientDescent():
    def __init__(self, learning_rate=0.01, momentum=0):
```

```
        self.learning_rate = learning_rate
        self.momentum = momentum
        self.w_updt = None

class Adam():
    def __init__(self, learning_rate=0.001, b1=0.9, b2=0.999):
        self.learning_rate = learning_rate
        self.eps = 1e-8
        self.m = None
        self.v = None
        # Decay rates
        self.b1 = b1
        self.b2 = b2
```

3.4.4 Layers

```
class Layer(object):

    def set_input_shape(self, shape):
        """ Sets the shape that the layer expects of the
        input in the forward pass method """

        self.input_shape = shape

    def layer_name(self):
        """ The name of the layer. Used in model summary. """
        return self.__class__.__name__

    def parameters(self):
        """ The number of trainable parameters used by the layer """
        return 0

    def forward_pass(self, X, training):
        """ Propogates the signal forward in the network """
        raise NotImplementedError()

    def backward_pass(self, accum_grad):
        """ Propogates the accumulated gradient backwards
        in the network.
        If the has trainable weights then these weights
        are also tuned in this method.
```

```
        As input (accum_grad) it receives the gradient
        with respect to the output of the layer and
        returns the gradient with respect to the
        output of the previous layer. """
        raise NotImplementedError()

    def output_shape(self):
        """ The shape of the output produced by forward_pass """
        raise NotImplementedError()


class Dense(Layer):
    """A fully-connected NN layer.
    Parameters:
    -----------
    n_units: int
        The number of neurons in the layer.
    input_shape: tuple
        The expected input shape of the layer.
        For dense layers a single digit specifying
        the number of features of the input.
        Must be specified if it is the first layer in
        the network.
    """
    def __init__(self, n_units, input_shape=None):
        self.layer_input = None
        self.input_shape = input_shape
        self.n_units = n_units
        self.trainable = True
        self.W = None
        self.w0 = None
class Dropout(Layer):
    """A layer that randomly sets a fraction p of the
    output units of the previous layer
    to zero.
    Parameters:
    -----------
    p: float
        The probability that unit x is set to zero.
    """
    def __init__(self, p=0.2):
```

```
        self.p = p
        self._mask = None
        self.input_shape = None
        self.n_units = None
        self.pass_through = True
        self.trainable = True

activation_functions = {
    'relu': ReLU,
    'sigmoid': Sigmoid,
    'elu': ELU,
    'softmax': Softmax,
    'leaky_relu': LeakyReLU,
    'tanh': TanH,
    'softplus': SoftPlus
}

class Activation(Layer):
    """A layer that applies an activation operation to the input.
    Parameters:
    name: string
        The name of the activation function that will be used.
    """

    def __init__(self, name):
        self.activation_name = name
        self.activation_func = activation_functions[name]()
        self.trainable = True
```

3.4.5 Neural Network

```
class NeuralNetwork():
    """Neural Network. Deep Learning base model.

    Parameters:
    -----------
    optimizer: class
        The weight optimizer that will be used to tune the
        weights in order of minimizing
        the loss.
```

```
    loss: class
        Loss function used to measure the model's
        performance. SquareLoss or CrossEntropy.
    validation: tuple
        A tuple containing validation data and labels (X, y)
    """
    def __init__(self, optimizer, validation_data=None):
        self.optimizer = optimizer
        self.layers = []
        self.errors = {"training": [], "validation": []}
        #self.loss_function = loss()
        #self.loss_fun=loss_fun
        self.progressbar = progressbar.ProgressBar
        (widgets=bar_custom_widgets)

        self.val_set = None
```

3.4.6 Confusion Matrix metrics

The confusion matrix is another metric that is often used to measure the performance of a classification algorithm.

```
#comp_confmat : Confusion Matrix
#actualc= y_test
#predicted=y_predicted
#functions defined in this meterics

def comp_confmat(actual, predicted)
def precision(label, confusion_matrix)
def precision_macro_average(confusion_matrix)
def recall_macro_average(confusion_matrix)
def accuracy(confusion_matrix)
def accuracy_score(y_true, y_pred)
```

3.5 Results

3.5.1 Data exploration

Visualizing data is very important step before processing with other methods because we may find common trends which are useful of analysis. It is usually done by plotting the data along various dimensions and exploring.

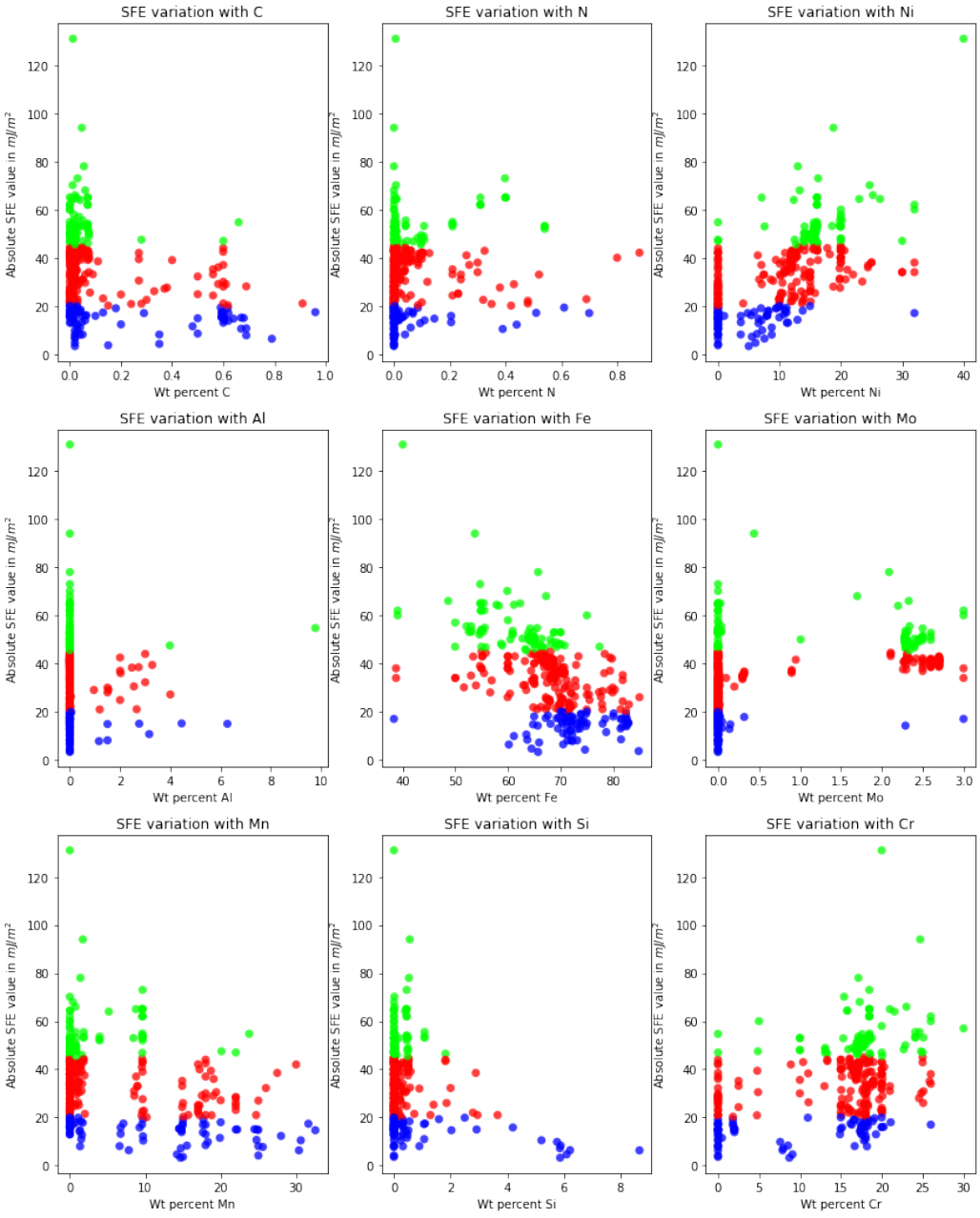

Figure 10: SFE with elements

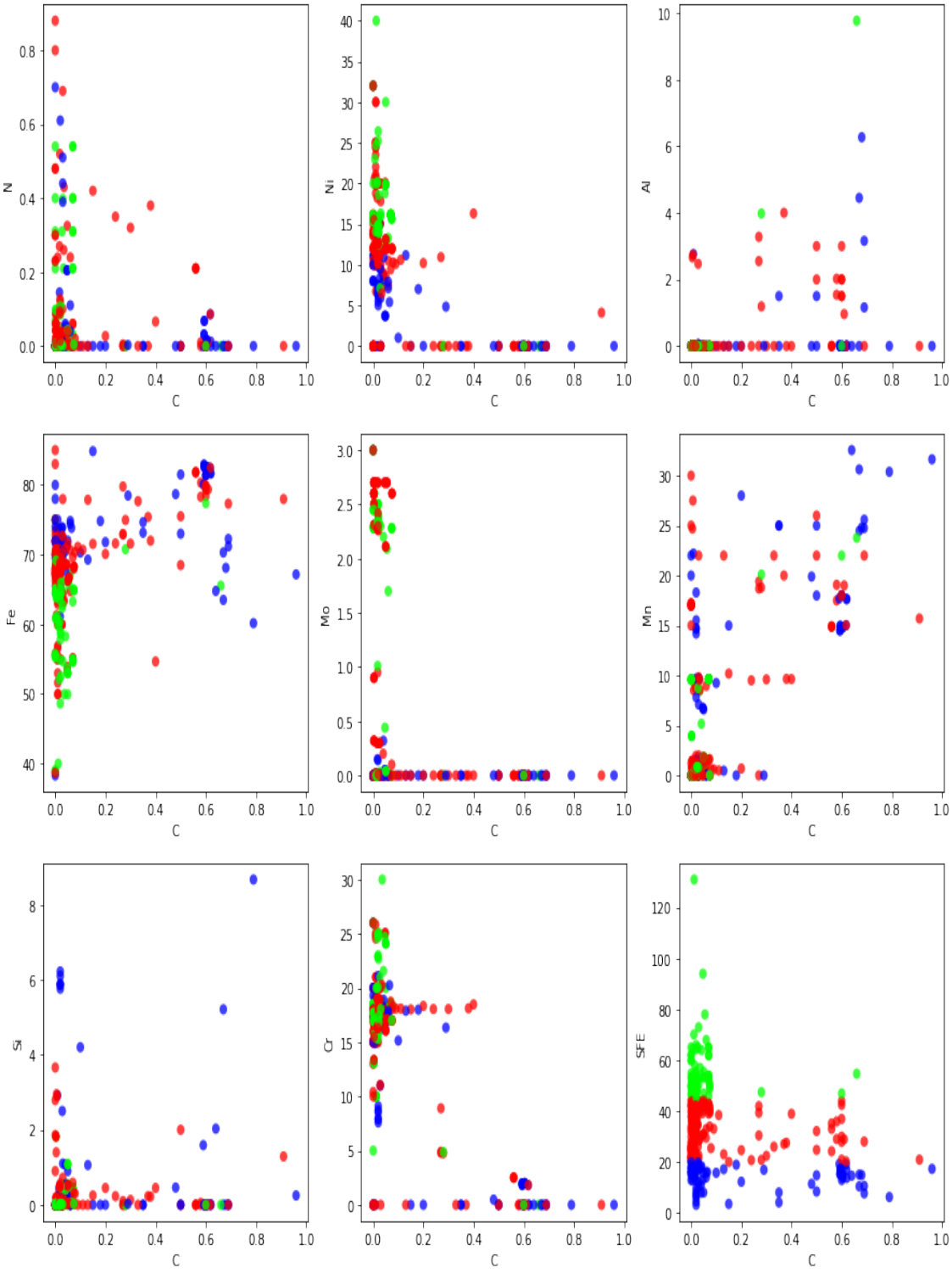

Figure 11: one element with all

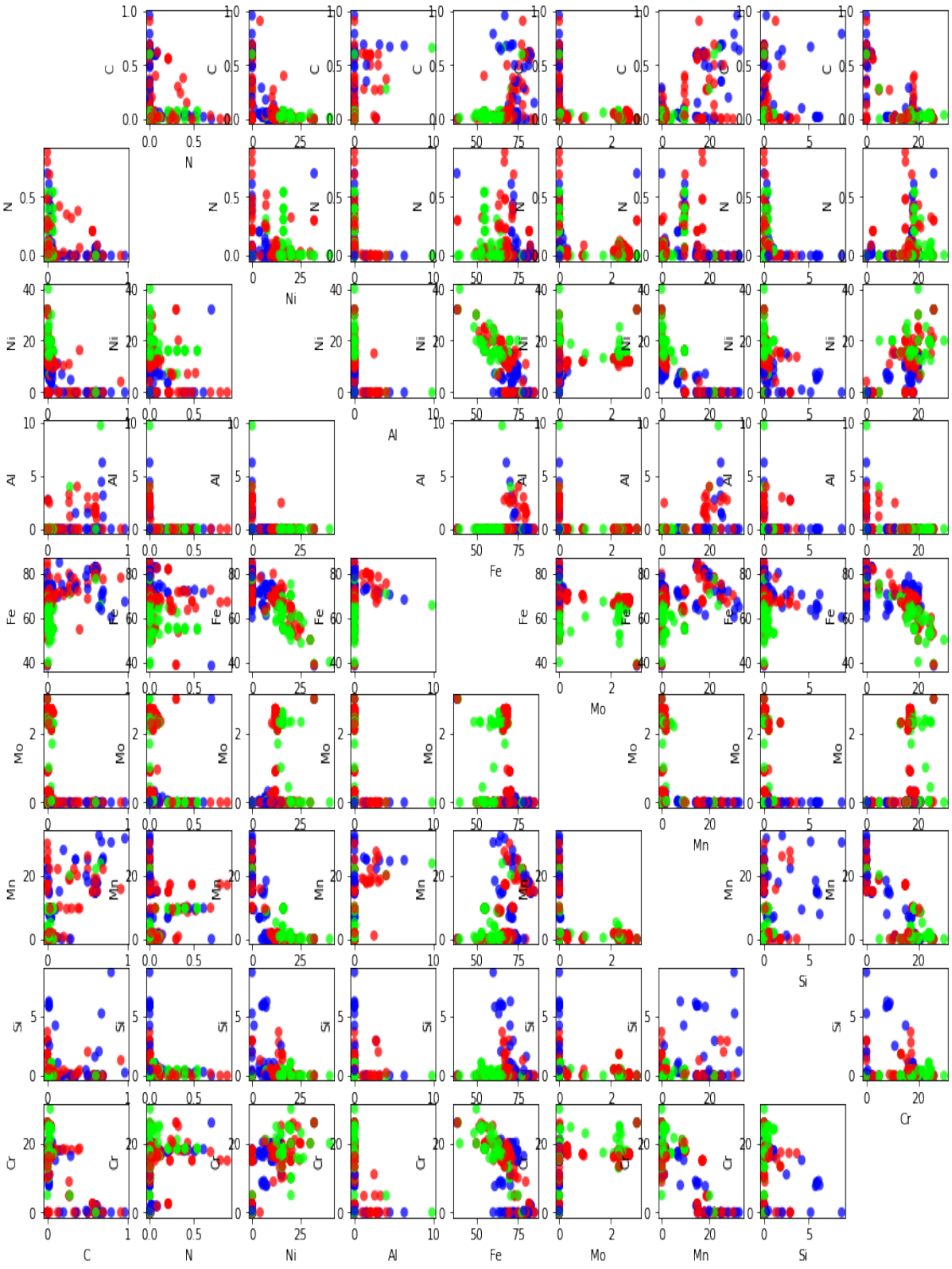

Figure 12: All elements with other elements

Figure 10 shows SFE regime based on all other elements. Figure 11 shows one element wt% based on all other elements wt% . Figure 12 shows all element wt% based on all other elements wt% .

1. As we observe plots we can see that SFE composition-SFE relationship of any element on other elements has no certain pattern clearly saying its non-linear.

2. The SFE plot with elements also shows clearly that there is no linear relationship between any element and SFE.

3.5.2 Dimensionality reduction and visualization

Using data exploration we can only visualize part of the features of the data at once, losing crucial insights about the true nature of SFE dependence on composition. If we could visualize all changing parameters at once and the dependence of SFE on these parameters, we would essentially know some pattern. When the data are higher dimensional, visualization becomes increasingly difficult. So, we used machine learning dimensionality reduction algorithms.

We have used both MDS and PCA methods. The 2d diemsional images of MDS and PCA are very similar.To aid in visualization, Figure 13 represents projections of this 3 components to visualize in 2 dimensions. we can see that for PC1>-0.1 there are no low SFE values and PC1<0.2 there are no low SFE values(excluding two or three may be due error)

The the classes are highly mixed in the composition space and there are no regions or cluster for single classes. we can see the non-linearity and complexity of the data and the use of classification algorithm.

3.5.3 Classification

These algorithms can essentially be thus treated as black boxes which when given an input composition, will output the SFE regime or class that the composition belongs to.

The performance of the above 3 algorithms on the hold-out test set. Each algorithm has an accuracy of about 85% on the validation set. The macro scores of precision and recall are very similar too. We have taken macro scores since all classes are equally important to us and hence no weighting has been provided to better performance over one of the classes.

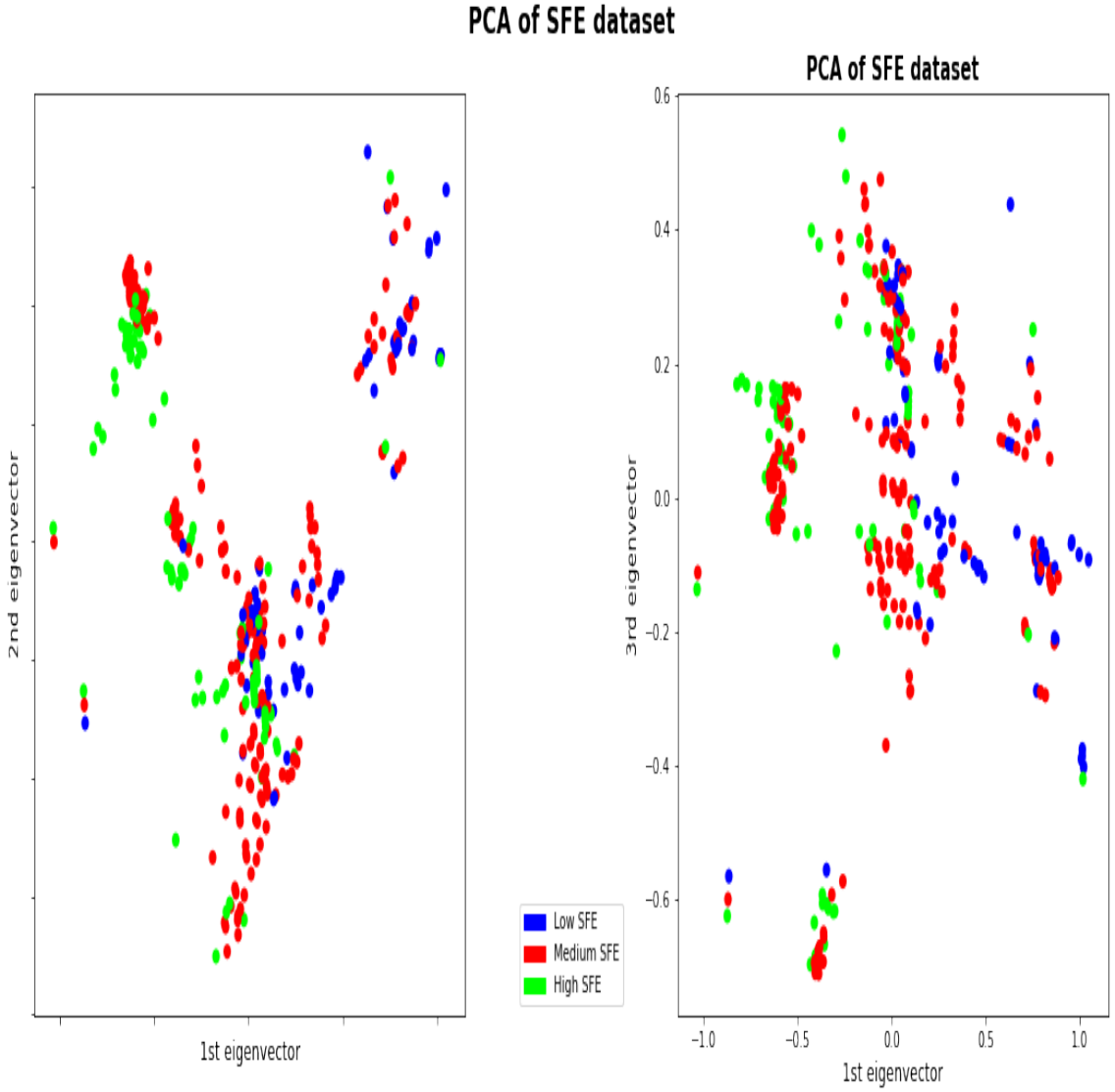

Figure 13: Principal component analysis

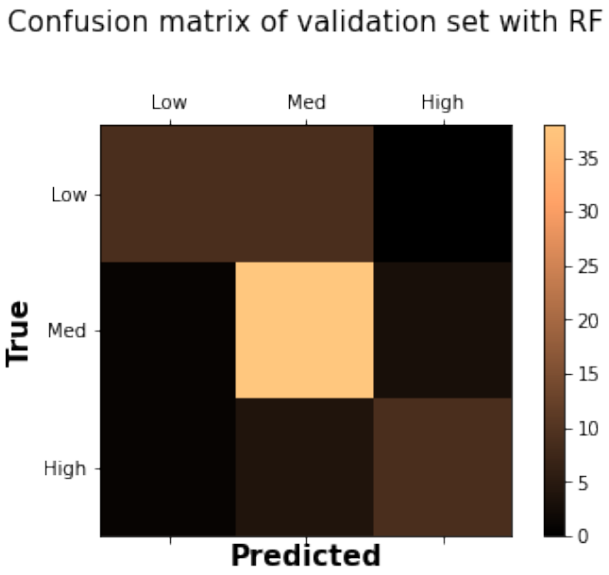

Figure 14: Random Forest Confusion Matrix

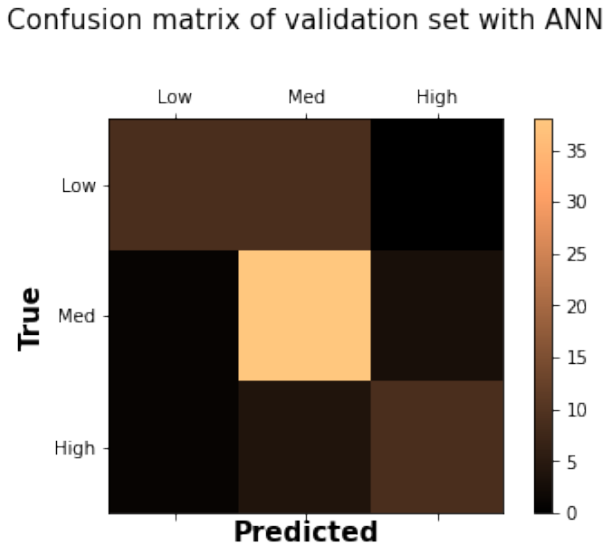

Figure 15: ANN Confusion Matrix

3.5.4 Validation and Testing

We have taken an array of untested data, which do not have any SFE values. We try to predict the data and , the classifier predicts low class.

3.5.5 Conclusions

A range of SFE values is essential to knowing the deformation regime and not exact SFE values itself. So, we used available data to find out deformation regime w.r.t to SFE values. Later we used untested data with just compositions(with no SFE values) to predict its deformation regime. So, this algorithm can directly used to test deformation regime of given composition in austenitic steels.

4 Git log

```
commit c8e364bf68f906fb73bcaef413681167ca58f71f
Author: RamyaReddyKoppula <your_email_address@example.com>
Date:   Tue Apr 19 20:59:07 2022 +0200

    final

commit 812dcb169bd1d6cbb4f488bc7f3fac290abdc260
Author: RamyaReddyKoppula <your_email_address@example.com>
Date:   Tue Apr 19 20:36:41 2022 +0200

    set

commit 1d6c9c5422c57e9f333e44d17f02275b2fac3603
Author: RamyaReddyKoppula <your_email_address@example.com>
Date:   Tue Apr 19 16:21:32 2022 +0200

    test

commit ff4e88da5436d8104e9ecda5c7d8aa076fc90849
Author: RamyaReddyKoppula <your_email_address@example.com>
Date:   Tue Apr 19 15:55:11 2022 +0200

    dhjgvdjh

commit c1193c114ac90bb51bdf93148d6f9a410324f591
```

```
Author: RamyaReddyKoppula <your_email_address@example.com>
Date:   Tue Apr 19 15:54:57 2022 +0200

    almost final

commit b257774d42ed7f70ee21acae2307f86868b1617c
Author: RamyaReddyKoppula <your_email_address@example.com>
Date:   Mon Apr 18 17:07:19 2022 +0200

    ann_final

commit 9d153055ad11be67ed01e3752ce8c653a50b6f03
Author: RamyaReddyKoppula <your_email_address@example.com>
Date:   Mon Apr 18 16:43:54 2022 +0200

    MDS

commit 54143ad3f3d507c3f8ea4bf50f7d6c8eff0d7b41
Author: RamyaReddyKoppula <your_email_address@example.com>
Date:   Sun Apr 17 22:55:24 2022 +0200

    cleat plot2

commit 008caef3ee6cc750807acca5073992f42c62be1e
Author: RamyaReddyKoppula <your_email_address@example.com>
Date:   Sun Apr 17 22:41:41 2022 +0200

    clear plots

commit 11edf0a6bff6162da7adc5d8db65aad8f46471f1
Author: RamyaReddyKoppula <your_email_address@example.com>
Date:   Sun Apr 17 15:06:28 2022 +0200

    Loss function imp

commit 1cf03713eac02775e4bb29d49a6b02489dfbe0c5
Author: RamyaReddyKoppula <your_email_address@example.com>
Date:   Sun Apr 17 02:20:30 2022 +0200

    che
```

```
commit 3fc111a77edb2e76a70bf788483c4448c979c86c
Author: RamyaReddyKoppula <your_email_address@example.com>
Date:   Sun Apr 17 02:19:23 2022 +0200

    check

commit 49836db8dbefe009f4bc3236b2448faeec7b6485
Author: RamyaReddyKoppula <your_email_address@example.com>
Date:   Fri Apr 15 22:45:26 2022 +0200

    ANN

commit c0fb8804d77cc162d88fe8f9e739354821417fa5
Author: RamyaReddyKoppula <your_email_address@example.com>
Date:   Fri Apr 15 22:00:20 2022 +0200

    tests

commit 83f6f81ccb874e6c15e050dbe422fb43e19dddaa
Author: RamyaReddyKoppula <your_email_address@example.com>
Date:   Mon Apr 11 16:59:24 2022 +0200

    ANN1

commit b5d70d14fb0c110c399a9f46f8be1ad8e578a39b
Author: RamyaReddyKoppula <your_email_address@example.com>
Date:   Sat Apr 9 14:00:49 2022 +0200

    runs after rf error

commit 19e4c08330555240724502d65381deb0bcb6ca71d
Author: RamyaReddyKoppula <your_email_address@example.com>
Date:   Sat Apr 9 02:08:09 2022 +0200

    true_branch

commit b6ed08b537cd83d86fb66457e5944a55d3363feb
Author: RamyaReddyKoppula <your_email_address@example.com>
Date:   Fri Apr 8 16:04:41 2022 +0200

    decision_tree
```

```
commit 6d9f8abb757534f94992d8f122c65e81e050cebc
Author: RamyaReddyKoppula <your_email_address@example.com>
Date:   Fri Apr 8 15:47:09 2022 +0200

    rf1

commit af372bce428e80ec9f2ef41ece0b3ca13687796a
Author: RamyaReddyKoppula <ramyasrireddy28@gmail.com>
Date:   Fri Apr 8 15:27:56 2022 +0200

    RF_1good

commit 5668e577838cdf113f4a79001082a9019e739a8a
Author: RamyaReddyKoppula <ramyasrireddy28@gmail.com>
Date:   Wed Apr 6 22:28:53 2022 +0200

    ref

commit fdc1e8692c7a62fd1447162cad475914ff5e7760
Author: RamyaReddyKoppula <ramyasrireddy28@gmail.com>
Date:   Tue Apr 5 21:06:29 2022 +0200

    man

commit f93656117610cac14074c48feff1a25df6a43966
Author: RamyaReddyKoppula <ramyasrireddy28@gmail.com>
Date:   Tue Apr 5 16:07:32 2022 +0200

    RF!

commit 47526b2ae695bb117d8ce715eb1b90cdc12c1860
Author: RamyaReddyKoppula <ramyasrireddy28@gmail.com>
Date:   Tue Apr 5 13:34:34 2022 +0200

    rf

commit 825d8b110dc12992680b35c49b3d3dc7a1818319
Author: RamyaReddyKoppula <ramyasrireddy28@gmail.com>
Date:   Mon Apr 4 23:05:39 2022 +0200
```

```
    ch

commit 07b33478d1b8687d16f51852cbbc82a4093200e6
Author: RamyaReddyKoppula <ramyasrireddy28@gmail.com>
Date:   Sat Mar 12 11:41:36 2022 +0100

    random

commit f83b097c5a97a628a6053ad098c17b847fd569ec
Author: RamyaReddyKoppula <ramyasrireddy28@gmail.com>
Date:   Thu Feb 24 13:23:13 2022 +0100

    try

commit ea097136a4c579a048f147b9fbb468aa780792b8
Author: RamyaReddyKoppula <ramyasrireddy28@gmail.com>
Date:   Tue Feb 15 11:18:41 2022 +0100

    trying ANN

commit ea5e98c0bc1f7b899574a018492867a46d730d87
Author: RamyaReddyKoppula <ramyasrireddy28@gmail.com>
Date:   Mon Feb 14 15:38:18 2022 +0100

    implementing logistic regression and neural network

commit b6e0c76f8dcc06c4739efa1cfb374b4c0333afcc
Author: RamyaReddyKoppula <ramyasrireddy28@gmail.com>
Date:   Mon Jan 31 15:27:54 2022 +0100

    trying

commit da242459ebbcb4b17f2c33df8c6b367fdb002b6a
Author: RamyaReddyKoppula <ramyasrireddy28@gmail.com>
Date:   Wed Jan 26 17:09:57 2022 +0100

    tryed sklearn split

commit d326d83f55ca34aa42d416cd257fa2274c1e79ee
Author: RamyaReddyKoppula <ramyasrireddy28@gmail.com>
Date:   Thu Jan 20 16:32:57 2022 +0100
```

```
    trying SVM using cvxopt

commit 5fe667d5bbded1d61d12438a0c2ae8bcdd1fa5bf
Author: RamyaReddyKoppula <ramyasrireddy28@gmail.com>
Date:   Wed Jan 19 12:57:44 2022 +0100

    save con

commit 082219b08a5dfdf53036788a180ae1297148f59f
Author: RamyaReddyKoppula <ramyasrireddy28@gmail.com>
Date:   Tue Jan 18 14:22:54 2022 +0100

    norm change

commit c113ce6badd3fc71642f8bc72203279ae0952e61
Author: RamyaReddyKoppula <ramyasrireddy28@gmail.com>
Date:   Tue Jan 18 14:00:48 2022 +0100

    normalization

commit cd9d44a1365b63e1f2de5239ac97509efea13721
Author: RamyaReddyKoppula <ramyasrireddy28@gmail.com>
Date:   Mon Jan 17 14:43:35 2022 +0100

    SVM

commit 1196104497eef2377093678810751545 8c7df025
Author: RamyaReddyKoppula <ramyasrireddy28@gmail.com>
Date:   Fri Jan 14 10:34:22 2022 +0100

    Multidimensional Scaling 1att

commit 0994256bf134569aeaec4b9353f5515362ada238
Author: RamyaReddyKoppula <ramyasrireddy28@gmail.com>
Date:   Thu Jan 13 15:08:29 2022 +0100

    change before imcoparting manual pca

commit cc8a74982fcc9ae399d2381a17c2f8cdcca0572a
Author: RamyaReddyKoppula <ramyasrireddy28@gmail.com>
```

```
Date:   Thu Jan 13 14:30:31 2022 +0100

    PCA_changes

commit 5a1150bffda1787748c680979d30444d04ae637e
Author: RamyaReddyKoppula <ramyasrireddy28@gmail.com>
Date:   Wed Jan 12 22:06:24 2022 +0100

    nor is bit ok

commit f40bf0c6d73a11997c8508891d7422ddb0e31360
Author: RamyaReddyKoppula <ramyasrireddy28@gmail.com>
Date:   Wed Jan 12 00:40:23 2022 +0100

    starting PCA

commit 6b19c72da92442f082f3d600eba227564309f0d3
Author: RamyaReddyKoppula <ramyasrireddy28@gmail.com>
Date:   Tue Jan 11 00:58:19 2022 +0100

    compared c with other elements

commit d220dcdacb2ca4a11766359ff6c7f153d1722c94
Author: RamyaReddyKoppula <ramyasrireddy28@gmail.com>
Date:   Mon Jan 10 01:26:36 2022 +0100

    starting element wise plotting

commit eda1a9deac08cbb0bdc2566cc56e7a1f430620a3
Author: RamyaReddyKoppula <ramyasrireddy28@gmail.com>
Date:   Sat Jan 8 01:07:13 2022 +0100

    three types of classes

commit 93cd2fc37fc68e54bb7104ab0eead404ea15bed7
Author: RamyaReddyKoppula <ramyasrireddy28@gmail.com>
Date:   Fri Jan 7 23:50:45 2022 +0100

    befoe giving index 1

commit 8727035280f37a1d40b15e68318f17aaa8f9d08e
```

```
Author: RamyaReddyKoppula <ramyasrireddy28@gmail.com>
Date:   Fri Jan 7 12:48:51 2022 +0100

    374 len

commit ddc99bce9798ae7cf602f40bdd77d9eb22ddceee
Author: RamyaReddyKoppula <ramyasrireddy28@gmail.com>
Date:   Thu Jan 6 21:55:48 2022 +0100

    reduce dataset

commit 0cd3f6aea97173f1195060c94e9f8c5843d95168
Author: RamyaReddyKoppula <ramyasrireddy28@gmail.com>
Date:   Wed Jan 5 14:34:17 2022 +0100

    experimental and room temp

commit e77b6b1f26f8ee691014e3a5b6f4c7fb2ecaede4
Author: RamyaReddyKoppula <ramyasrireddy28@gmail.com>
Date:   Wed Jan 5 02:00:25 2022 +0100

    first
```

References

[1] Dieurix develops machine learning models.

[2] https://theaisummer.com/optimization/.

[3] https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6/.

[4] machinelearningmastery.com/choose-an-activation-function-for-deep-learning/.

[5] machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/.

[6] medium.datadriveninvestor.com/decision-trees-lesson-101-f00dad6cba21.

[7] sas.com/en-in/insights/analytics/neural-networks.html.

[8] towardsai.net/p/machine-learning/how-to-build-and-train-your-first-neural-network-9a07d020c4bb.

[9] Sadhvi Anunaya. www.analyticsvidhya.com/blog/2021/08/data-preprocessing-in-data-mining-a-hands-on-guide/.

[10] NAYAN Chaudhary, A Abu-Odeh, I Karaman, and R Arróyave. A data-driven machine learning approach to predicting stacking faulting energy in austenitic steels. *Journal of Materials Science*, 52(18):11048–11076, 2017.

[11] Nagesh Singh Chauhan. decision-tree-algorithm-explained. 2022.

[12] Niklas Donges. builtin.com/data-science/random-forest-algorithm.

[13] IBM Cloud Education. ibm.com/cloud/learn/random-forest.

[14] Casper Hansen. mlfromscratch.com/.

[15] Zakaria Jaadi. builtin.com/data-science/step-step-explanation-principal-component-analysis.

[16] Bernd Klein. python-course.eu/machine-learning/confusion-matrix-in-machine-learning.php. 2022.

[17] Joydwip Mohajon. Confusion matrix for your multi-class machine learning model, published in towards data science. 2020.

[18] I. Karaman N. Chaudhary, A. Abu-Odeh and R. Arro´yave. A data-driven machine learning approach to predicting stacking faulting energy in austenitic steels.

[19] Ashwini Kumar Pal. Multi-dimension scaling, blog.paperspace.com/dimension-reduction-with-multi-dimension-scaling/.

[20] Sebastian Ruder. https://ruder.io/deep-learning-optimization-2017/index.html. 2017.

[21] Keerthana V. analyticsvidhya.com/blog/2021/04/artificial-neural-network-its-inspiration-and-the-working-mechanism/. 2021.