Project Report

Applying Machine Learning techniques for the prediction of Damage and Stress Hotspots

Submitted by
Abhijeet Pendyala

Supervised by
Dr. Hamad ul Hassan

January 2019

Acknowledgements

I would like to express my sincere gratitude to Dr.-Ing. Hamad ul Hassan, for his unwavering support technically and organizationally throughout the course of my project work. I also thank Prof.Dr. Alexander Hartmaier for giving me this opportunity to work at Chair of ICAMS, RUB and chance to explore this avenue of research. I would like to whole-heartedly thank Prof.Dr. Tobias Glasmachers, for his valuable time, feedback, and discussions which have shaped the course of my work.

Finally, I would like to express my profound gratitude to my family, friends and colleagues. This accomplishment would not have been possible without them.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Machine learning in computational material science- State of the art

With the advent of material informatics, artificial intelligence is being used to extract correlations between physical characteristics of materials and observed phenomenon [1]. In this paper, the need to focus on four V's of big data was highlighted, instead of just through putting large volumes of simulation or experimentation data. Volume, velocity, variety and veracity are the different characteristics of big data which have a collective impact on the search for identification of key parameters or structural features.

Twin nucleation within grains and twin propagation across grain boundaries in Mg alloys were studied using decision trees and attribute based rules were extracted by [2] . The data was mined from electron backscatter diffraction (EBSD) scans and relationships between microstructure and twin formation were predicted. 86.5% and 96.1% accuracies where achieved for predicting twin nucleation and twin propagation respectively across grain on a cross validated set of 104 grains.

A compressed-sensing based feature selection methodology for discovering the best physical descriptors that describe the material properties of interest was introduced by [3]. Out of a large list of feature space, LASSO was used for prescreening and ℓ_0-norm minimization for identification of most important features. The model presented could capture linear correlations among different features and was found to be better than kernel ridge regressor (KRR).

Hybrid optimization software suite (HOSS) a FDEM analysis tool was chosen to predict the location and timing of the fracture coalescence for brittle material by [4]. Decision trees and Random forests used 63 parameters and predicted the time to failure of various material fracture distributions within 5% of the ground truth model.

In [5] ANN was trained with Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm, and the physiochemical properties like molar heat capacity, standard molar entropy and lattice energy of crystals with different compositions. A topological approach for crystal structure representation was deployed which helped in determining properties of crystals

based on elementary properties of atoms with a mean absolute percentage error (MAPE) less than 8%.

For Face centred Cubic Materials and Hexagonal close packed materials [6], [7] have shown that Random forests can achieve an AUC score of 74% and 82% respectively in predicting the location of stress hotspots when subjected to simulated Uniaxial tensile deformation. 22 geometrical descriptors and crystallographic descriptors were chosen from feature selection using FeaLect method and were used to train the Random forest models with a tailored K-fold cross validation.

A one of kind online learning model was used by [8] to identify crack behaviour in concrete dams. Here a sequential learning algorithm called OS-ELM (online sequential extreme learning machine) which combines genetic algorithm with bootstrap confidence was used instead of a batch learning model. The proposed model had superior values in all four metrics viz. KMAPE, KMSE, KNMSE, and R2 and could efficiently identify concrete dam crack behaviour.

As a bio-mechanics application [9] used Deep neural network to substitute structural FEA analysis and predicted stress distribution of the aorta in the heart. For an input aorta geometry, the trained DL model estimates the wall stress distributions faster than FEA calculations and achieves a normalized mean average error of 0.49%.

Computational aided microstructure design aided with machine learning techniques was deployed by [10] to establish structure-property linkages. Gaussian process regression (GPR) models were trained with low dimensional representation of microstructural features obtained from model reduction technique (PCA) as training examples and full-field simulations as labelled data. Using this trained model accurate structure-property linkages of over 1100 synthetic microstructures derived from DREAM.3D were established. In addition, GPR combined with Expected improvement function (EI), could find the optimal microstructure that maximizes a specific property.

A High resolution statistical damage classification (4-classes) approach based on Deep convolutional neural networks for ex- and in-situ experiments was proposed by [11] . The damage was classified damage based on panoramic images from scanning electron microscope of Dual-phase 800 steels.

Instead of using the most common approach of deterministic ML models in computational material science, [12] used a probabilistic approach. Dynamics Bayesian Network (DBN) was used to account for uncertainty, in the context of health monitoring of orthotropic steel deck (OSD), which is often subjected to constant cyclic fatigue load. The DBN based framework along with Gaussian process (GP) surrogate model implemented into as conditional probability distribution (CPD), create a framework to diagnose crack growth as well as predict it.

Most recently processing-structure properties which are fundamental for the design of new materials were predicted from microstructural images using Deep convolutional networks [13]. CNNs were used for microstructure classification and property prediction for Ti-6Al-4V alloys used in jet engine. Also, High-cycle fatigue curves were predicted accurately from random samples of various heat treated Ti-6Al-4V microstructure images.

Inspired from such works, the goal of the project work was set to utilize the large scientific data obtained from material simulations to garner uncaptured phenomenon between microstructural characteristics and possibly extend machine learning (ML) techniques as a substitute to the costly and time-consuming simulations.

1.2 Structure

The project work is organized as follows: Firstly, a gentle introduction is given to all the ML models deployed for this work in chapter-2. In depth analysis of material and fatigue modeling done to simulate microstructural data for damage prediction is done in chapter-3, followed by application of ML models and relevant results. Similar structure is followed in chapter-4, where theory behind prediction of stress hotspots is laid out followed by ML models and results. In the end, chapter-5 summary of work is presented with conclusions.

2 Machine Learning Fundamentals

Machine learning (ML) algorithms are broadly classified into four groups: *Supervised learning, unsupervised learning, semi-supervised learning and reinforcement learning* [14], [15] and [16]. The categorization is based on how learning is performed. A schematic with examples is illustrated below:

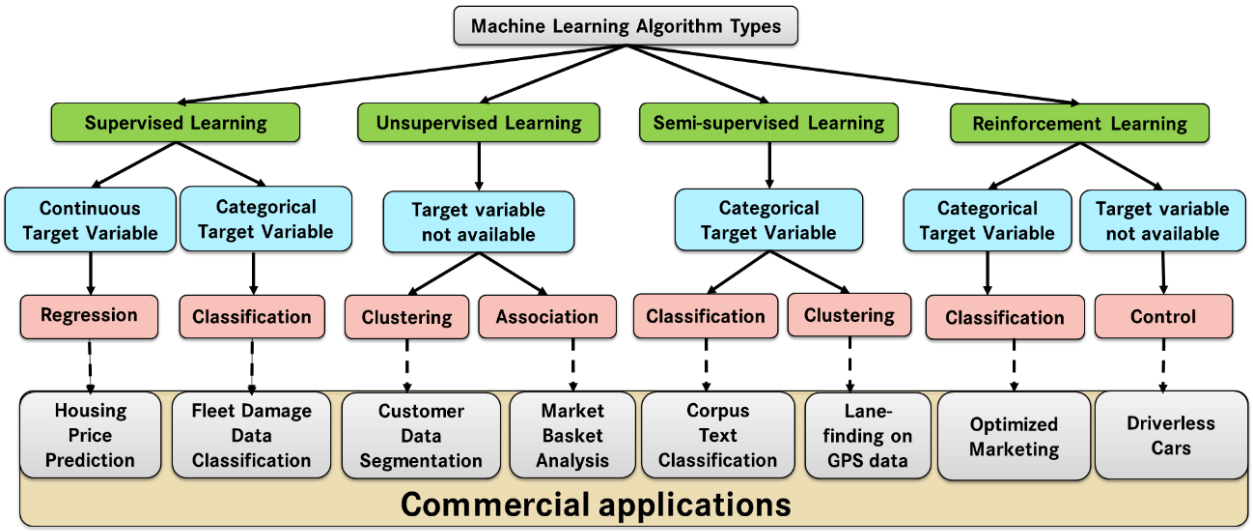


Fig. 2-1: Machine learning landscape in commercial world [adapted from [15]]

This project work focuses mainly on supervised learning ML algorithms. A supervised learning algorithm is a function map between the given inputs and outputs. The input data is called the training data and their respective outputs are called as targets. The function map learnt by the machine learning algorithm through the training process is the called the model [17].

As a first step in the training process, it is provided with the training data and its corresponding target labels. Based on the given data the learning algorithm will formulate an initial hypothesis and certain parameters are learnt to make predictions. These predictions from the hypothesis are compared with the real training targets and the errors made by the algorithm are calculated. These errors are rectified/reduced by

adjusting the learnt parameters in the model. This iterative process is repeated until an acceptable performance level is reached. Then the machine learning model is ready to do make predictions on the new data [17].

The supervised ML algorithms have proven to be successful in almost all the domains of science and engineering where labelled data is available. From Fig. 2-1 it can be observed that the supervised learning algorithms are classified into two groups based on the target specifications: ***Regression algorithms*** for continuous targets and ***classification algorithms*** for categorical targets. In this project work, both these algorithms are deployed. Therefore, to understand the in-built mathematics, necessary concepts from probability theory and the algorithms for model building are discussed in detail in the following sub-sections.

2.1 Linear Regression

In Linear regression, one of the supervised learning algorithms a linear function is used to map the dependent variable Y (target) and the independent variable X (features). A simple linear regression [Fig. 2-2] corresponds to the case where the independent variables X is only one. A multivariate linear regression [86] has more than one independent variable. The multivariate regression is represented as:

$$Y = \boldsymbol{f}(X), \qquad where\ X = (x_1, \dots, x_n) \qquad \text{Eq. 2-1}$$

The function $\boldsymbol{f}$ is modeled as a linear function with parameters called as *weights* and *biases*. Incorporating them turns the equation into:

$$\tilde{Y} := \boldsymbol{W^T} X + \boldsymbol{b}, \qquad where\ \boldsymbol{W} := weights\ \&\ \boldsymbol{b} := biases \qquad \text{Eq. 2-2}$$

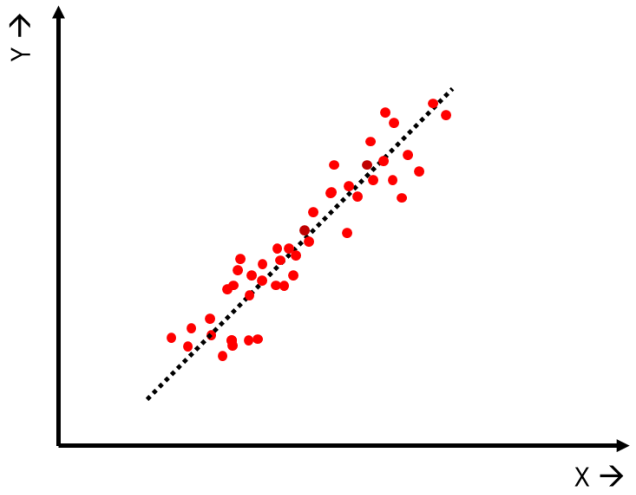


Fig. 2-2: Simple linear regression

In eq. 2-2 the LHS is represented as $\tilde{Y}(predicted\ values)$ instead of Y as the linear function is an approximate function defined on the feature space X. In order to estimate the optimal weights and biases, the algorithm needs to be trained with training data until the convergence of the loss function mentioned in eq. 2-3.

In general, the given dataset is divided in two parts: ***Training set*** and the ***testing set***. The training set is used for training the algorithm and thus obtaining optimum values of weights and biases. The testing set, which is the unknown set for the learnt model, is used to measure the accuracy of the model, which evaluates its performance.

The model adjusts the weights and biases to minimize the error. To capture the error, i.e. discrepancy between predicted and actual values a cost function is used. In linear regression, least squared error cost function is most popular one, defined as:

$$Least\ squares\ cost\ function: = \frac{\mathbf{1}}{\mathbf{2m}} * \sum_{\boldsymbol{j=1}}^{\boldsymbol{m}} (\tilde{\boldsymbol{y}}^{\boldsymbol{j}} - \boldsymbol{y}^{\boldsymbol{j}})^{\mathbf{2}} \qquad \text{Eq. 2-3}$$

The summation in eq. 2-3 is carried out on the entire training set. The progress made by a single step of eq. 2-2 and eq. 2-3 constitute a **forward propagation step**.

The machine learning training process is mathematically a minimization problem, where the cost function is minimized. This iterative minimization process is carried uses a Gradient Descent approach to reach the minimum [5]. Here, a gradient of the cost

function is taken and is used to update the weights and biases of the linear function in the direction of descent. A hyperparameter called learning rate controls the speed of the update process. The cost function from eq. 2-3 can also be represented as:

$$cost\ fn\ (W,b) := \frac{\mathbf{1}}{\mathbf{2m}} * \sum_{\mathbf{j=1}}^{\mathbf{m}} ((W^{\mathbf{T}} x^j + b^j) - \mathbf{y^j})^{\mathbf{2}} \qquad \text{Eq. 2-4}$$

After substituting eq. 2-1 in eq. 2-2 the least squares cost function is converted into a function of weights and biases. Subsequently the gradients are calculated with respect to weights and biases. The gradients of the cost function are:

$$\frac{\partial\ cost(W,b)}{\partial W} = \frac{1}{m} * \sum_{\mathbf{j=1}}^{\mathbf{m}} ((W^{\mathbf{T}} x^j + b^j) - \mathbf{y^j}) * x^j \qquad \text{Eq. 2-5}$$

$$\frac{\partial\ cost(W,b)}{\partial b} = \frac{1}{m} * \sum_{\mathbf{j=1}}^{\mathbf{m}} ((W^{\mathbf{T}} x^j + b^j) - \mathbf{y^j}) \qquad \text{Eq. 2-6}$$

For the gradient descent update step, the learning rate α is used. The update must be simultaneous in one step for both the weights and biases. The update rule is as follows:

$$W := W - \alpha * \frac{\partial\ cost(W,b)}{\partial W} \qquad \text{Eq. 2-7}$$

$$b := b - \alpha * \frac{\partial\ cost(W,b)}{\partial b} \qquad \text{Eq. 2-8}$$

Updating the weights and biases using the gradient of cost function corresponds to a **backward propagation step**. In other words, the overall learning process is an iterative sequence of forward and backward propagation steps until the cost is minimized.

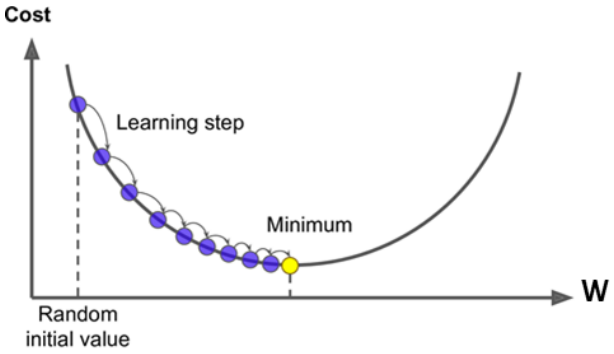


Fig. 2-3: Gradient descent algorithm [18]

If the cost function reaches a global minimum the training process is complete, and the model is tested on the testing dataset. The testing dataset is used to make the predictions

from the new model and they are compared to the actual values for evaluating the accuracy of the linear model. The most common evaluation metric for such task is the RMSE (root mean squared error) which is defined as follows:

$$RMSE: = \sqrt{\frac{\sum_{j=1}^{m}(\tilde{y}^j - y^j)^2}{m}} \qquad \text{Eq. 2-9}$$

The general norm is that, if RMSE is less and nearly equal to zero then the linear model is good. RMSE avoids canceling of positive and negative error terms, which makes it advantageous over mean absolute error (MAE). Other popular metrics are $R^2\ score$ and mean squared log error (MSLE) which are widely used for measuring the performance of the model [7]. In a concise summary the complete process consists of the training phase where the weights are updated accordingly by gradients.

2.2 Logistic Regression Classifier

The linear regression model is applicable when the target variable is defined as a continuous value. Logistic regression classifier is a ***linear model*** which deals with categorical targets and was developed by statistician David cox in 1958 [8]. Initially it was developed for binary targets, but later it was extended to multinomial targets as multinomial logistic regression [19]. The fundamental difference between a linear and logistic model is that in linear regression a linear hyperplane equation is approximated as a function for the data, whereas in logistic regression classifier the same linear hyperplane is used as separator or classifier for the categorical targets. This is illustrated in the figure below:

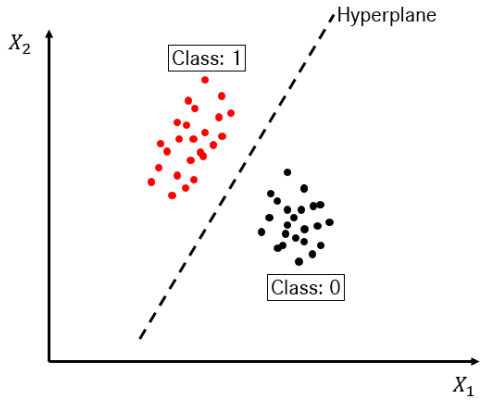


Fig. 2-4: Logistic regression classifier output

The approximating linear function for a logistic model is same as the one for linear model as represented in Eq. 2-2. Although, here this equation is activated with an activation function before outputting the probabilities or predictions. The popular choice for activation function is a sigmoid function as depicted below:

$$sigmoid = \sigma(\tilde{y}) := \frac{1}{1 + e^{-\tilde{y}}} \qquad \text{Eq. 2-10}$$

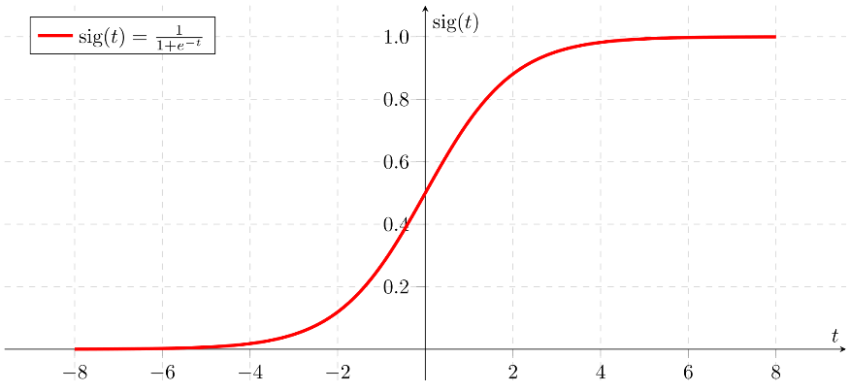


Fig. 2-5: Sigmoid function

For very large positive input values the sigmoid value is close to 1 and for large negative input values its value is almost equal to zero. This boundedness property of sigmoid function is helpful in obtaining the predicted probabilities for a respective target class. Other popular activation functions are $tanh, relu\ and\ softmax$ functions which have specific purposes based on the application. Due to the addition of such activation function cost function [20] for logistic regression classifier is newly defined as:

$$logistic - cost = \frac{-1}{m} * \sum_{j=1}^{m} (y^j * log(\tilde{y}^j) + (1 - y^j) * log(1 - \tilde{y}^j)) \qquad \text{Eq. 2-11}$$

$$\text{where } \tilde{y}^j := \sigma(W^T x^j + b^j)$$

In logistic regression the forward propagation and backward propagation are the same as for linear regression but with the addition of the activation function and the usage of logistic cost function instead of least squares cost function. The gradients for the logistic regression with new cost function are shown below:

$$\frac{\partial\, cost(W,b)}{\partial W} = \frac{1}{m} * \sum_{j=1}^{m} (\boldsymbol{\sigma}(W^{T} x^{j} + b^{j}) - \boldsymbol{y^{j}}) * x^{j} \qquad \text{Eq. 2-12}$$

$$\frac{\partial\, cost(W,b)}{\partial b} = \frac{1}{m} * \sum_{j=1}^{m} (\boldsymbol{\sigma}(W^{T} x^{j} + b^{j}) - \boldsymbol{y^{j}}) \qquad \text{Eq. 2-13}$$

The update rules for logistic regression remain same as eq. 2-7 and eq. 2-8. The predictions after thresholding can then be compared with the actual targets to evaluate the accuracy of the classifier. Since, the targets are no longer continuous, a new set of metrics have been introduced to deal with categorical targets. Some new metrics are, to use a Confusion matrix, misclassification accuracy, etc. which will be used to measure the performance of the logistic regression classifier.

2.3 Support Vector Machines

In the previous section logistic regression classifier was discussed briefly, but such classifiers are prone to 'Overfitting' [21] . Overfitting phenomenon is observed when the trained model performs well on the training data but poorly on the testing data, i.e. it fails to generalize for unseen data. For example, in a linearly separable data shown below:

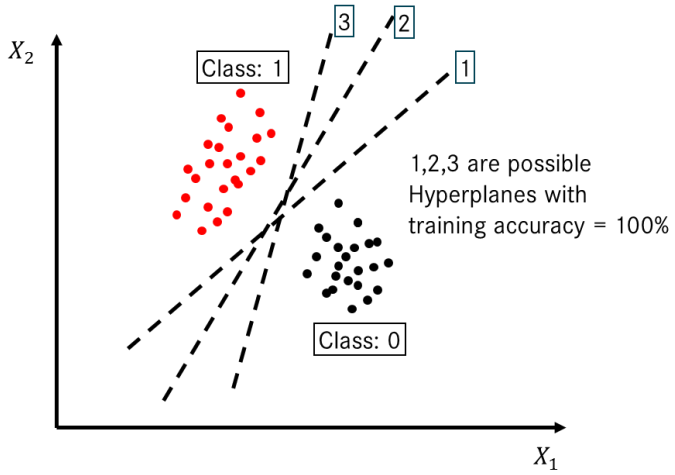


Fig. 2-6: Linearly separable data

There can be several hyperplanes which can classify all the training points accurately but can have varying test accuracy. Hence, the best hyperplane [Fig. 2-8] from the collection of all the hyperplanes must be determined. Support vector machines (SVMs) can determine such a best hyper plane. SVMs deploy a "**kernel trick"** where the input vectors

are mapped in to a higher dimensional feature space using the non-linear mapping functions called kernels. After reconstructing the higher dimensional feature space, a linear decision surface can be constructed which can have higher generalization ability [22]. To achieve this, the two main problems which are the consequences of the above-mentioned algorithm in [22] must be solved. The first problem is to find the hyperplane in higher dimensional space and the second problem is related to constructing the higher dimensional feature space which could be computationally expensive.

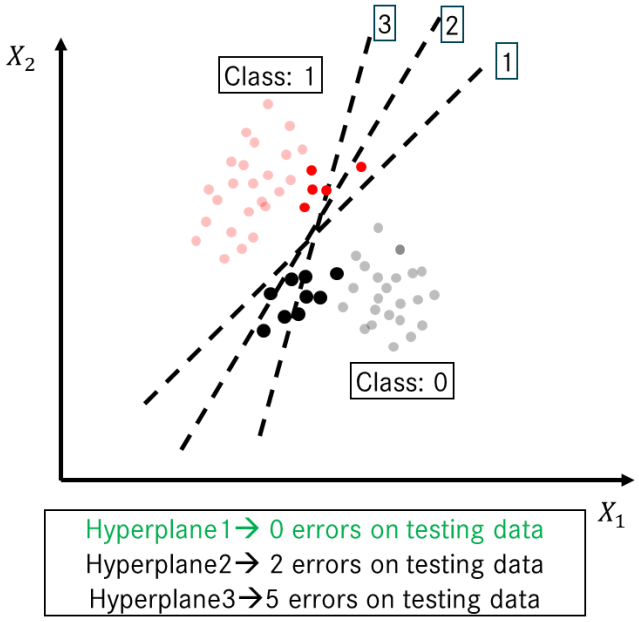


Fig. 2-7: Generalization error in test set (hyperplane 1 - best hyperplane)

The first problem was solved in [23] by selecting the hyperplane with maximum margin. Mathematically this can be achieved by maximizing the margin between the data points of different classes. Therefore, by finding this hyperplane the generalization error could be solved for a linearly separable data. This can be viewed as follows:

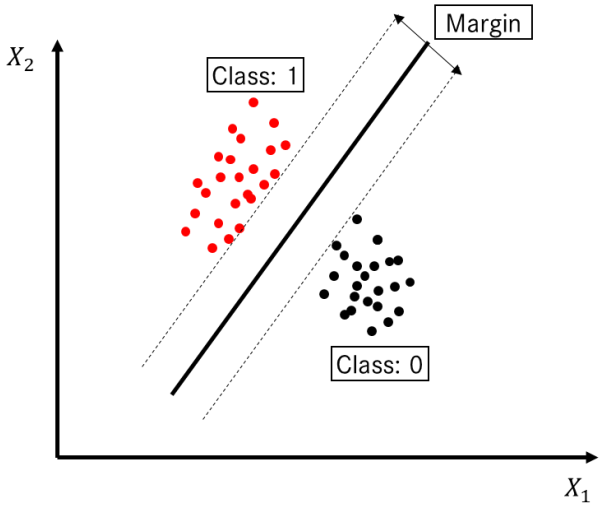


Fig. 2-8: Best margin hyperplane

The problem of maximizing the margin between hyperplane and nearest data point is formulated as a convex optimization problem and an optimal solution can be obtained using the following procedure (binary classification is chosen to illustrate):

$$\begin{gathered} margin \overset{\text{def}}{=} d_- + d_+; \\ where\ d_- \overset{\text{def}}{=} distance\ to\ the\ nearest\ negative\ training\ samples \\ where\ d_+ \overset{\text{def}}{=} distance\ to\ the\ nearest\ positive\ training\ samples \end{gathered} \qquad \text{Eq. 2-14}$$

The margin of the hyperplane in eq. 2-14 is shown in Fig. 2-8 where the distances d_- and d_+ are measured from the hyperplane to the dotted line. The weight vector w is in fact perpendicular to the hyperplane. w and bias b can be chosen in such a way that $d_- = d_+ = \frac{1}{\|w\|}$ which simplifies the problem. Since, the data is linearly separable there should be a hyperplane which classifies the data samples given as:

$$w^T x + b \geq +1 \text{ for all the +ve samples} \qquad \text{Eq. 2-15}$$

$$(w^T x + b) \leq -1 \text{ for all the -ve samples} \qquad \text{Eq. 2-16}$$

$$\text{And } margin = \frac{2}{\|w\|} \qquad \text{Eq. 2-17}$$

Eq. 2-17 is the consequence of the consideration of the weights w and biases b as mentioned above and using eq. 2-14. In eq. 2-17 in order to maximize margin, $\|w\|$ should be minimized. Therefore, the cost function is formulated as:

$$cost\ function := argmin\left(\frac{1}{2} * \|w\|^2\right) for\ w, b \qquad \text{Eq. 2-18}$$

$$constarined\ wrt. \rightarrow\ k * (w^T x + b) \geq 1\ ;\ for\ k\ \in \{-1, +1\} \qquad \text{Eq. 2-19}$$

The eq. 2-18 and eq. 2-19 form a ***Quadratic optimization problem*** and can be solved using Lagrange multipliers [24]. After solving using the Lagrange multipliers the optimal weights and biases are obtained for the hyperplane with largest margin. This procedure eliminates the overfitting problem found in linear model and logistics model, but the concern of non-linearly separable data is not yet addressed. Another issue with data in machine learning is the presence outliers. In experimental measurements, outliers in a dataset could be anomalous data points measured during the data collection process. Outliers can be depicted as:

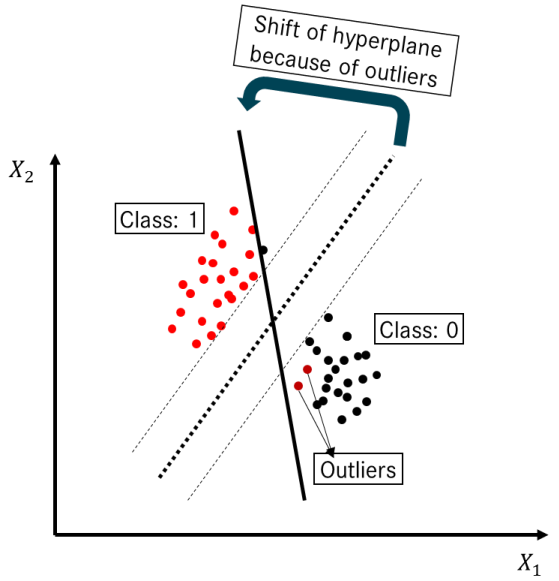


Fig. 2-9: Shift of the hyperplane because of outliers

Due to the presence of these outliers the present formulation will consider them as the end points for the class and tries to maximize the margin between them. This consideration results in shift of the hyperplane as shown in Fig. 2-9. In order to address this issue, slack variables are introduced in the optimization problem with a tradeoff parameter (regularization parameter) C in the objective function. The slack variables are introduced for each data point and is depicted in the Fig. 2-10.

$$\xi_n = 0 \text{ for all the points on the correct side of the margin} \qquad \text{Eq. 2-20}$$

$$\xi_n = |k - y(x_n)| \text{ for all the outlier points}$$
$$\text{Where } y(x_n) = w^T x_n + b \qquad \text{Eq. 2-21}$$

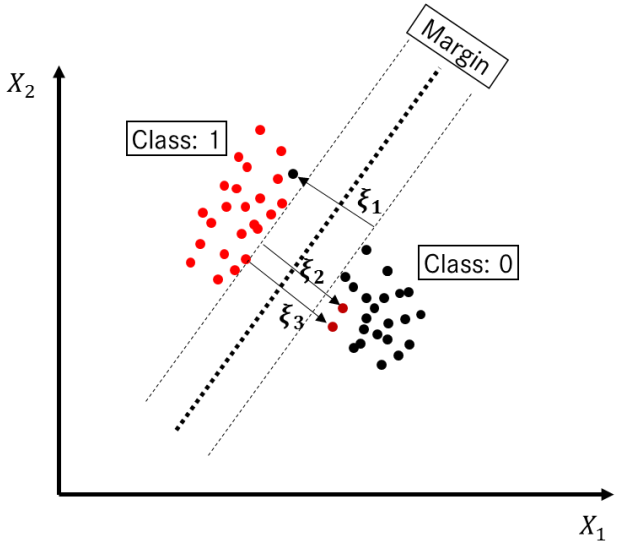


Fig. 2-10: Slack variables for the outliers

After the introduction of slack variables, the optimization problem for SVM is shown below:

$$cost\ function := argmin\left(\frac{1}{2} * \|w\|^2 + C\sum_{n=1}^{N}\xi_n\right) for\ w, b \qquad \text{Eq. 2-22}$$

$$constarined\ wrt. \rightarrow\ k * (w^T x_n + b) \geq 1 - \xi_n\ ;\ for\ k\ \in \{-1, +1\} \qquad \text{Eq. 2-23}$$

$$\text{And } \xi_n \geq 0$$

The optimization problem remains **quadratic** and it can be solved with Lagrange multipliers and using **KKT conditions** [25]. The problem setup mentioned above will form the basis for the nonlinear SVM as well but with the use of additional basis functions.

In [22] the idea of using higher dimensional spaces is floated in order to reduce the generalization error. The construction of this higher dimensional space will allow the separation of the training data. The transformation of the data points to the higher dimensional space is performed by a nonlinear transformation with the use of basis functions ϕ.

$$\boldsymbol{x} \in \mathbb{R}^D \quad \boldsymbol{\phi}: \mathbb{R}^D \rightarrow \mathcal{H} \qquad \text{Eq. 2-24}$$

$$(\boldsymbol{w}^T \phi(\boldsymbol{x}) + b) = y \qquad \text{Eq. 2-25}$$

Eq. 2-24 depicts the hyperplane in higher dimensional space $\mathcal{H}$ (linear classifier) but remapping into $\mathbb{R}^D$ gives a non-linear classifier. But calculation of $w^T\phi(\boldsymbol{x})$ becomes

cumbersome when the dimensions become very large. Eq. 2-25 is re written in the following form:

$$y = \left(\sum_{n=1}^{N} a_n t_n \, k(x_n, \boldsymbol{x}) + b \right) \quad \text{Eq. 2-26}$$

$$\text{Where } kernel\ function\ k(x_n, \boldsymbol{x}) \coloneqq \boldsymbol{\phi}(\boldsymbol{x})^T * \boldsymbol{\phi}(\boldsymbol{x}) \quad \text{Eq. 2-27}$$

Eq. 2-27 implicitly maps the data into a higher dimensional space without having to compute the basis function explicitly. The validity of a kernel is given by mercer's theorem [21] which states that every positive definite symmetric function is a kernel. Hence there are many kernels which satisfy this condition, for example:

$$Polynomial\ kernel = k(x, y) \coloneqq x^T y + 1 \quad \text{Eq. 2-28}$$

$$radial\ basis\ kernel = \ k(x, y) \coloneqq \exp\{\tfrac{-(x-y)^2}{2\sigma^2}\} \quad \text{Eq. 2-29}$$

Therefore, using the kernel functions in the optimization setup with slack variables the nonlinear decision boundary can be formulated [25].

The introduction of kernels has solved the problem of non-linearity, but there are many limitations in SVMs. The limitations are:

- Selecting the right kernel for the dataset
- Choosing the best kernel parameters
- Solving the quadratic formulation with slack variables is slow when there are large number of points

These problems are addressed in complex methods like the tree-based methods and ensemble methods which are discussed in the subsequent sub sections.

2.4 Tree based methods

Tree based methods or also known as Decision trees are quite old methods, like the logistic regression and SVM's, but are excellent choice for large datasets [26]. The core of the tree-based algorithms are if-else decision-making rules which makes them very easy to interpret. Decision trees are preferred to the models of linear, logistic and SVMs for categorical datasets [21]. The decision trees look like a flow chart with each block making a single decision. An example is shown below:

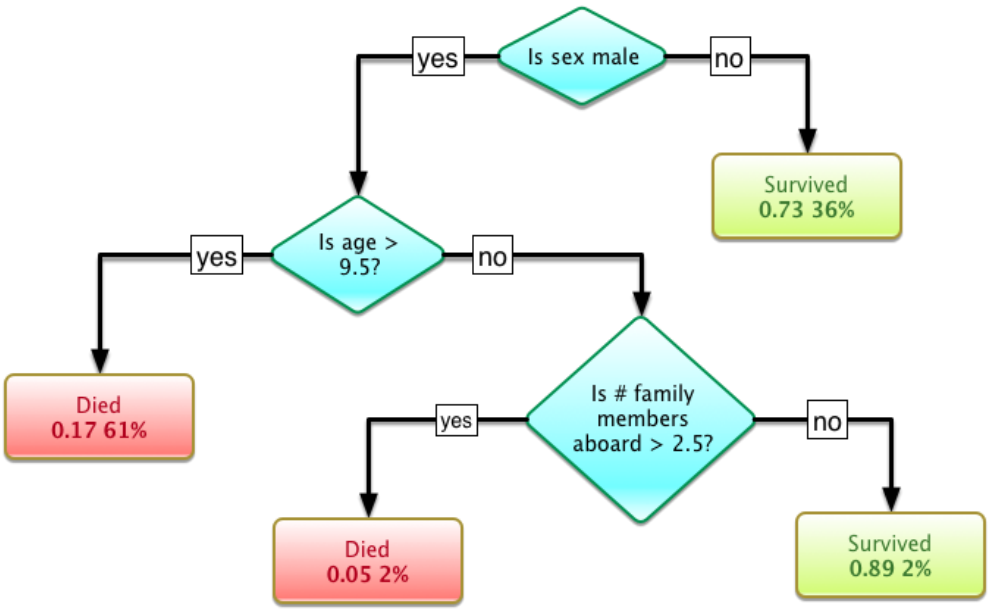


Fig. 2-11: Decision tree for Titanic survival dataset [149]

Similar to their linear counterparts (linear and logistic models), classification trees classify the data into classes while regression tress deal with continuous target variables. As a typical example, the decision tree shown in Fig. 2-11, starts with a root node at the top. This node takes in the input features and makes a decision to split the feature space into smaller subspaces. The arcs emerging from root nodes are the sub categories where further decision-making blocks or intermediate nodes are added. This process continues until the end of the tree, i.e. till leaf nodes are reached, where there is no further decision making possible.

As shown in the Fig. 2-11 a decision tree is built by splitting the training set into subsets based on the feature values at every stage. This process is called recursive partitioning [26]. The recursion is completed when the subset at a node has all the same value of the target variable, or when splitting no longer adds value to the predictions. This process is called greedy top-down induction of decision trees (TDIDT) [27]. TDIDIT can lead to huge division of feature space (over fitting) or creation of huge trees with high depth. These situations can be handled through choosing a proper decision criterion at each split. Amongst the several tree-based algorithms, most notable ones are the ID3 algorithm [27], C4.5 trees [28], CART [29] and Conditional Inference trees [30]. All these methods have a top-down approach to build up the tree, but they vary in the metric for feature space splitting i.e. the decision criteria. Two most important metrics are **Gini Impurity** and **Information gain**.

Gini Impurity is mainly used in the CART algorithm and is defined as the expected error rate at a node if the category label at that node is selected randomly. In other words, Gini impurity basically measures the misclassification probability. If the Gini impurity is smaller, the split created at the node basically gives almost uniform samples of a single class in the subsequent level. The Gini impurity from [29] is defined as:

$$I(p) = \sum_{i=1}^{J} p_i * (1 - p_i) = 1 - \sum_{i=1}^{J} p_i^2 \qquad \text{Eq. 2-30}$$

Information gain is used in the ID3, C4.5 algorithms. Information gain is based on the concept of entropy from the information theory. The entropy is defined as the amount of information gained when an event occurred. The entropy is larger when a rare event occurs, and it is smaller when a very likely event occurs. The mathematical representation of entropy $H(T)$ is:

$$H(T) = I(p_1, p_2, \dots, p_J) = \sum_{i=1}^{J} p_i * \log(p_j) \qquad \text{Eq. 2-31}$$

In eq. 2-31, the $p_1, p_2, \dots, p_J$ are the probabilities of each class present in the subsequent child node after the split from a precedent parent node in the tree. The average information gain (IG) is defined as the follows:

$$IG(T) = H(parent\ node) - weighted\ average\ H(child\ nodes) \qquad \text{Eq. 2-32}$$

Information gain is used to decide which feature to split on at each step while building the tree. Based on Occam's razor principle, the goal is to keep the tree smaller and simpler. In order to create simpler and smaller trees, at each step the split that results in the purest child nodes should be chosen (entropy is less for pure nodes). This basically implies that maximizing the average information gain at each split result in smaller trees. This idea of maximizing the information gain forms the objective function for the ID3 algorithm [27], which also is the basis for much complex variants like the C4.5 and its successor C5.0 algorithm. The decision trees are also very important building blocks for the ensemble methods (section 2.6) as they form the essential base learning algorithms in random forests and gradient boosting machines.

Because of their greedy nature of decision making, trees are prone to overfitting. To avoid these two approaches are followed: Pre-pruning and Post-pruning. In pre-pruning the

algorithm stops growing the trees during the top-down construction where there is no longer sufficient data is available to make reliable decisions. In post-pruning the tree is grown completely and then the algorithm removes the subtrees that do not have sufficient evidence [31]. Despite of the existence of such techniques, tree-based methods have following limitations:

- Do not generalize well on the data
- Deep trees produce trees which over fit the data
- Shallow trees produce trees which under fit the data
- Trees are also extremely sensitive to outliers and they can produce different results if the outliers are removed

In order to overcome these limitations, the ensemble methods were introduced.

2.5 Deep Neural Networks

Neural networks or artificial neural networks are a class of supervised learning algorithms inspired from the human biological neurons. They mimic the functioning of the biological neurons, and are built based on similar simple processing units as shown below:

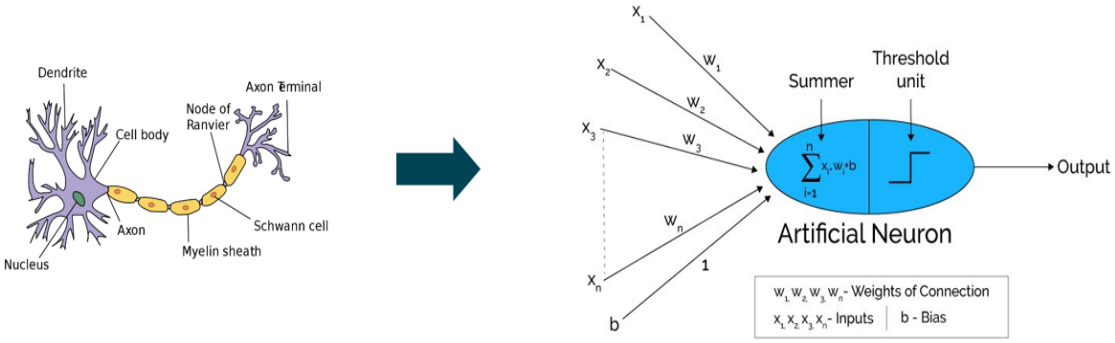


Fig. 2-12: An ANN resembling a biological neuron [30]

Input signals from adjacent cells travel to the center of neuron i.e. nucleus through dendrites where the information is processed. Then an impulse signal is transmitted to terminal of neuron through the through the axon to the axon terminal by activating the signal. Finally, the terminal transfers the signal to the next neuron. In a similar fashion, the perceptron (basic artificial neuron) [32], has two parts: a '**summer unit'** akin to nucleus of a neuron. Summer unit outputs a linear function with weights and biases. The second part of the artificial neuron, called a '**threshold unit'** (similar to axon and axon terminal) outputs a non-linear function transforming the incoming linear function with help an activation function. This activation function in the perceptron generates an

output of 0 or 1 also known as binary output. This is achieved by applying threshold [33] as shown below:

$$output = \begin{cases} 0 \; if \; \sum_j w_j x_j \leq threshold \\ 1 \; if \; \sum_j w_j x_j > threshold \end{cases} \qquad \text{Eq. 2-33}$$

The popularity of neural networks was sky rocketed after the extensive use of GPUs in the year 2006. By 2010, several neural network architectures were introduced which are very widely used even today in object recognition for autonomous vehicles, image recognition in biomedical applications, natural language processing and in finance for stock price forecast, etc. [34]

There are different types of neural networks and the most important ones are shown below.

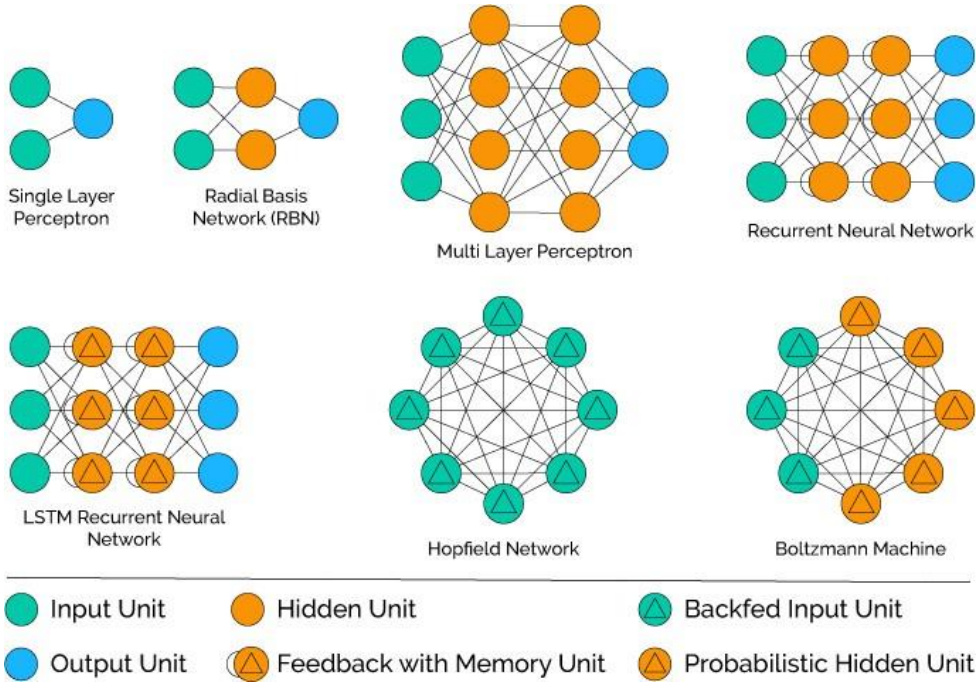


Fig. 2-13: Popular artificial neural network architectures [35]

The multi-layer perceptrons (MLP) in Fig. 2-13 are the most commonly used deep neural networks. This project works focuses mainly on MLP.

The architecture of the neural network should be understood, in order utilize them at full potential. The Fig. 2-14 illustrates a 4-layer MLP with two hidden layers, and one input, output layers. The choice of activation function used in the output layer of a MLP problem determines the target use. Rectified linear unit (ReLu) is used for regression problem,

sigmoid activation function is used for binary classification tasks, and softmax activation function is used for multiclass classification. To speed up the training process (ReLu) is used for all the remaining hidden layers of an artificial neural network [36].

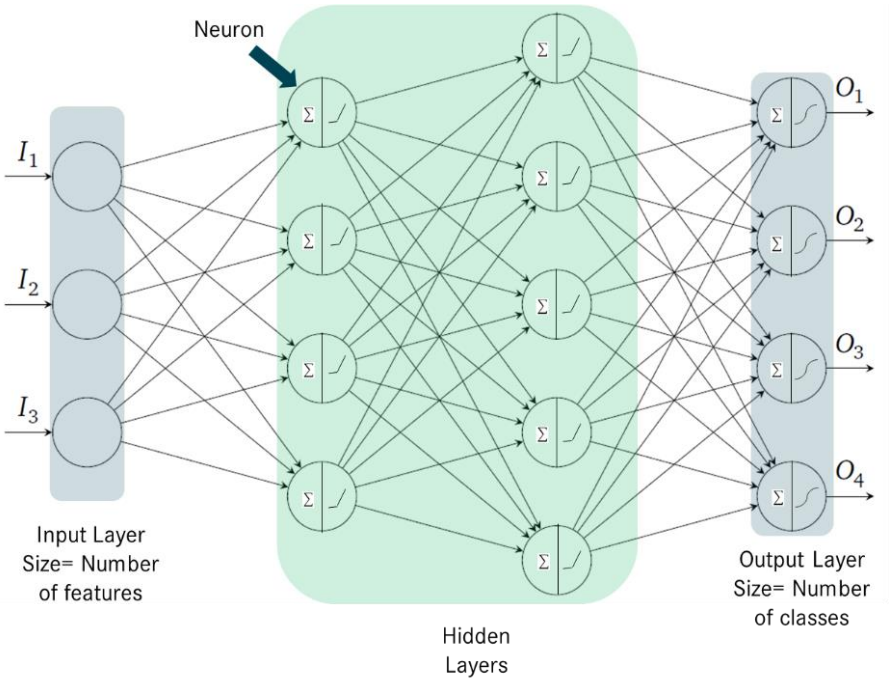
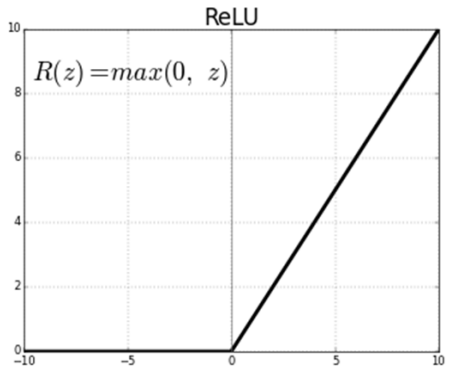


Fig. 2-14: Example of a four-layer Multi-Layer Perceptron

There are no activation or summer units in the input layer. Taking a classification problem as an example, the number of neurons in the input layer are equal to the number of features j in the given dataset i.e. $\left(x_1^m, \ldots, x_j^m\right)$ $with\ j\ features$. The hidden layers can have arbitrary number of neurons and the output layer has two neurons, since it's a binary classification task. Such MLP with less than two hidden layers is a shallow neural network, and a deep neural network has more than two hidden layers. MLP shown in Fig. 2-14 is for a multi-class classification problem, the activation functions in the hidden layers is ReLu and the output layer has softmax function. These activation functions are shown below:



Softmax

It is a normalized exponential function. It calculates the probabilities distribution of the event over 'n' different events. The functional form is:

$$softmax = \frac{\exp(x_i)}{\sum_{j=0}^{k} \exp(x_j)} \ where\ i = 0,1,2 \dots, k$$

Fig. 2-15: ReLu and Softmax activation function used in Fig. 2-14

The neural networks follow the same principles of forward and backward propagation as explained in the previous sections. A weight matrix is used to multiply the input vector before it is activated using an activation function giving the predictions $\tilde{y}_{predicted\_probab}$. At the output layer the threshold as explained in (eq. 2-33) is applied to convert into binary class problem. The Fig. 2-16 shows forward propagation step for a perceptron (single artificial neuron):

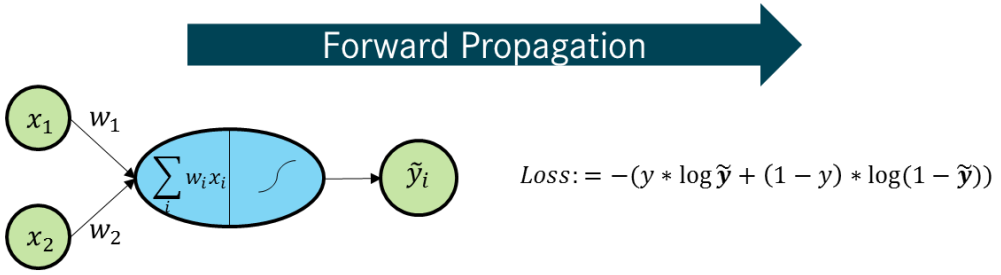


Fig. 2-16: Forward propagation with logistic loss function

For illustration purposes, the bias term was not shown in Fig. 2-14 and Fig. 2-15, which obviously needs to be added before activation. Logistic loss function is used as a cost function.

Gradient descent method is used for backpropagation and calculation of weights, biases with a learning rate of α. Backpropagation for a perceptron is shown below:

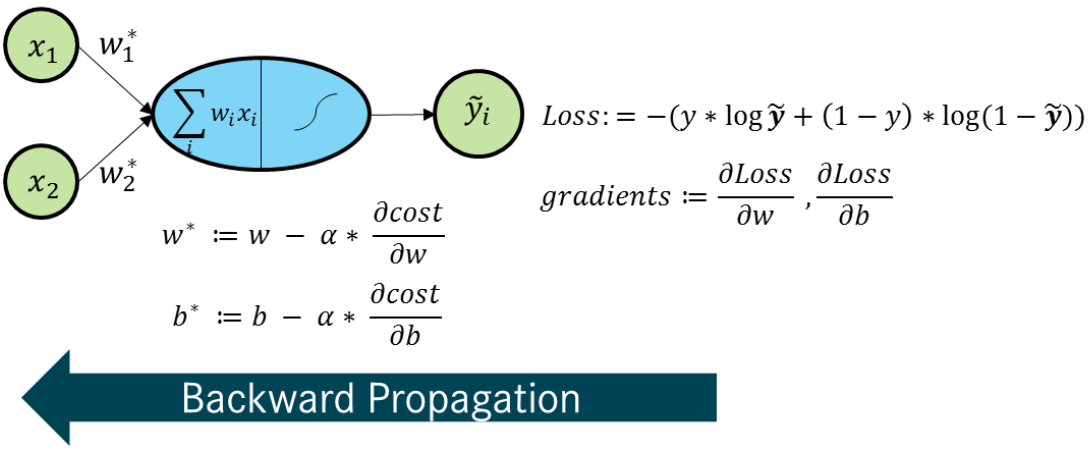


Fig. 2-17: Backward propagation using gradient descent

For a deep neural network, a concept called mini-batch gradient descent is used. In this method, the forward and backward propagation is done for a batch of training examples instead of single training example in every iteration. It is the most common implementation of gradient descent used in the field of deep learning [37].

Deep learning is currently the state-of-the-art research field in machine learning. Neural networks are capable of realizing any complex function if the number of hidden layers and other hyperparameters are rightly tuned [33], [38]. Because of these unique capabilities, deep neural networks are the popular choice for supervised learning tasks, but they also have certain limitations.

- Large amounts of data is required to train deep neural networks to attain good accuracy.
- Choosing hyperparameters like number of layers, neurons in layers, learning rate and the type of activation function are key challenges.

In the next chapter ensemble methods are discussed which have proven to be successful in achieving very good training accuracies.

2.6 Ensemble Methods

Ensemble methods are a class of supervised learning algorithms, designed from a set of base classifiers. In simple terms, they predict the output of the given labeled data by taking a weighted average of the predications from base classifiers. A key assumption for ensemble methods, is the errors in the base classifiers need to be uncorrelated, and each of the classifiers must have error probability of less than 0.5 on training set, also known as weak classifiers [39].

The biggest challenge to use ensemble methods is the need to generate different classifiers. This could be done by training same supervised algorithm on different datasets, but that would require huge datasets. This problem is alleviated by using random subsampling of the given training data. This technique of subsampling of the by randomly selecting M samples from N training data samples was used by Breiman [40] to train different decision trees, along with weight averaging using an equal weight for each base classifier in the ensemble to get the predictions. This method is called as Bootstrap aggregation or bagging. The classic example would be the random forest algorithm which combines random decision trees with bagging to achieve very high classification accuracy [40].

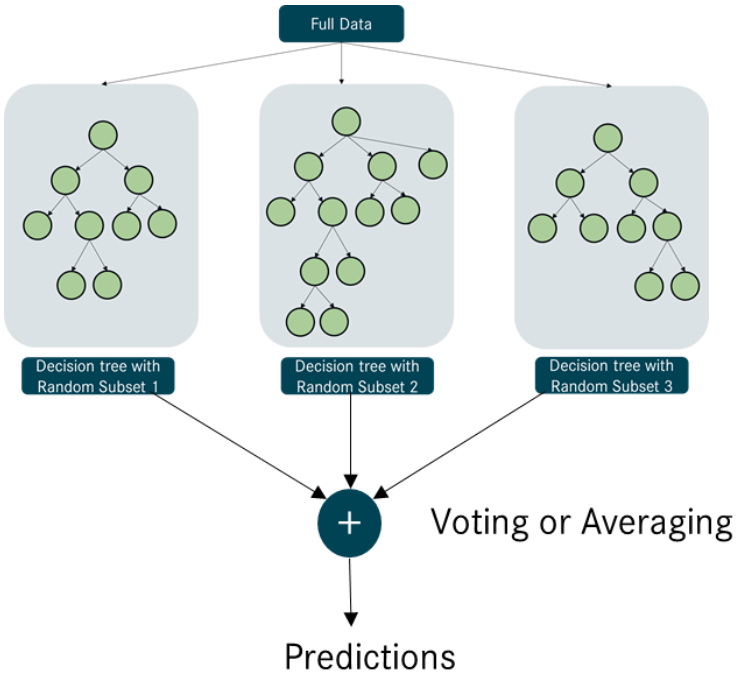


Fig. 2-18: Example of a Random Forest

An ensemble classifier can also be designed using different kinds of classifiers. This is done by using three techniques viz. **boosting, Bayesian model averaging and stacking**.

In Adaptive boosting (AdaBoost), one of the boosting algorithms, reweighting of the misclassified training examples is done rather than resampling the data. The weak base classifiers theoretically could be any supervised learning classifier, but in most cases the decision stumps are used. Decision stumps are short decision trees with maximum tree depth in the range of 3 to 7. The objective of AdaBoost is to minimize the weighted error

function. Detailed explanation is present in [41], while in Fig. 2-19 a simple case for the 2D feature space is presented. The central ideas are as follows: At first the weak classifiers split the training samples with at least 50% accuracy, then training data samples misclassified are given more weightage in the next iterations. The weights of the misclassified points are increased and then another weak classifier is used to learn over the weight increased dataset, this process repeats until the accuracy is improved. Finally, the weighted average of all the weak learners is constructed to obtain a strong classifier.

$$H_{strong}(x) = sign(\sum_{m=1}^{M} \alpha_m * h_m(x)) \text{ where } h_m(x) \text{ is a weak classifier} \quad \text{Eq. 2-34}$$

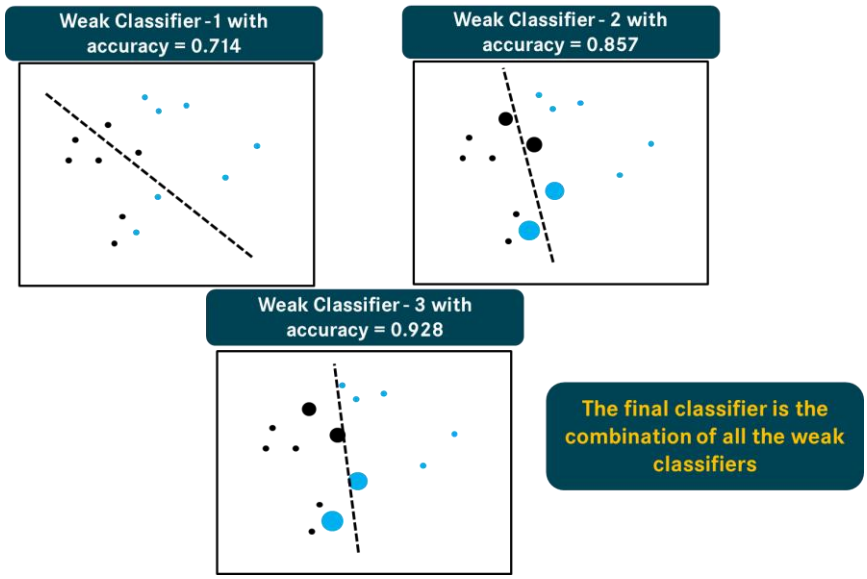


Fig. 2-19: Intuition of AdaBoost algorithm in 2D space [adapted from [41]]

Bayesian model averaging is another kind of ensembling technique which is explained in detail in [42].

Stacking or stacked generalization is a technique where different base classifiers are trained on same data and then a Metal classifier is trained on the output of the base classifiers. This is illustrated below:

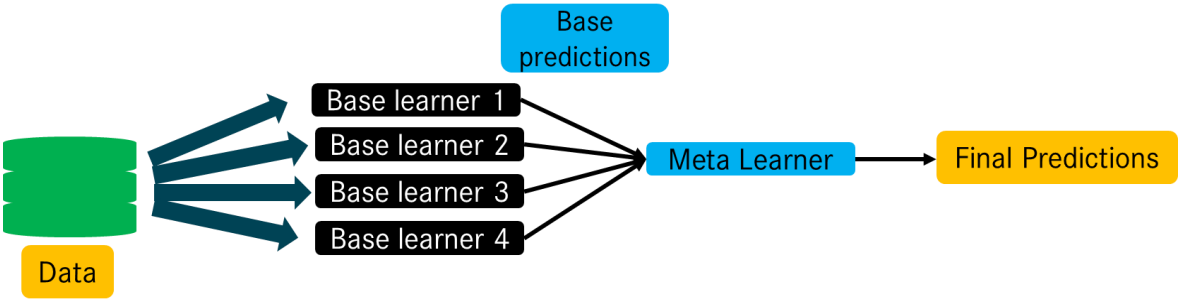


Fig. 2-20: Example of Stacking

Stacking typically performs better than any of the base learners from which it has been trained [43]. To make the errors of the base learners uncorrelated it is wise to choose a diverse set of base learners. Such can be obtained using different kinds of algorithms to create the base classifiers or by using different datasets (changing the features in the dataset) or by using different cross validation techniques. This technique is popular in online Kaggle competitions as it achieves higher accuracies. Stacking is not used in this project work due to its complexity.

3 Damage prediction using machine learning

3.1 Introduction

Although a lot of research work focused on fatigue life and fatigue life growth of materials in computational material science , [44] focused on modelling the fatigue crack initiation. An equiaxial grain micromechanical model was generated using Dynamic microstructure generator (DMG) and Crystal plasticity (CP) was incorporated to model plastic deformation which is followed by Fatigue crack initiation. The constitutive model, the fatigue crack initiation model and simulation methods are extensively discussed in [44]. The micromechanical features identified for damage prediction and the 3000 values of damage were used as training examples and labelled data for several machine learning models discussed in the forthcoming sections, in order to find a surrogate ML model as a replacement for tedious and costly simulations. The list of features can be found in Fig. 3-1.

3.2 Machine learning models

The data obtained from simulations was the input to several machine learning regression models. These models were trained on training set (80% of data) and prediction of the damage value was done on a test data set (a 20% split on original). The dataset constituted of 3,392 training examples with 10 features. All the features were continuous variables.

3.2.1 Implementation details

All the machine learning models were implemented in python and run in JUPYTER notebooks. To ensure a fair comparison based on algorithm efficiency rather than on efficiency of the implementation or environment, all the experiments were carried out on a single machine configuration. The power of parallel processing and GPUs were utilized for very fast training of models. The environment was virtual machine running 64-bit Windows operating system on Google cloud platform (GCP). The configuration constituted 4 Intel Core Broadwell vCPUS with 18.0GB of installed RAM and one NVIDIA Tesla K80 GPU.

3.2.2 Fast.ai library

The fast.ai library is used widely by machine learning practitioners and Kaggle competitors in the world of AI. It is a simple yet fast library to train accurate Random forests, neural nets using modern best practices. It's based on research in to deep learning best practices undertaken at fast.ai and university of San Francisco. This library was used in this project work to fit the regression models.

3.2.3 Data pre-processing

The first step is to import the dataset into Jupyter notebook.

```
data.tail()
```

	D[-]	D_dot[1/s]	p[-]	p_dot[1/s]	Mises[MPa]	Hydro[MPa]	teq[-]	eeq[-]	V[mm^3]	Sig_tri[-]	LoadingState
3388	0.248556	1.983663	0.062001	0.116834	284.832377	18.745354	0.061949	0.001537	0.000212	0.066052	7.0
3389	0.248579	1.948299	0.062003	0.116802	284.825419	18.743141	0.061950	0.001537	0.000212	0.066044	7.0
3390	0.248600	1.886919	0.062004	0.110842	284.818513	18.740990	0.061950	0.001537	0.000212	0.066037	7.0
3391	0.248601	1.059331	0.062004	0.135451	284.818233	18.740799	0.061951	0.001537	0.000212	0.066036	7.0
3392	0.248601	1.091343	0.062004	-0.737903	284.817976	18.740666	0.061951	0.001537	0.000212	0.066036	7.0

```
data.shape
```

```
(3393, 11)
```

Fig. 3-1: Last 5 rows and shape of the dataset for Regression

There are no categorical columns that need to be one-hot encoded. Then, the data is manipulated and the damage values D[-] is saved as the independent variable.

3.2.4 Evaluation metrics

The most common metric used for Regression task is Root mean squared error (RMSE). It tells how far the data from the best fit regression line or curve is, i.e. it is a measure of residuals. The formula is:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} (X_{obs,i} - X_{model,i})^2}{n}}$$

where X_{obs} is observed values and X_{model} is modelled values at time/place i.

3.2.5 Linear Regression

In a linear model, the target value is expected to be a linear combination of input variables, which is explained in detail in section 2.1. The linear model for the dataset obtained a RMSE value of 0.241, which is very high. This gives a hint that the nature of the dataset is non-linear. In addition to linear regression other linear models have been designed and the results of all models are summarized in Table 3-1.

3.2.6 Support Vector Regression and Kernel Ridge Regression

Using the kernel trick explained in section 2.4, a non-linear or a polynomial curve could be predicted to fit the data. Both Support vector regression (SVR) and Kernel ridge regression (KKR) use the same technique (a Radial basis kernel), except they differ in the

loss function used (epsilon sensitive loss vs ridge). The learned model by KKR is non-sparse, unlike SVR, thus takes more prediction time.

SVR and KRR are sensitive to hyperparameters, hence a grid search was performed to obtain the optimal hyperparameters. A very similar RSME score of 0.181 was obtained from both the models for the given dataset.

3.2.7 Random Forest Regressor

Random forests, which are an ensemble of decision tress, are very good at learning complex, highly non-linear relationships. They usually achieve high performance, better than polynomial counterparts. For this dataset, RMSE obtained with a random forest regressor was 0.109.

In addition, to its pros as a good supervised learning algorithm, the most interesting advantage of random forests, is ability to predict the feature importance. The feature importance graph in Fig. 3-2 shows, which features effect the learning performance of the model in ascending order. According to the graph, von mises stress is the most important features, which is natural and loading state is the least important.

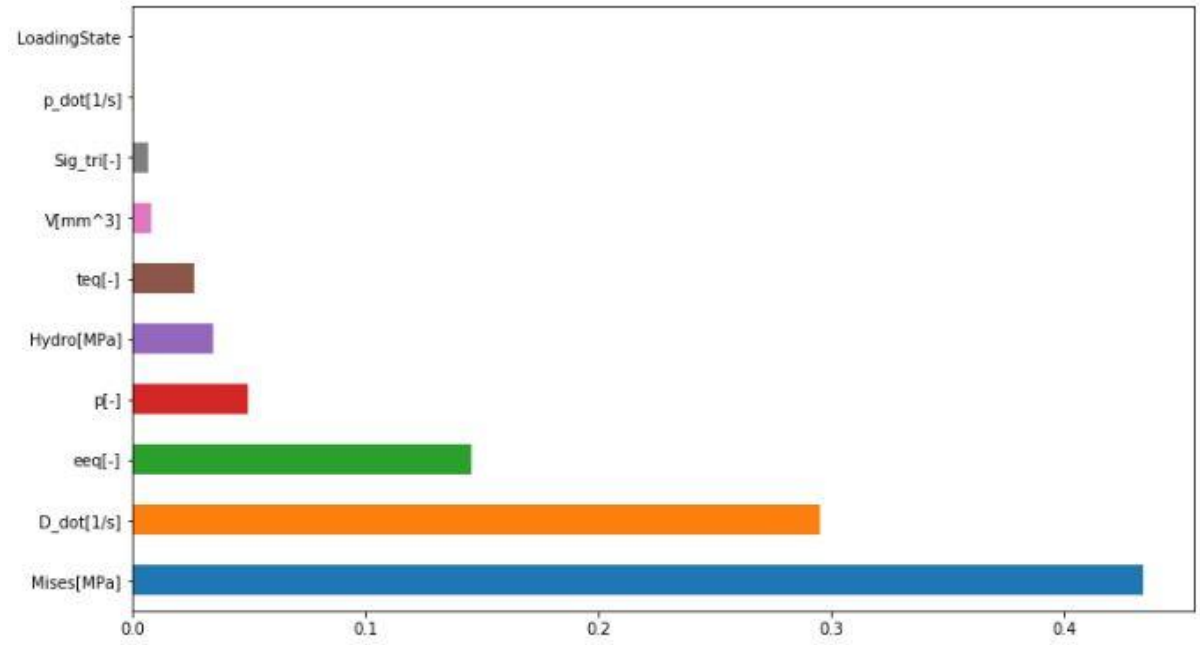


Fig. 3-2: Feature importance from Random forest regressor

3.2.8 Deep Neural Network

A multilayer perceptron with 300 hidden layers and activation ReLu was designed to fit a non-linear decision curve for the regression data. The RMSE obtained was 0.16

3.3 Results

As it is evident from the table below, it has been found that Random forest regressor is the best model. One cannot generalize this phenomenon to all datasets or the ones similar to this, as the "**no free lunch**" theorem in machine learning states, there is no one size that fits all.

Table 3-1: Results board with Random forest as best regressor

Machine learning model	RMSE
Linear Regression	0.241
BayesianRidge	0.241
LassoLars	0.172
ARDRegression	0.272
PassiveAgressiveRegressor	0.175
TheilSenRegressor	0.332
Support Vector Regressor	0.193
Kernel Ridge Regressor	0.181
Random Forest Regressor	0.109
Deep Neural Network	0.164

4 Stress hotspots prediction using machine learning

4.1 Motivation

Fracture is the breakage of a specimen under applied stress and Ductile fracture is the most common modes of failure. It starts by void formation, followed by void coalescence (also known as crack formation), then the crack propagates leading to the failure of the material. The external stress applied on a material is heterogeneously distributed creating regions of stress accumulations, also known ***stress hotspots*** [45]. To study the nucleation of this damage is desirable, as it helps us to understand the fracture, which ultimately defines the useful lifetime of a material.

Recently, Mangal [6] successfully applied machine learning techniques to study the impact of microstructural features on stress hotspots. Plastic deformation of single-phase face centered cubic (FCC) polycrystals is studied to ascertain the local microstructural characteristics related to the regions of high stress concentrations. Taking inspiration from this work, the dataset published is used and results of the machine learning model are reproduced. This lays a foundation to deploy these proven machine learning models with inhouse developed experimental and simulation data.

4.2 Using synthetic data from simulations

Under the influence of external stimuli such as load or heat, materials are usually tested in a non-destructive setup and their three-dimensional microstructure is characterized. High energy synchrotron X-rays are used in near field and far field High Energy X-Ray Diffraction Microscopy (nf-HEDM, ff-HEDM), which are widely implemented microstructure characterization techniques in the community of material science and engineering. They output mesoscale microstructural data like grain crystallography, centroids and strain fields. Machine learning techniques have been applied to this kind of data [2] to gain deeper insights. However, obtaining such datasets is non-trivial and expensive, instead using simulation generated dataset is the current norm [6]. Simulating the deformation of a material instead of doing physical experiments has the advantage of preserving both the initial and final structures, thus helping in correlating the final and initial states of the material. Here the uniaxial tensile deformation is simulated in a number of synthetic microstructures of a polycrystalline FCC material using an image

based full field crystal plasticity Elasto-Viscoplastic Fast Fourier Transform (EVPFFT) model [46].

For stress hotspot prediction, so called crystallographic and geometric microstructural descriptors have been identified in [6]. These were used as features for the machine learning model. Table 4.1 below lists the acronyms and descriptions of the features used. The dataset from this paper was highly imbalanced with 1:9 ratio between positive (hotspots) and negative examples (non-hotspots). Several machine learning techniques were applied to this dataset and results are discussed in the following section.

Table 4-1: Crystallographic and Geometric descriptors used for machine learning

Feature name	**Description**	**Feature name**	**Description**
Schmid	FCC Schmid factor	KernelAvg	Average misorientation within a grain
AvgMisorientations	Average misorientation between a grain and its nearest neighbour	FeatureBoundary ElementFrac	Fraction of grain touching the periodic boundary
Surface Features	1 if grain touches the periodic boundary else 0	001_IPF_0	Distance of loading direction from the <001> crystal direction in the inverse pole figure
Omega3s	3rd invariant of the second order moment matrix for the grain, without assuming a shape type	001_IPF_1	Distance of loading direction from the <110> crystal direction in the inverse pole figure
mPrimeList	Slip transmission factor for fcc materials	001_IPF_2	Distance of loading direction from the <111> crystal direction in the inverse pole figure

FeatureVolumes	Volume of grain	Surface area volume ratio	Ratio between surface area and volume of a grain
TJEuc	Average distance of a grain to triple junctions	Equivalent Diameters	Equivalent spherical diameter of a grain
GBEuc	Average distance of a grain to grain boundaries	AspectRatios	Ratio of axis lengths (ba and ca) for best-fit ellipsoid to grain shape
QPEuc	Average distance of a grain to quadruple junctions	NumNeighbors	Number of nearest neighbors of a grain
Neighborhoods	Number of grains having their centroid within the 1 multiple of equivalent sphere diameters from each grain		

4.3 Implementation details

All the machine learning models were implemented in python and run in JUPYTER notebooks. To ensure a fair comparison based on algorithm efficiency rather than on efficiency of the implementation or environment, all the experiments were carried out on a single machine configuration. The power of parallel processing and GPUs were utilized for very fast training of models. The environment was virtual machine running 64-bit Windows operating system on Google cloud platform (GCP) [47]. The configuration constituted 4 Intel Core Broadwell vCPUS with 18.0GB of installed RAM and NVIDIA Tesla K80 GPU.

4.4 Evaluation metrics

The aim is to find the best ML algorithm to predict the presence of a stress hotspot given a grain in the microstructure. As mentioned in previous sections, this project work focuses on binary classification problems. There are several well-known metrics to evaluate and use as comparison criteria while finding best models and their relative performance. shows a brief list.

Table 4-2: Popular metrics for classification tasks

S.No	Metrics for classification tasks
1	True Positive Rate (TPR) or Hit Rate or Recall or Sensitivity = TP / (TP + FN)
2	False Positive Rate (FPR) or False Alarm Rate = 1 - Specificity = 1 - (TN / (TN + FP))
3	Accuracy = (TP + TN) / (TP + TN + FP + FN)
4	Error Rate = 1 – accuracy or (FP + FN) / (TP + TN + FP + FN)
5	Precision = TP / (TP + FP)
6	F-measure: 2 / ((1 / Precision) + (1 / Recall))
7	ROC (Receiver Operating Characteristics) = plot of FPR vs TPR
8	AUC (Area Under the Curve)
9	Kappa statistics

For stress hotspot classification, predicting all the grains as normal (non-hotspots) will still result in a 90% classification accuracy as only 10% of the grains are hotspots. Two-dimensional confusion matrix indexed in one dimension by the true class and in the other by the predicted class serves as a better representation. A correctly predicted hot grain is true positive; a misclassified hot grain is a false negative. The normal grains predicted as hot are false positives, and the normal grains predicted normal are true negatives. It is desired that the classifier correctly predicts all hotspots, maximizing the true positives while minimizing the false positives. The receiving operator characteristic curve is the plot of false positives vs. true positives. The area under the receiving operator characteristic curve (AUC) is a good evaluation metric for such unbalanced datasets. For this task, the AUC parameter is used for comparison purposes, because of the imbalanced nature of the dataset.

4.5 Dealing with imbalanced dataset

Learning from imbalanced dataset is challenging, because the model could be biased towards the major class or is not capable of capturing the rare instances. An engineering example is detection of oil spills from satellite images. Since the instances of oil spills are

quite low compared to normal images, the model fails to generalize on new unseen data. To overcome this problem several techniques have been proposed. Only a couple of most popular methods are implemented in this project work and are briefly described below.

Random undersampling: In this technique the majority class samples are randomly selected and are removed from the data. This reduction is carried on until the imbalance is reduced. The problem with undersampling would be by removing the majority class samples, which would make the classifier to miss important phenomenon of the majority class.

Oversampling: In this sampling technique the minority class samples are replicated and added to the minority class in order to increase the number of minority class samples and thereby avoiding the imbalance. ADASYN is a widely used variant of oversampling technique.

ADASYN: The adaptive synthetic sampling approach for learning from imbalanced datasets uses a weighted distribution for different minority class examples according to their level of difficulty in learning. Then more synthetic data is generated for minority class examples that are harder to learn compared to those minority examples that are easier to learn. ADASYN reduces the bias introduced by class imbalance and adaptively shifts the classification boundary towards the difficult examples. [48] explains the algorithm in more detail.

4.6 Search for best ML model

In this section various machine learning models and their variants viz., Logistic regression, SVMs, Random forest, Neural network and XGBoost are run on the one instance of the dataset. A single RF model is trained on all the microstructures with different textures except the first microstructure in texture one. Then the AUC score is reported akin to k-fold cross validation by testing on that microstructure data not trained on. The model with best score is chosen and further iterations are done on the complete dataset to derive the total Mixed-model framework as explained in section 4.6.2.

4.6.1 Data pre-processing

The very first step is to import the dataset into Jupyter notebook.

1. Data I/O

```
In [8]: data=pd.read_csv('df_impfeatures_withid.csv')
        data.head()
```

Out[8]:

	001_IPF_0	001_IPF_1	001_IPF_2	100_IPF_0	100_IPF_1	100_IPF_2	111_IPF_0	111_IPF_1	111_IPF_2	AspectRatios_0	...	Omega3s	QPEuc	Schmid	S
0	0.314819	0.427569	0.274501	0.214385	0.460550	0.372547	0.255733	0.607882	0.669000	0.821072	...	0.801818	0.766849	0.459804	
1	0.216026	0.467399	0.384040	0.542158	0.444626	0.125266	0.060509	0.545659	0.526708	0.896180	...	0.715912	0.727803	0.497166	
2	0.117264	0.510865	0.470377	0.248346	0.467535	0.376097	0.441562	0.429924	0.196210	0.808131	...	0.861303	0.766598	0.478242	
3	0.039997	0.558397	0.546043	0.526644	0.429348	0.060637	0.427246	0.507653	0.383033	0.592307	...	0.000004	0.823149	0.455669	
4	0.463466	0.426146	0.160350	0.409262	0.415742	0.181440	0.178657	0.484543	0.420328	0.908052	...	0.824643	0.781541	0.450828	

5 rows × 29 columns

Fig. 4-1: First 5 rows of Mangal dataset for Classification

Then categorical columns like 'hotspot' need to be **one-hot encoded**, i.e. replace 'True' with 1 and 'False' with 0. Also, the missing values in the cells are replaced with 0. A plot of the frequency of occurrence of hotspots shows the highly imbalance nature of the dataset.

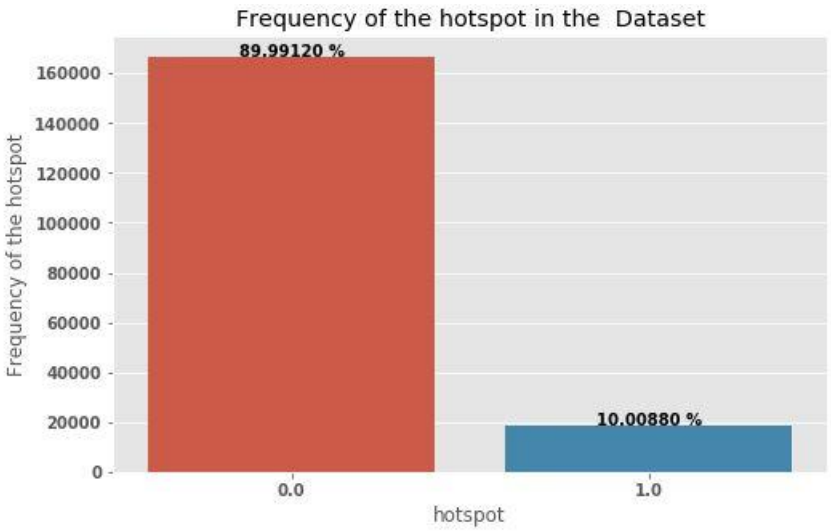


Fig. 4-2: Frequency of hotposts in the Mangal dataset

4.6.2 K-fold cross validation: Splitting the dataset

An ideal machine learning model should achieve good performance on the training data and fit well on test data. It is important to choose the correct way of splitting the dataset into train, test sets. Here, K-fold cross validation (CV) [49] technique is deployed. The dataset is partitioned into k subsamples. Then (k-1) subsamples are used to train the model, which is validated on the kth subsample. This process is repeated k times (the

folds), such that each fold is used exactly once for cross validation. The k results are then averaged to get the validation estimation.

The dataset from Mangal paper [6] is built on set of textures as shown in the figure below. For each texture kind, stochastically different datasets are created via multiple microstructure instantiations. The location of stress hotspots is affected by texture, geometry and the constitutive parameters. In this project the Mixed-model framework is designed, and the results are validated with the paper. This mode constitutes a single RF model is trained on all the microstructures with different textures, and the AUC score is reported using k-fold cross validation (average of validation performance on 2 randomly chosen microstructures from each texture kind).

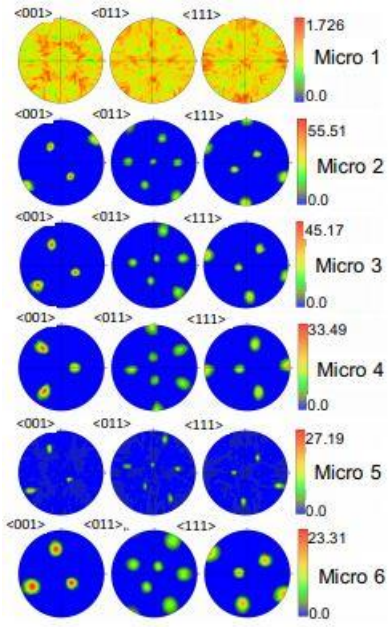


Fig. 4-3: Representative pole figures for six different FCC textures [6]

4.6.3 Logistic Regression

As a benchmark, a simple logistic regression model is used to classify the hotspot zones. The Area under curve obtained was 0.69. The imbalanced nature of the dataset could be seen from the confusion matrix, where true positive and false negative values are zeros.

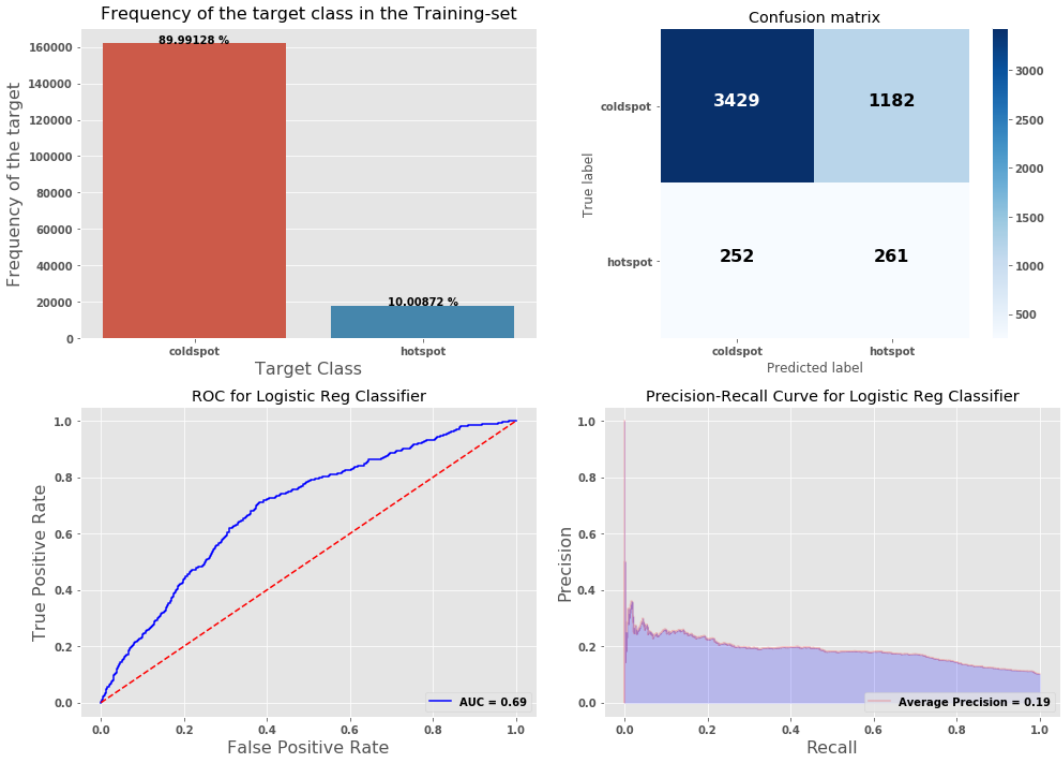


Fig. 4-4: Classification parameters- Logistic Regression

4.6.4 Logistic Regression with ADASYN

A logistic model with ADASYN is shown in Fig. 4-5. The frequency of coldspot and hotspot is now almost equivalent to 50% each. The dataset is balanced and can be seen from the confusion matrix. However, the AUC score hasn't improved and a value of 0.52 is obtained.

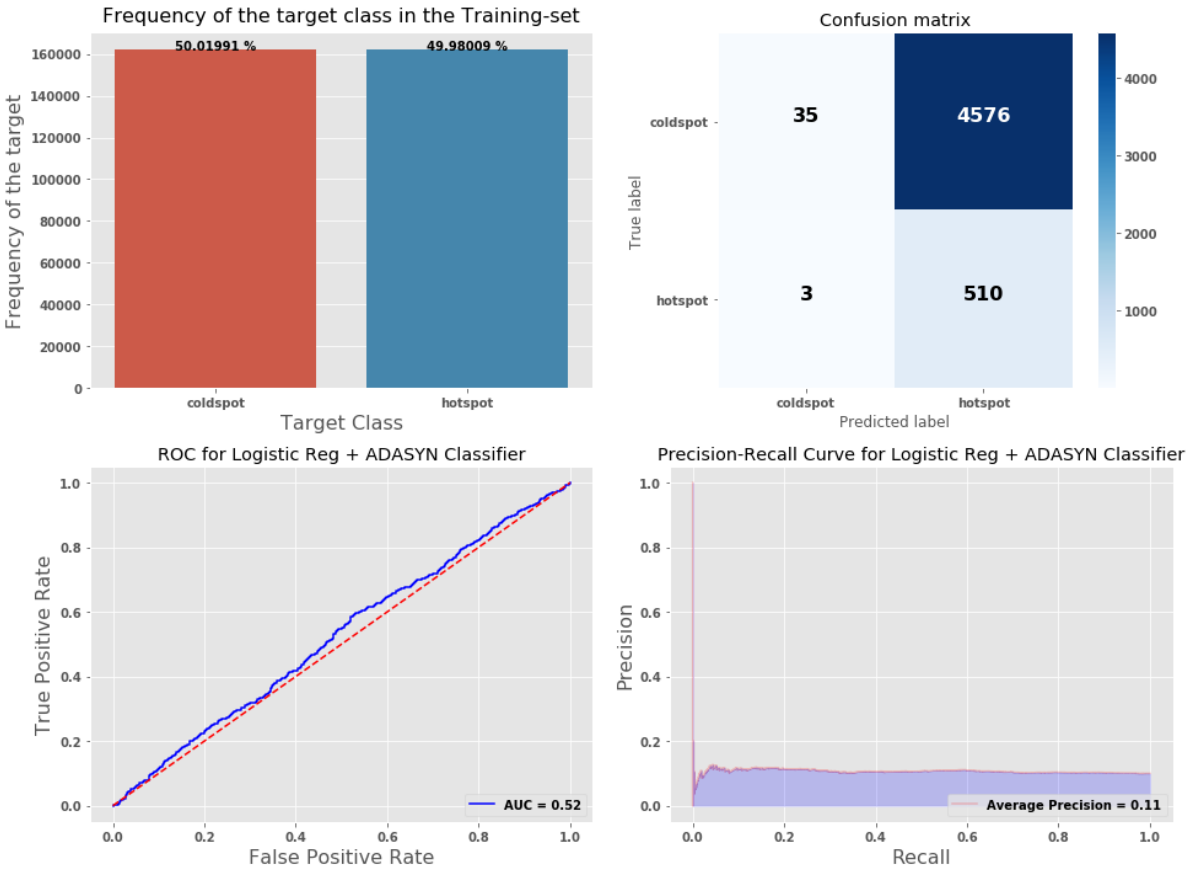


Fig. 4-5: Classification parameters- Logistic Regression with ADASYN

4.6.5 Logistic Regression with Undersampling

Another technique to deal with imbalanced datasets is used to random under sampling. Combined with logistic regression, AUC score obtained was 0.52. It is interesting to note that, although the frequency of hotposts and coldspots is 50% with random undersampling, the confusion matrix shows some imbalance.

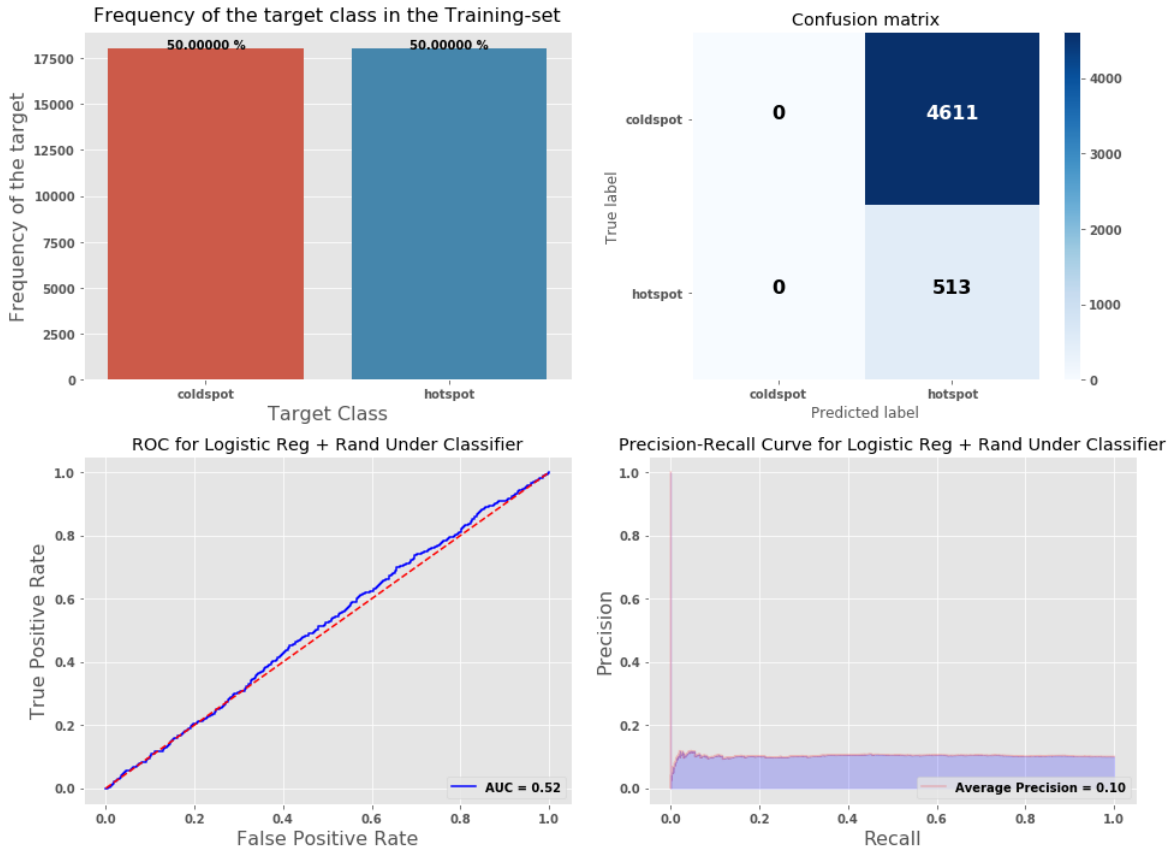


Fig. 4-6: Classification parameters- Logistic Regression with random undersampling

4.6.6 Random Forest Classifier

A random forest classifier with the same hyperparameters mentioned in the Mangal [6] was used. An AUC score of 0.72 was obtained.

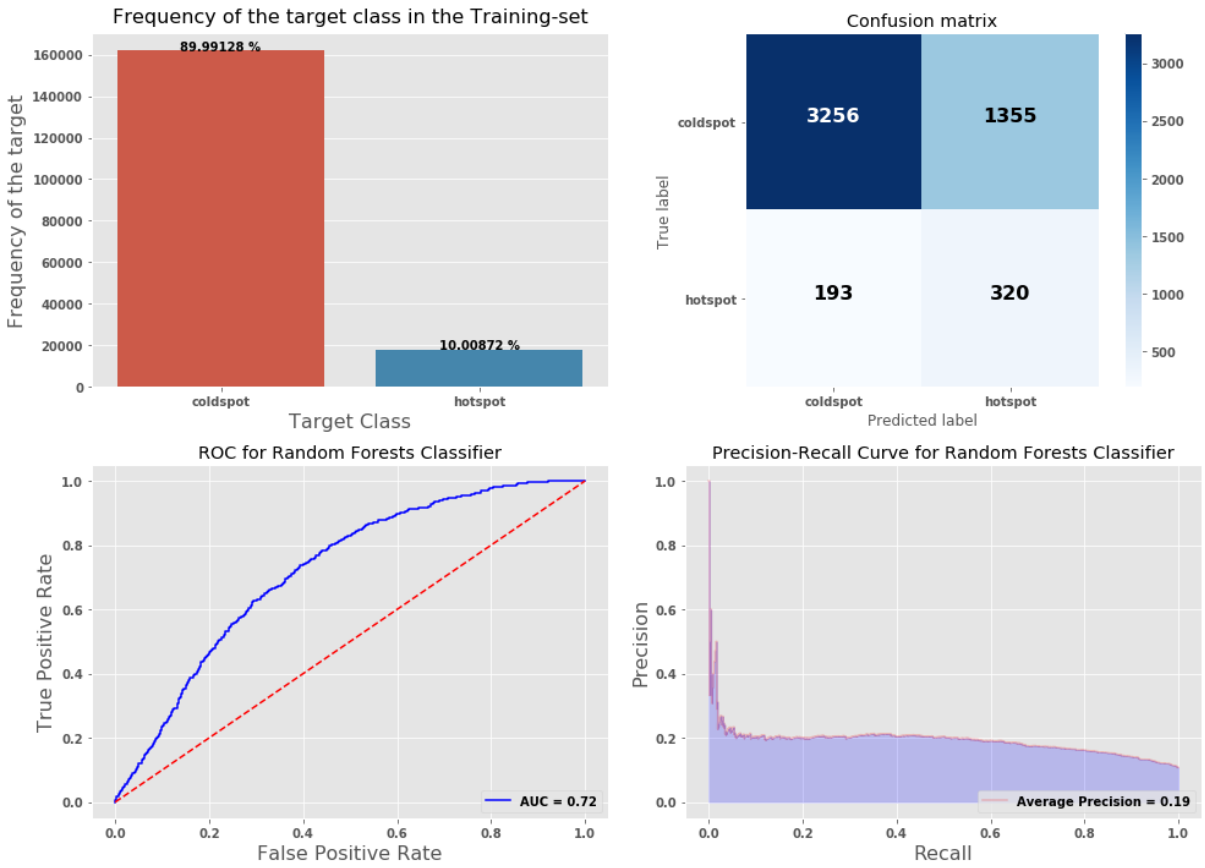


Fig. 4-7: Classification parameters- Random Forest Classifier

4.6.7 Random Forest Classifier with ADASYN

A Random forest classifier with ADASYN technique obtained an AUC score of 0.57.

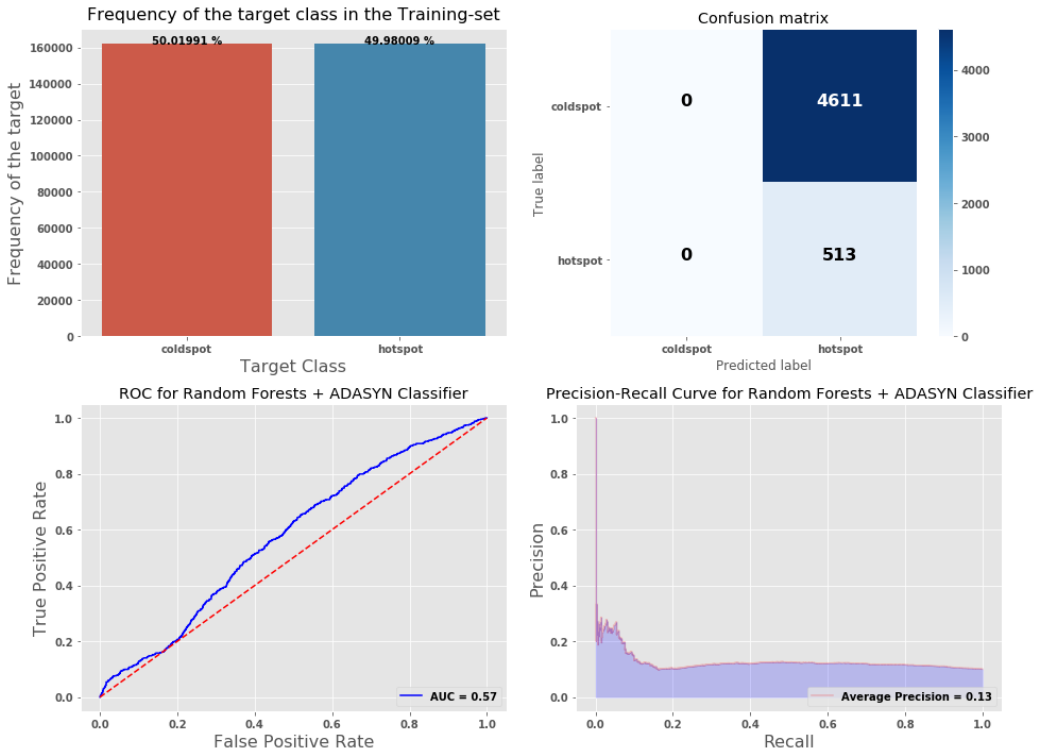


Fig. 4-8: Classification parameters- Random Forest Classifier with ADASYN

4.6.8 Random Forest Classifier with Undersampling

A Random forest classifier with random under sampling obtained an AUC score of 0.56. It is evident that, over sampling procedure like ADASYN produces slightly better result than undersampling, where the important information if lost.

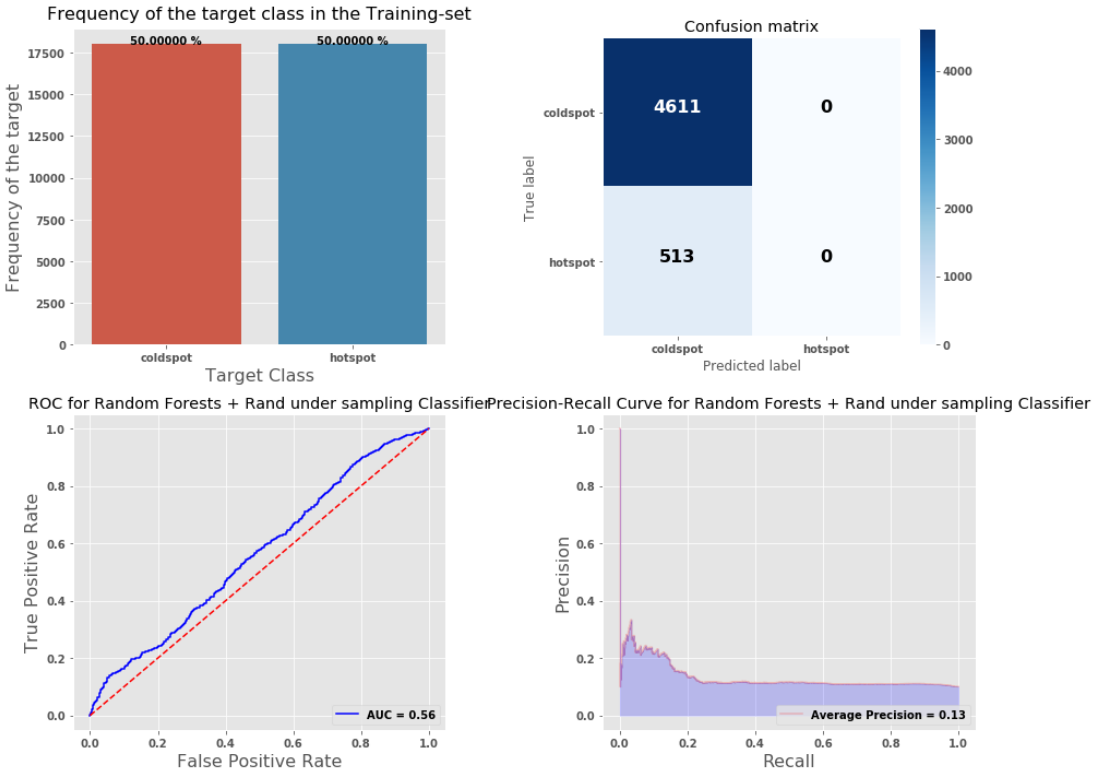


Fig. 4-9: Classification parameters- Random Forest Classifier with random undersampling

4.6.9 XGBoost Classifier

XGBoost classifiers usually deal better with bias in the datasets, which is evident from the confusion matrix below, however the AUC score of 0.71 is not better than Random forest for this problem.

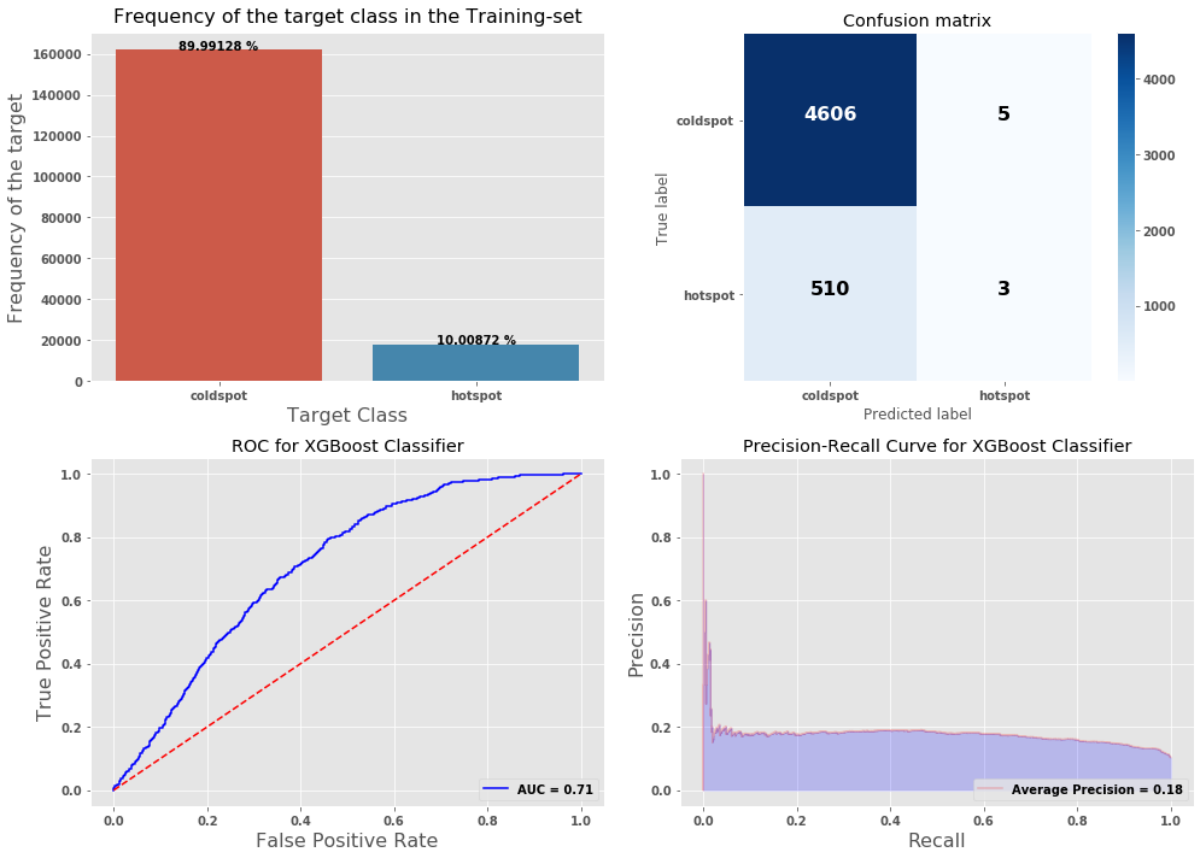


Fig. 4-10: Classification parameters- XGBoost classifier

4.6.10 XGBoost Classifier with ADASYN

With addition of oversampling technique like ADASYN gave no improvement in the AUC score, which was found to be 0.59.

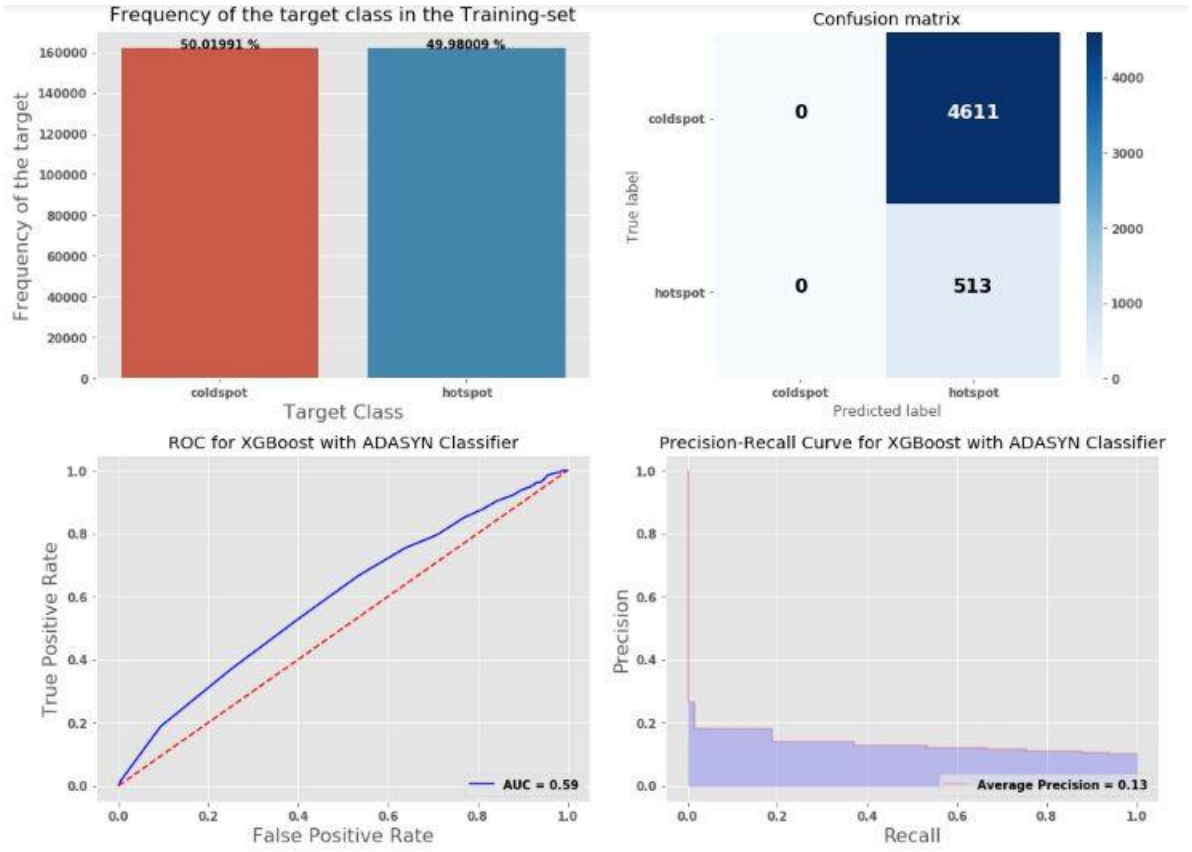


Fig. 4-11: Classification parameters- XGBoost classifier with ADASYN

4.6.11 XGBoost Classifier with Undersampling

As seen from the logistic and random forest models, undersampling has always worse compared to base model and oversampling, which is also the case for XGBoost classifier with an AUC score of 0.54.

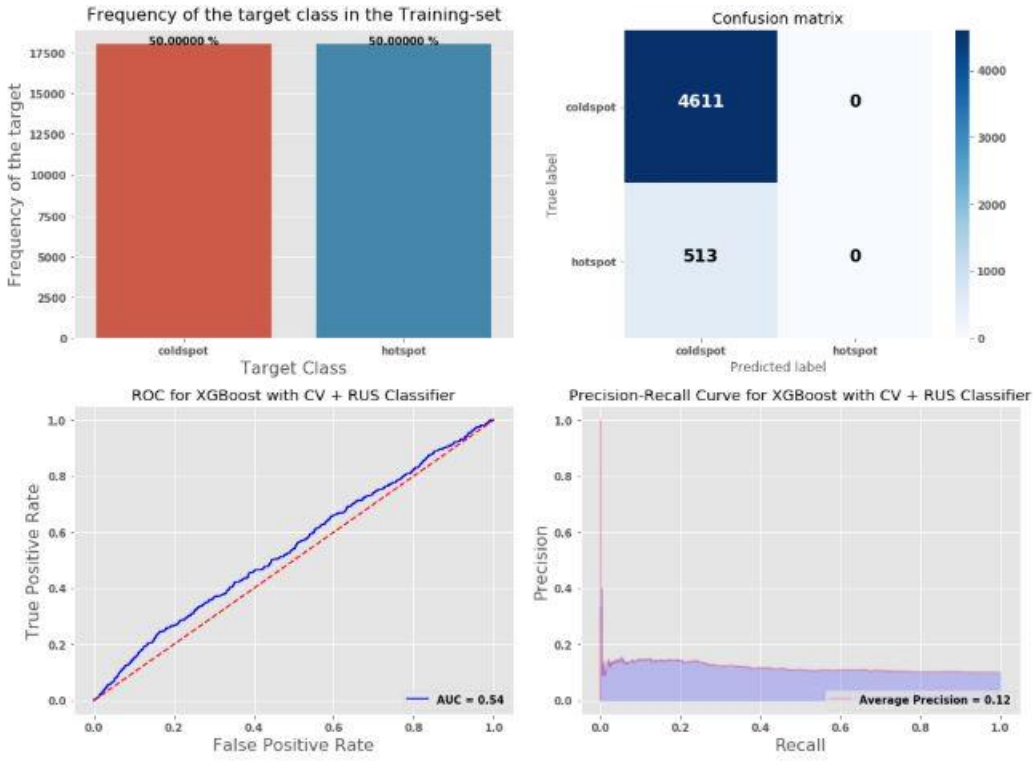
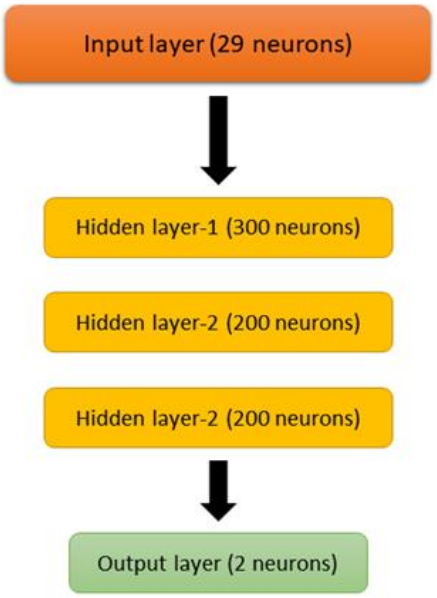


Fig. 4-12: Classification parameters- XGBoost classifier with random undersampling

4.6.12 Deep Neural Network

A 3-layer Neural network architecture was designed as shown in the figure below. Dropouts and batch-normalization layers were incorporated after each dense layer to reduce the overfitting. The AUC-score was found to be 0.54.



Layer (type)	Output Shape	Param #
dense_3 (Dense)	(None, 300)	8100
dropout_1 (Dropout)	(None, 300)	0
batch_normalization_1 (Batch	(None, 300)	1200
dense_4 (Dense)	(None, 200)	60200
dropout_2 (Dropout)	(None, 200)	0
batch_normalization_2 (Batch	(None, 200)	800
dense_5 (Dense)	(None, 200)	40200
dropout_3 (Dropout)	(None, 200)	0
batch_normalization_3 (Batch	(None, 200)	800
dense_6 (Dense)	(None, 2)	402

Total params: 111,702
Trainable params: 110,302
Non-trainable params: 1,400

Fig. 4-13: Deep neural network architecture

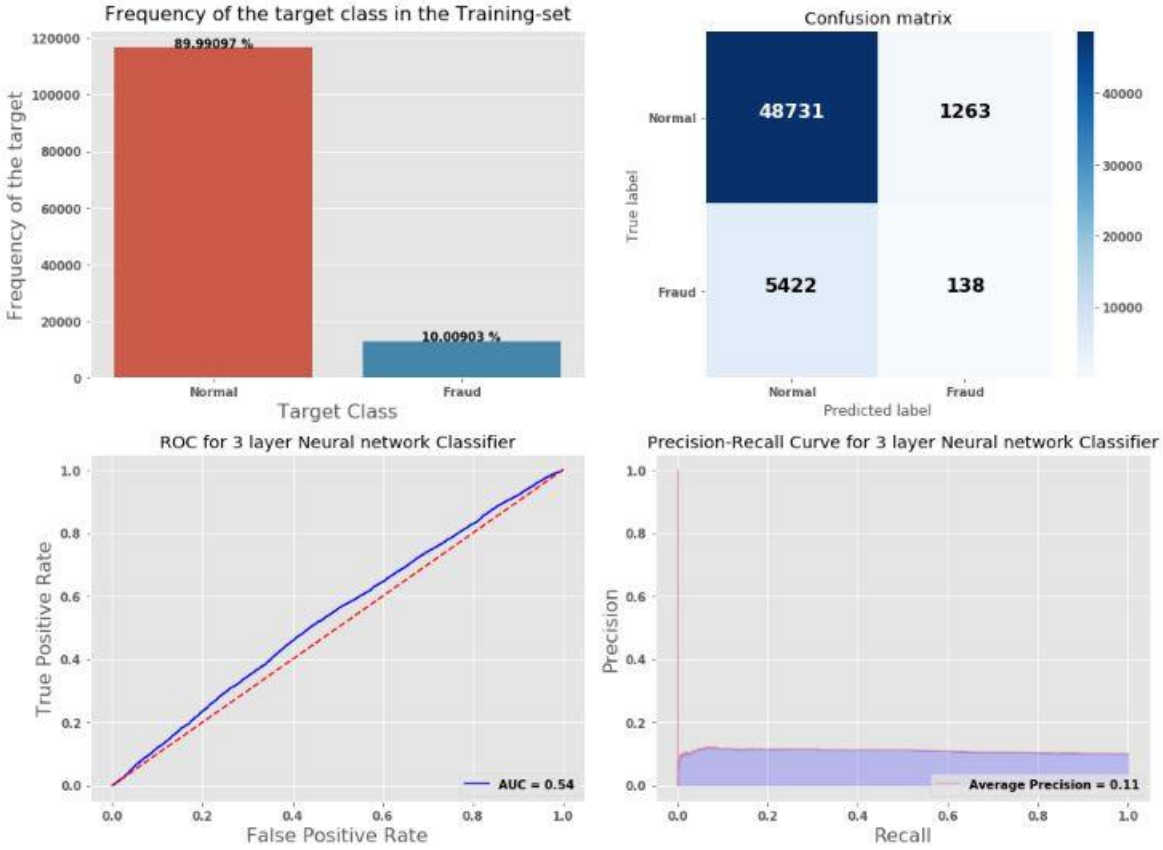


Fig. 4-14: Classification parameters- Deep neural network

4.6.13 Results

The Random forest classifier which is an ensemble method (section 2.5) with bootstrap aggregation of the data had better results when compared to all the other classifiers (Table 4-3). Similar conclusions were made in the [6]. In the next section, Random forest classifier is run with K-fold cross validation on the complete dataset spanning all the microstructures and textures.

Table 4-3: Results board with Random Forest as best classifier

Models	**Metrics**	**AUC**		
	Strategy	**Regular**	**Under-sampled**	**Oversampled (ADASYN)**
Logistic Regression		0.69	0.52	0.52
Random Forest		0.72	0.57	0.56
XGBoost		0.71	0.59	0.54
Neural Network		0.54	-	-

4.7 Comparing results with literature

The results of Mixed-model framework with K-fold cross validation for a Random forest classifier are presented in the table. The average of two highest AUC scores per textures is taken. These values match/are better in some cases when compared with the results presented in [6].

Table 4-4: K-fold cross validation for Mangal dataset

Texture	**AUC score**						**Avg. AUC**
	Micro 1	**Micro 2**	**Micro 3**	**Micro 4**	**Micro 5**	**Micro 6**	
1	0.72	0.72	0.70	0.74	0.72	0.71	0.73
2	0.78	0.73	0.73	0.78	0.72	0.75	0.78
3	0.69	0.62	0.68	0.63	0.66	0.68	0.68
4	0.68	0.81	0.79	0.79	0.71	0.73	0.8
5	0.73	0.76	0.78	0.77	0.77	0.75	0.775
6	0.70	0.75	0.77	0.73	0.74	0.71	0.76

5 Conclusions

The central idea of this project work was to use machine learning (ML) models to extract information from microstructural features obtained from simulation schemes, which have several advantages over conducting physical experiments. First step in this direction was to do extensive survey of ML models for regression and classification tasks. Then the efficacy of these algorithms for the given tasks was evaluated with a regression task for damage prediction and classification task for stress hotspot prediction.

The necessary theoretical background for machine learning model was provided in chapter 2. Supervised learning algorithms viz. Linear regression, logistic regression, SVMs, Tree based methods, Artificial neural networks and Ensemble methods were laid out in detail.

Chapter 3 focuses on damage prediction using regression models. 80% of 3000 examples with nine microstructural features obtained from simulations and their damage values was used as training set. After applying several regression ML models to predict the damage, it was concluded that Random forest regressor was the best with a RMSE of 0.109.

The aim of chapter 4 was to predict the location of stress hotspots in FCC materials under simulated plastic deformation and reproduce classification results of Mangal. A gentle introduction to the synthetic dataset involved and the microstructural features used was given. Various ML models were tested on this dataset, and random forest classifier was found to give highest average AUC on Mixed-model framework with K-fold cross validation for all the different microstructure configurations.

In conclusion, this project lays a foundation for future assignments, where the machine learning models could be directly adapted to similar datasets obtained from corresponding simulation setups with slight modifications to data pre-processing and hyperparameters.

References

[1] K. Rajan, “Materials Informatics: The Materials ‘Gene’ and Big Data,” *Annu. Rev. Mater. Res.*, vol. 45, no. 1, pp. 153–169, Jul. 2015.

[2] A. Orme, I. Chelladurai, T. Rampton, D. T. Fullwood, A. Khosravani, M. Miles, and R. K. Mishra, *Insights into twinning in Mg AZ31: A combined EBSD and machine learning study*, vol. 124. 2016.

[3] L. M. Ghiringhelli, J. Vybiral, E. Ahmetcik, R. Ouyang, S. V Levchenko, C. Draxl, and M. Scheffler, “Learning physical descriptors for materials science by compressed sensing,” *New J. Phys.*, vol. 19, no. 2, p. 023017, Feb. 2017.

[4] B. A. Moore, E. Rougier, D. O’Malley, G. Srinivasan, A. Hunter, and H. Viswanathan, “Predictive modeling of dynamic fracture growth in brittle materials with machine learning,” *Comput. Mater. Sci.*, vol. 148, pp. 46–53, 2018.

[5] A. V. Fedorov and I. V. Shamanaev, “Crystal Structure Representation for Neural Networks using Topological Approach,” *Mol. Inform.*, vol. 36, no. 8, p. 1600162, Aug. 2017.

[6] A. Mangal and E. A. Holm, “Applied machine learning to predict stress hotspots I: Face centered cubic materials,” *Int. J. Plast.*, vol. 111, pp. 122–134, Dec. 2018.

[7] A. Mangal and E. A. Holm, “Applied machine learning to predict stress hotspots II: Hexagonal close packed materials,” *Int. J. Plast.*, Aug. 2018.

[8] B. Dai, C. Gu, E. Zhao, K. Zhu, W. Cao, and X. Qin, “Improved online sequential extreme learning machine for identifying crack behavior in concrete dam,” *Adv. Struct. Eng.*, vol. 22, no. 2, pp. 402–412, Jul. 2018.

[9] L. Liang, M. Liu, C. Martin, and W. Sun, “A deep learning approach to estimate stress distribution: a fast and accurate surrogate of finite-element analysis,” *J. R. Soc. Interface*, vol. 15, no. 138, p. 20170844, Jan. 2018.

[10] J. Jung, J. I. Yoon, H. K. Park, J. Y. Kim, and H. S. Kim, “An efficient machine learning approach to establish structure-property linkages,” *Comput. Mater. Sci.*, vol. 156, pp. 17–25, Jan. 2019.

[11] C. Kusche, T. Reclik, M. Freund, T. Al-Samman, U. Kerzel, and S. Korte-Kerzel, “High-resolution, yet statistically relevant, analysis of damage in DP steel using artificial intelligence,” Sep. 2018.

[12] J. Zhu, W. Zhang, and X. Li, “Fatigue damage assessment of orthotropic steel deck using dynamic Bayesian networks,” *Int. J. Fatigue*, vol. 118, pp. 44–53, Jan. 2019.

[13] O. Olesegun, R. Noraas, M. Giering, and N. Somanath, “Structural Material Property Tailoring Using Deep Neural Networks,” Jan. 2019.

[14] D. Fumo, “Types of Machine Learning Algorithms You Should Know.” [Online]. Available: https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861.

[15] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg, “Top 10 algorithms in data mining,” in *Knowledge and Information Systems*, 2008, vol. 14, no. 1, pp. 1–37.

[16] S. Raschka, “Machine learning – An introduction for programmers - JAXenter.” [Online]. Available: https://jaxenter.com/machine-learning-an-introduction-for-programmers-122135.html.

[17] T. Burr, “Pattern Recognition and Machine Learning,” *J. Am. Stat. Assoc.*, vol. 103, no. 482, pp. 886–887, 2008.

[18] A. Géron, “Training Models - Hands-On Machine Learning with Scikit-Learn and TensorFlow [Book],” .

[19] Duo Ding, Xiang Wu, J. Ghosh, and D. Z. Pan, “Machine learning based lithographic hotspot detection with critical-feature extraction and classification,” in *2009 IEEE International Conference on IC Design and Technology*, 2009, pp. 219–222.

[20] Y. Shen, “Loss functions for binary classification and class probability estimation,” *Diss. available from ProQuest*, Jan. 2005.

[21] T. M. (Tom M. Mitchell, *Machine Learning*. .

[22] C. Cortes and V. Vapnik, “Support-vector networks,” *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, Sep. 1995.

[23] V. N. Vapnik and S. Kotz, *Estimation of dependences based on empirical data*. Springer, 2006.

[24] C. J. C. Burges, “A Tutorial on Support Vector Machines for Pattern Recognition,” *Data Min. Knowl. Discov.*, vol. 2, no. 2, pp. 121–167, 1998.

[25] B. Schölkopf and A. J. Smola, *Learning with kernels : support vector machines, regularization, optimization, and beyond*. MIT Press, 2002.

[26] L. Rokach and O. Maimon, *Data mining with decision trees : theory and applications*. World Scientific, 2008.

[27] J. R. Quinlan, “Induction of decision trees,” *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, Mar. 1986.

[28] S. L. Salzberg, “C4.5: Programs for Machine Learning by J. Ross Quinlan. Morgan Kaufmann Publishers, Inc., 1993,” *Mach. Learn.*, vol. 16, no. 3, pp. 235–240, Sep. 1994.

[29] L. Breiman, J. H. (Jerome H. . Friedman, R. A. Olshen, and C. J. Stone, *Classification and regression trees*. .

[30] T. Hothorn, K. Hornik, and A. Zeileis, “Unbiased Recursive Partitioning: A Conditional Inference Framework,” *J. Comput. Graph. Stat.*, vol. 15, no. 3, pp. 651–674, Sep. 2006.

[31] R. O. Duda, P. E. (Peter E. Hart, and D. G. Stork, *Pattern classification*. Wiley, 2001.

[32] F. Rosenblatt, *The perceptron, a perceiving and recognizing automaton Project Para*. Buffalo NY: Cornell Aeronautical Laboratory, 1957.

[33] S. J. Russell, P. Norvig, J. F. Canny, J. M. Malik, and D. D. Edwards, *Artificial Intelligence A Modern Approach*. 1995.

[34] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. .

[35] N. Singh, “Artificial Neural Networks and Neural Networks Applications - XenonStack.” [Online]. Available: https://www.xenonstack.com/blog/data-science/artificial-neural-networks-applications-algorithms/.

[36] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 315–323.

[37] M. A. Nielsen, “Neural Networks and Deep Learning.” Determination Press, 2015.

[38] K. Hornik, “Approximation capabilities of multilayer feedforward networks,” *Neural Networks*, vol. 4, no. 2, pp. 251–257, Jan. 1991.

[39] T. G. Dietterich, “Ensemble Methods in Machine Learning,” Corvallis, Oregon, USA.

[40] L. Breiman, “Bagging Predictors,” *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, 1996.

[41] Y. Freund, Y. Freund, and R. E. Schapire, “Experiments with a New Boosting Algorithm,” *Proc. Thirteen. Int. Conf. Mach. Learn.*, pp. 148--156, 1996.

[42] T. M. Fragoso and F. L. Neto, “Bayesian model averaging: A systematic review and conceptual classification,” Sep. 2015.

[43] D. H. Wolpert, “Stacked generalization,” *Neural Networks*, vol. 5, no. 2, pp. 241–259, Jan. 1992.

[44] M. Boeff, “Micromechanical modelling of fatigue crack initiation and growth,” 2016.

[45] S. Qidwai, A. Lewis, and A. B. Geltmacher, *Using image-based computational modeling to study microstructure–yield correlations in metals*, vol. 57. 2009.

[46] R. A. Lebensohn, A. K. Kanjarla, and P. Eisenlohr, “An elasto-viscoplastic formulation based on fast Fourier transforms for the prediction of micromechanical fields in polycrystalline materials,” 2012.

[47] “Virtual Machine Instances | Compute Engine Documentation | Google Cloud.” [Online]. Available: https://cloud.google.com/compute/docs/instances/.

[48] H. He, Y. Bai, E. A. Garcia, and S. Li, “ADASYN: Adaptive synthetic sampling approach for imbalanced learning,” in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, 2008, pp. 1322–1328.

[49] R. Kohavi and R. Kohavi, “A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection,” pp. 1137--1143, 1995.