EP4130 Project
Optimising Predictive Techniques for Astrophysics: A Comparative Study on Data Imputation and Classification

M Dinesh Kumar
EP20BTECH11011

Abstract

A crucial issue in many scientific and engineering domains is the need to model existing data to uncover new data points, especially in fields like astrophysics, where direct measurements are often difficult. For generating informed decisions and predictions in a variety of disciplines, such as in physics, biology, economics, and engineering, the precision and reliability of statistical analysis are essential. The task of classification is also equally important, as it helps us better understand phenomena by defining the underlying patterns. In this report, we set out to fill in the missing parameters in JINAbase and classify them efficiently using existing data. To achieve this, a total of five statistical methods were investigated on both the taks in depth to identify the best approaches for each task.

1 Introduction

1.1 Data Imputation

Data imputation is the process of filling in missing or incomplete data within a dataset. This technique involves using observed data to estimate and replace missing values, allowing for a more complete dataset that can be used in statistical analysis or modeling. By applying imputation methods, the integrity of the data is preserved, enabling more accurate predictions and analyses on both the existing and new, unseen data.

1.2 Classification

Classification is the task of assigning a label or category to an input based on its features or characteristics. It is the process of training a model to recognise patterns in the input data and use them to assign the most appropriate class label to a new input.

2 Need for Imputation & Classification

In astrophysics, we deal with vast amounts of data collected from telescopes, satellites, and other instruments. This data helps us understand the universe, from the behavior of stars to the structure of galaxies. However, not all of this data is complete or easy to interpret. Some data points might be missing due to the limitations of our instruments, or they might be obscured by noise and interference:

- Handling missing data: Data imputation is crucial for filling in gaps in our astrophysical datasets, allowing us to maintain the integrity of the data. By accurately estimating and replacing missing values, we can ensure that our analyses and models are based on a complete dataset, leading to more reliable and meaningful conclusions.
- Classifying objects: Classification based on photometric magnitudes or stellar parameters allows us to systematically study different types of celestial objects, such as stars, galaxies, or quasars. This helps identify patterns and correlations in their properties, enhancing our understanding of the universe.
- Testing models: Data imputation is vital for testing and refining theoretical models in astrophysics. By ensuring that the dataset is complete and accurate, we can compare predictions with observations more effectively, assessing the accuracy of these models and improving our understanding of the underlying physical processes.

3 Predictive Techniques

There are several algorithms available for data imputation and classification in data science and statistics. Here are some of the most commonly used ones:

3.1 Random Forest Ensemble

It is an improvement on bagging that creates a set of de-correlated trees by picking input variables at random at each split and use bootstrap samples. Through the use of numerous tree averages, this strategy lowers the variation of predictions. Random Forests use the mean of individual tree's output for regression and a majority vote among the trees' predictions for classification. By reducing correlation between trees and increasing prediction accuracy, the technique outperforms standard bagging. Random forests are preferred because of their simplicity and robustness; they frequently match or outperform other techniques like boosting in terms of performance, particularly if trained with a large number of trees. [1]

Algorithm 1 Random Forest

```
1:  Input: Training data $\{(x_i, y_i)\}_{i=1}^N$, number of trees $B$, minimum node size $n_{min}$, number of variables $m$
2:  Output: Ensemble of trees $\{T_b\}_{b=1}^B$
3:  for $b = 1$ to $B$ do
4:      Draw a bootstrap sample $Z^*$ of size $N$ from the training data
5:      Grow a random forest tree $T_b$ to the bootstrapped data:
6:      for each node in the tree do
7:          if the node size is greater than $n_{min}$ then
8:              Select $m$ variables at random from the $p$ available variables
9:              Pick the best variable and split point among the $m$ variables
10:             Split the node into two daughter nodes
11:         end if
12:     end for
13: end for
14: Return the ensemble of trees $\{T_b\}_{b=1}^B$
15: Prediction:
16: if Regression then
        $$\hat{f}_{\text{rf}}(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$$
17: else if Classification then
        $$\hat{C}_{\text{rf}}(x) = \text{majority vote}\{\hat{C}_b(x)\}_{b=1}^B$$
18: end if
```

3.2 Support Vector Machine

These are versatile supervised learning algorithms that can be used in regression, classification, and outlier detection. Using support vectors, they find the hyperplane that maximises the space between classes in high-dimensional spaces. Support Vector while doing regression tries to fit training points within the hyperplanes that separates spaces. SVMs use a variety of kernel functions, including linear, polynomial, and RBF, to handle both linear and non-linear data. It is important to select the appropriate kernel and parameters, such as C and gamma. SVMs do better in high-dimensional spaces and when the number of features exceeds the number of samples. [2]

Algorithm 2 Support Vector Machines (SVM)

```
1:  Input: Training data $\{(x_i, y_i)\}_{i=1}^N$, regularisation parameter $C$, kernel function $k(\cdot,\cdot)$, tolerance $\epsilon$
2:  Output: Model parameters $\{\alpha_i\}_{i=1}^N$, $\beta$
3:  Compute the kernel matrix $K$ where $K_{ij} = k(x_i, x_j)$ for all $i, j$
4:  Initialise $\alpha_i$ and $\beta$ to zero for all $i$
5:  repeat
6:      for each training sample $(x_i, y_i)$ do
7:          Compute the prediction $\hat{y}_i = \sum_{j=1}^N \alpha_j k(x_j, x_i) + \beta$
8:          Compute the error $e_i = y_i - \hat{y}_i$
9:          if $|e_i| > \epsilon$ then
10:             Update $\alpha_i$ and $\beta$ based on the error $e_i$ and the regularisation parameter $C$
11:             Adjust $\alpha_i$ and $\beta$ to minimise the loss function
12:         end if
13:     end for
14: until Convergence criteria are met or maximum number of iterations is reached
15: Return model parameters $\{\alpha_i\}_{i=1}^N$, $\beta$
16: Prediction:
17: if Regression then
        $$\hat{f}_{\text{svm}}(x) = \sum_{i=1}^{N} \alpha_i k(x_i, x) + \beta$$
18: else if Classification then
        $$\hat{C}_{\text{svm}}(x) = \text{sign}\left(\sum_{i=1}^{N} \alpha_i k(x_i, x) + \beta\right)$$
19: end if
```

3.3 k-Nearest Neighbours (kNN)

kNN is a simple yet effective method for regression and classification. It takes the average of these closest points' results to produce a prediction. By averaging the values of a data point's k-nearest neighbors, KNN predicts a new data point's value in regression. By averaging the labels of a class's k-nearest neighbors, KNN calculates the likelihood of a class for classification. After that, the class is guessed using a threshold, usually 0.5. This means that if the averaged probability is 0.5 or higher, the point is classified into a positive class else into a negative class. This method works well for a variety of problems. [3]

Algorithm 3 k-Nearest Neighbors (k-NN)

```
1:  **Input:** Training data $\{(x_i, y_i)\}_{i=1}^N$, number of neighbors $k$, distance metric $d$
2:  **Output:** Predicted values for new data points
3:  **for** each new data point $\mathbf{x}_0$ **do**
4:      Compute the distance $d(\mathbf{x}_0, \mathbf{x}_i)$ between $\mathbf{x}_0$ and each training point $\mathbf{x}_i$
5:      Sort the training points by increasing distance to $\mathbf{x}_0$
6:      Select the $k$ nearest neighbors: $\{\mathbf{x}_{\text{NN}(\mathbf{x}_0,j)}\}_{j=1}^k$
7:      **if** Regression **then**
8:          Compute the predicted value $\hat{y}_0$ as the average of the $y$ values of the $k$ nearest neighbors:
                $\hat{y}_0 = \frac{1}{k}\sum_{j=1}^{k} y_{\text{NN}(\mathbf{x}_0,j)}$
9:      **else if** Classification **then**
10:         Compute the predicted probability $\hat{p}_0$ as the average of the labels of the $k$ nearest neighbors:
                $\hat{p}_0 = \frac{1}{k}\sum_{j=1}^{k} y_{\text{NN}(\mathbf{x}_0,j)}$
11:         Threshold the probability to determine the predicted class:
                $\hat{c}_0 = \begin{cases} 1 & \text{if } \hat{p}_0 \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$
12:     **end if**
13: **end for**
14: **Return** the predicted values for all new data points
```

3.4 XGBoosting

XGBoost, or Extreme Gradient Boosting, enhances traditional gradient boosting by optimising speed and performance. It builds an ensemble of decision trees, where each tree corrects errors made by the previous ones. For regression, XGBoost minimises residuals using gradient descent to improve predictions iteratively. For classification, it adjusts predictions to maximise the likelihood of correct class assignments, often using logistic loss for binary classification or softmax for multi-class problems. XGBoost's regularisation, handling of missing values, and flexibility with objective functions make it highly effective for both regression and classification tasks. [4]

Algorithm 4 XGBoost Algorithm

```
1:  **Input:** Training data $\{(x_i, y_i)\}_{i=1}^N$, number of boosting rounds $T$, learning rate $\eta$, set of base learners $\{h_t\}_{t=1}^T$
2:  **Output:** A strong predictor
3:  Initialise the model $f_0(x) = 0$
4:  **for** $t = 1$ to $T$ **do**
5:      Compute the residuals $r_i^{(t)} = y_i - f_{t-1}(x_i)$ for $i = 1, \ldots, N$
6:      Fit a base learner $h_t(x)$ to the residuals $r_i^{(t)}$:
            $h_t(x) = \arg\min_h \sum_{i=1}^{N} \text{loss}(r_i^{(t)}, h(x_i))$
7:      Compute the step sise $\gamma_t$:
            $\gamma_t = \arg\min_\gamma \sum_{i=1}^{N} \text{loss}(y_i, f_{t-1}(x_i) + \gamma h_t(x_i))$
8:      Update the model:
            $f_t(x) = f_{t-1}(x) + \eta\gamma_t h_t(x)$
9:  **end for**
10: **Prediction:**
11: **if** Regression **then**
            $\hat{y}(x) = f_T(x)$
12: **else if** Classification **then**
            $\hat{y}(x) = \text{argmax}_k\,(f_T(x))$
13: **end if**
```

3.5 Multi-Layer Perceptron (MLP)

To increase complexity in function computation, a MLP joins numerous simple processing units to form a neural network. Linear models are not able to capture relations such as XOR, while MLPs consist of multiple layers of units with non-linear activation functions. In regression, MLPs learn the data and produce a value, while in classification MLPs determine the class of the output of the units. Linear models benefit more from MLPs, as they are less rigid by conquering complex patterns based on hidden layers and therefore make better predictions and perform more tasks. [5]

Algorithm 5 Multi-Layer Perceptron (MLP)

```
1: **Input:** Training data $\{(x_i, y_i)\}_{i=1}^{N}$, learning rate $\alpha$, number of epochs $E$, number of hidden units $H$
2: **Output:** MLP model with parameters $(W, b)$
3: Initialise weights $W^{(1)}$, $W^{(2)}$, biases $b^{(1)}$, $b^{(2)}$ randomly
4: **for** epoch $e = 1$ to $E$ **do**
5:     **for** each training sample $(x_i, y_i)$ **do**
6:         **Forward pass:**
7:         Compute hidden layer activations: $h_i = \sigma(W^{(1)}x_i + b^{(1)})$ where $\sigma$ denotes a non-linear activation function
8:         Compute output layer: $o_i = W^{(2)}h_i + b^{(2)}$
9:         Compute loss: $L_i = \frac{1}{2}(o_i - y_i)^2$
10:        **Backward pass:**
11:        Compute gradients of the loss with respect to weights and biases
12:        Update weights and biases:
               $W^{(1)} \leftarrow W^{(1)} - \alpha \nabla_{W^{(1)}} L_i$
               $W^{(2)} \leftarrow W^{(2)} - \alpha \nabla_{W^{(2)}} L_i$
               $b^{(1)} \leftarrow b^{(1)} - \alpha \nabla_{b^{(1)}} L_i$
               $b^{(2)} \leftarrow b^{(2)} - \alpha \nabla_{b^{(2)}} L_i$
13:    **end for**
14: **end for**
15: **Return** trained MLP model with parameters $(W, b)$
16: **Prediction :**
        $F(x) = \sum_{i=1}^{N} (v_i + b_i) \cdot \sigma(w_i^T x + b_i)$
```

4 Dataset

4.1 Description

JINAbase is a compilation of the properties of metal-poor stars from the literature. The primary goal of JINAbase is to provide chemical abundance data as obtained from the various studies in the literature in one place for easy access. Accordingly, there naturally are limitations as to what JINAbase can provide, as there are multiple intermediate measurements and steps individual to every study that lead to these abundances. [6]

Data from JINAbase include :

- **Priority label:** to choose from stars with multiple entries.
- **Evolutionary status of star:** RG: Giants, SG: Subgiants, HB: Horizontal branch and MS: Main sequence.
- **Carbon enhancement:** CEMP: Carbon-rich & neutron-capture-rich, CEMP-no: Carbon-rich but not neutron-capture-rich. C-rich stars with [C/Fe] $>$0.7.
- **Neutron-capture element enhancement:** r-I, r-II, s-rich, i-rich, or r+s stars.
- **Location:** Halo (HA), bulge (BU), ultra-faint dwarf galaxy (UF), or classical dwarf galaxy (DW).
- **Position:** Right ascension & Declination.
- **Radial Velocity:** Radial velocity & reference for radial velocity
- **Photmetric Magnitudes:** Values for U, B, V, R, I, J, H, K.
- **Stellar parameters:** Effective temperature, surface gravity, metallicity, microturbulence.
- **Chemical Abundances:** $log_\epsilon(X)$, abundances for elements from Li to U, when available & (Ca, Ti, Cr, Mn, Fe), when available.

4.2 Pre-Processing & Approaches

Data preprocessing involves cleaning, transforming, and organising raw data to improve its quality and make it suitable for analysis. This phase includes handling, reduction, normalisation, encoding, and detection of missing values. Preprocessing methods we used are listed:

4.2.1 Data Cleaning

This step involves identifying and removing missing, inconsistent, or irrelevant data like removing/filling in null values, or handling outliers. Below is the count of null values for each column.

```
data.isnull().sum(axis = 0)
```

```
Sci_key        0   R_mag        873
C_key          0   I_mag       1146
Loc            0   J_mag        408
Type           0   H_mag        408
RA             4   K_mag        391
DEC            4   Teff           0
Vel          372   logg           0
U_mag       1773   Fe/H           0
B_mag        357   Vmic         107
V_mag        329   Ba/Fe          0
```

Null values were dealt with in 2 different ways:

- The rows with null microturbulence (Vmic) values were extracted for our problem statement.
- The remaining null values were filled with zeroes.

Note: Initially, we considered modeling radial velocity (Vel) as one of our objectives. However, after performing exploratory data analysis (EDA), we realized that it was independent of all other variables. As a result, we decided to drop this idea since it could not be effectively modeled with the available data.

From the EDA, we found that surface gravity $[logg]$, effective temperature $[Teff]$, and microturbulence $[Vmic]$ had a strong correlation and were likely the most useful variables for our objectives. These variables were used to identify and remove outliers. Outliers were eliminated using the interquartile range (IQR), with any data points falling outside the 25th to 75th percentile considered outliers and removed.

4.2.2 Data Transformation

This step involves converting the data into a format that is more suitable for our models to learn & interpret. This can include normalising numerical data, creating dummy variables, and encoding categorical data. We normalised our dataset using StandardScaler and encoded categorical variables with LabelEncoder.

4.2.3 Dimensionality Reduction

This step involves selecting a subset of data that is relevant to our task through feature selection, extraction, or reduction. We chose columns that showed strong correlation with the target variable, particularly effective temperature $[Teff]$ and surface gravity $[logg]$. Initially, we considered Principle Component Analysis (PCA) for dimensionality reduction. Our exploratory data analysis revealed that t-Distributed Stochastic Neighbor Embedding (t-SNE) would be better suited for capturing the non-linear relationships in our data compared to traditional linear methods.

Theory of t-SNE: t-SNE is a technique for dimensionality reduction that focuses on preserving the local structure of the data. It transforms the similarities between data points in high-dimensional space into probabilities, representing how likely points are to be neighbors. The method then creates a low-dimensional map where these probabilities are preserved, revealing patterns such as clusters. Unlike linear methods like PCA, t-SNE excels in capturing non-linear relationships by optimizing the difference between the two probability distributions in high- and low-dimensional spaces. [7]

Algorithm 6 Simple version of t-SNE

```
1: Data: data set $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$,
2: Cost function parameters: perplexity $Perp$
3: Optimization parameters: number of iterations $T$, learning rate $\eta$, momentum $\alpha(t)$
4: Result: low-dimensional data representation $\mathcal{Y}^{(T)} = \{y_1, y_2, \ldots, y_n\}$.
5: begin
6: compute pairwise affinities $p_{j|i}$ with perplexity $Perp$ (using Equation 1)
7: set $p_{ij} = \frac{p_{i|j}+p_{j|i}}{2n}$
8: sample initial solution $\mathcal{Y}^{(0)} = \{y_1, y_2, \ldots, y_n\}$ from $\mathcal{N}(0, 10^{-4}I)$
9: for $t = 1$ to $T$ do
10:     compute low-dimensional affinities $q_{ij}$ (using Equation 4)
11:     compute gradient $\frac{\delta C}{\delta y_i}$ (using Equation 5)
12:     set $\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta\frac{\delta C}{\delta y_i} + \alpha(t)(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)})$
13: end for
14: end
```

4.3 Exploratory Data Analysis

In Data Science, Exploratory Data Analysis (EDA) is an approach for analyzing data sets to summarize their main characteristics, often using statistical graphics and other data visualization techniques. While a statistical model can be used, EDA primarily focuses on understanding the data beyond formal modeling, contrasting traditional hypothesis testing. Simply put, EDA is the process of examining and analyzing data to uncover its main characteristics, patterns, relationships, and outliers. It serves as a preliminary step to gain deeper insights into the data and to guide further analysis.[8] EDA is useful for several reasons:

- It helps in identifying data quality issues such as missing values, outliers, and inconsistencies.
- It provides insights into the distribution of variables, their central tendency, and variability, aiding in understanding the data and selecting appropriate statistical methods.
- It can uncover patterns and relationships between variables, helping in generating hypotheses and understanding the underlying mechanisms of the data.
- It helps in identifying potential biases, limitations, or errors in the data, which may affect the validity of the analysis.
- EDA can assist in identifying the need for additional data collection or feature engineering, guiding the enhancement of the dataset for more robust analysis.
- It enables better decision-making in the data preprocessing stage, helping to select appropriate transformations, handling of missing values, and outlier treatment.
- EDA can also highlight any unusual trends or unexpected behaviors in the data that may require further investigation before proceeding with modeling.

4.3.1 Summary Statistics

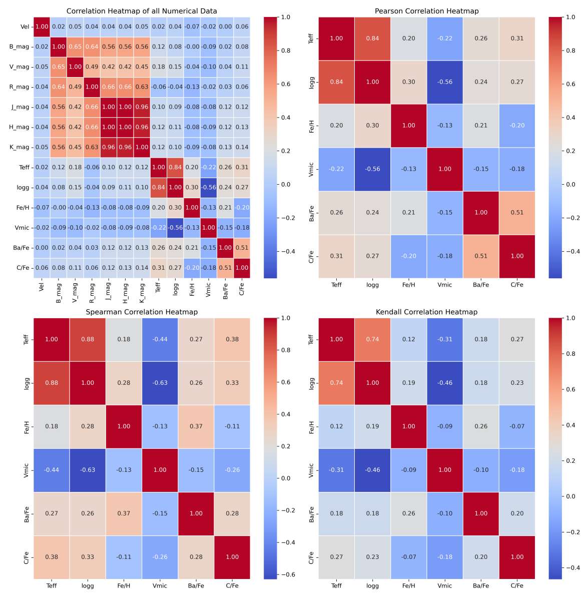
1. **Descriptive Statistics:** Measures of central tendency (mean, median, mode) and variability (variance, standard deviation, interquartile range) provide insights into the center and spread of the data.
2. **Correlation Coefficients:** Metrics like Pearson's r, Spearman's ρ, and Kendall's τ measure the strength and direction of relationships between pairs of variables. These coefficients help identify both linear and non-linear correlations, revealing how variables may be interrelated and guiding the choice of further analytical techniques.

4.3.2 Visualisations

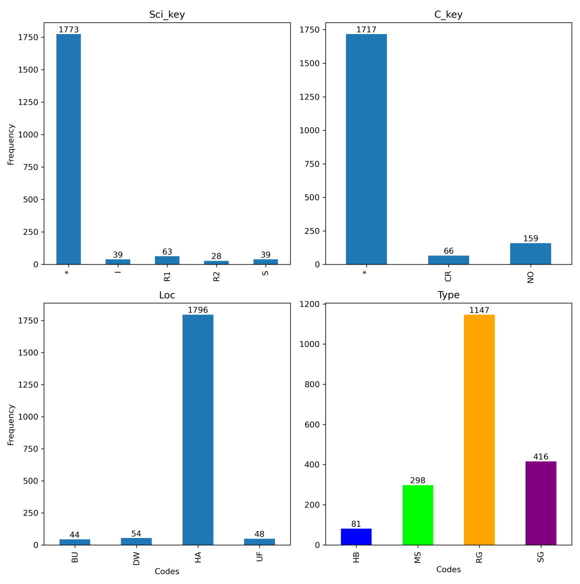
1. **Histograms:** Illustrate the distribution of a continuous variable, showing how data points are spread across intervals.
2. **Box Plots:** Display median, quartiles, and outliers, summarising the distribution and variability of a variable and allowing comparisons across categories.
3. **Violin Plots:** Combine box plot and density plot elements, showing the distribution and probability density of a variable across categories, offering a more detailed view of data distribution.
4. **Scatter Plots:** Illustrate relationships between two continuous variables, helping to identify trends, correlations, and outliers.
5. **Bar Charts:** Represent categorical data, showing the frequency or proportion of each category for easy comparison.
6. **Heatmaps:** Heatmaps use color gradients to represent data values, often used for visualising correlation matrices or patterns in missing data. This tool is effective for quickly identifying areas of high or low concentration within the data.
7. **Pair Plots:** Pair plots offer a matrix of scatter plots, displaying pairwise relationships between multiple variables.

	Vel	B_mag	...	K_mag	Teff	logg	Fe/H	Vmic	Ba/Fe
Mean	4.90	10.82	...	8.70	5349.92	2.56	-2.56	1.73	-0.16
Std	137.95	5.66	...	4.95	696.33	1.25	0.79	0.49	0.84
Min	-1164.00	0.00	...	0.00	3900.00	-0.30	-7.30	0.00	-2.55
25%	-67.17	8.85	...	6.04	4810.00	1.50	-3.03	1.45	-0.65
50%	0.00	13.37	...	10.64	5186.50	2.44	-2.68	1.65	-0.23
75%	79.45	14.85	...	12.64	5900.00	3.73	-2.20	2.00	0.09
Max	488.80	22.60	...	16.78	7950.00	5.00	-0.03	4.50	3.44

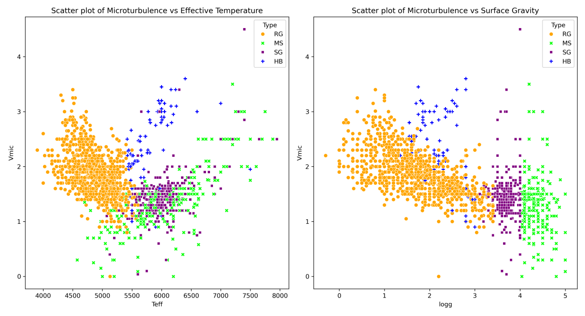
Descriptive statistics for various stellar and photometric parameters, including velocity (Vel), magnitudes in different bands (B_{mag} ... K_{mag}), effective temperature (T_{eff}), surface gravity ($logg$), metallicity ($[Fe/H]$), microturbulence velocity ($Vmic$), and elemental abundance ratios ($[Ba/Fe]$ and $[C/Fe]$). The statistics include the mean, standard deviation (std), minimum (min), 25th percentile (25%), median (50%), 75th percentile (75%), and maximum (max) values across 1942 observations.



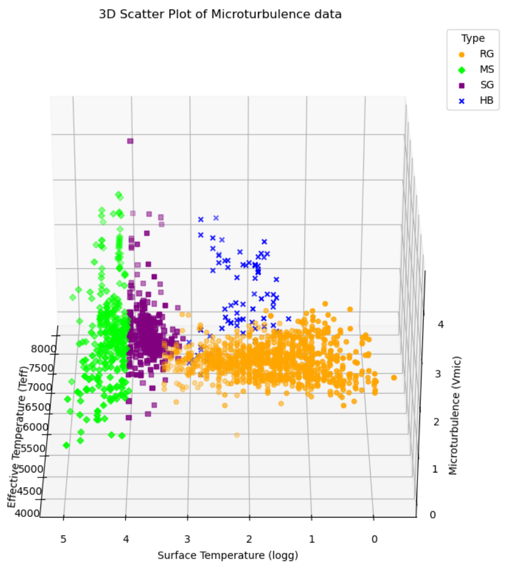
Correlation Matrices for numerical data and selected subset of variables. The top left panel shows the Pearson correlation matrix for all numerical variables in the dataset, capturing all the linear relationships across the dataset. The top right panel shows the Pearson correlation for the subset of variables, specifically Stellar Parameters & Chemical Abundances ($T_{eff}, logg, Fe/H, Vmic, Ba/Fe, C/Fe$). The bottom left panel displays the Spearman correlation for the same subset, while the bottom right panel shows the Kendall correlation. The visualisations suggest a strong non-linear correlation between microturbulence $[Vmic]$ & stellar parameters effective temperature $[T_{eff}]$ & surface gravity $[logg]$ can be observed.



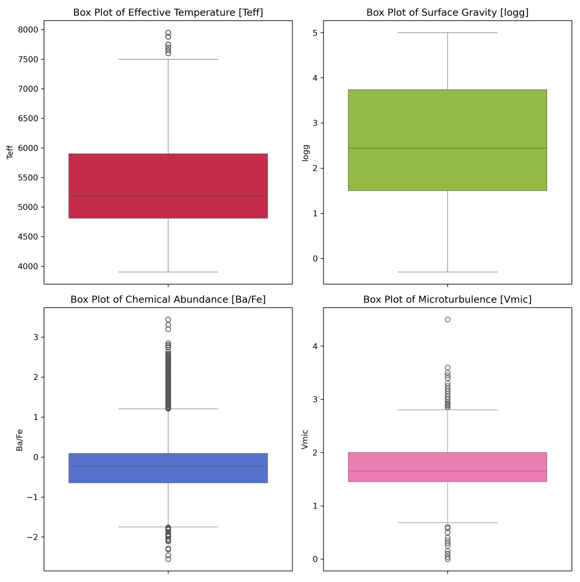
Distribution Analysis (Bar Plots) in the bar plots, we can observe that the stellar dataset is dominated by red giants located in the galactic halo. In addition, there are no measured neutron capture elements or signs of carbon enhancement in these stars. This indicates that the sample represents the older, more metal-poor stars found generally in the halo and therefore can not have neutron capture processes or carbon enrichment.



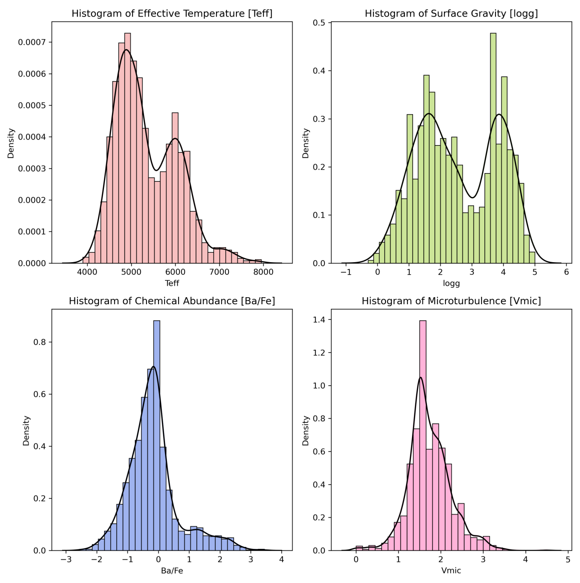
Pair Plots/Scatter Plots between microturbulence $[Vmic]$ and a couple of stellar parameters, categorised by $[Type]$. The left plot depicts microturbulence $[Vmic]$ vs effective temperature $[Teff]$, while the right plot shows microturbulence $[Vmic]$ vs surfcae gravity $[logg]$. The scatter plots reveal a weak clustering suggests a possible separation in the higher dimensions which will be clear in the next plot. This separation also suggests that microturbulence is a good parameter for distinguishing between different stellar types, particularly when combined with effective temperature and surface gravity.



Teff vs logg vs Vmic This 3D scatter plot visualises the relationship between $Vmic$, $Teff$, and $logg$ in the dataset. The x-axis represents logg, the y-axis represents Teff, and the z-axis represents Vmic. The plot reveals clear clustering in the 3D space unlike the pair plots, suggesting that microturbulence, effective temperature, and surface gravity can be used to effectively differentiate between these stellar types. This can also turn out to be useful for our regression task later.



Box Plots illustrating the distribution of effective temperature $[Teff]$ (top left), surface gravity $[logg]$ (top right), chemical abundance $[Ba/Fe]$ (bottom left), and microturbulence $[Vmic]$ (bottom right). Each box plot shows the median, quartiles, and outliers of the data. As we can observe chemical abundances and microturbulences has got a lot of outliers which will be removed as we proceed for our goals to achieve better prediction accuracy.

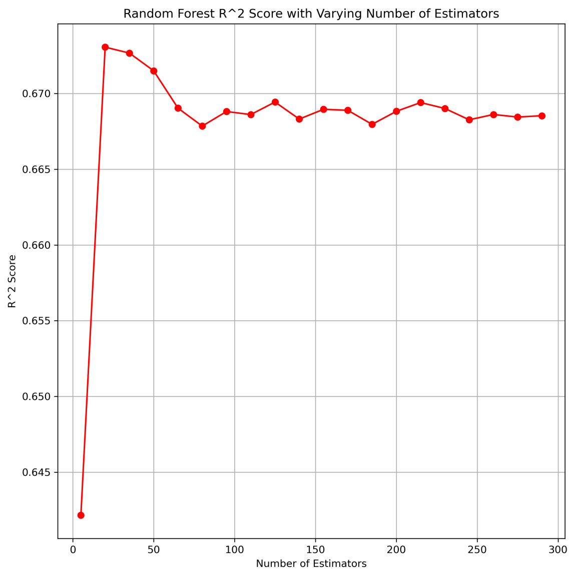


Distribution Analysis (Histograms) showing the distribution of effective temperature (Teff), surface gravity (logg), chemical abundance (Ba/Fe), and microturbulence (Vmic). $Teff$ & $logg$ seem to have a bimodal distribution which make them the candidates for clustering.

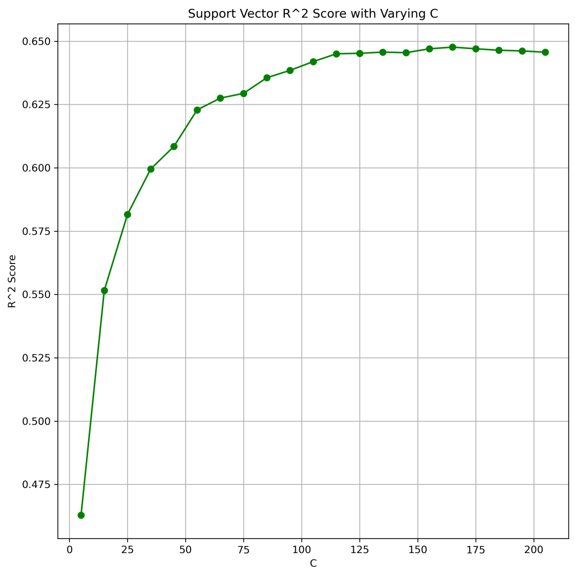
5 Experiments and Results

5.1 Imputation

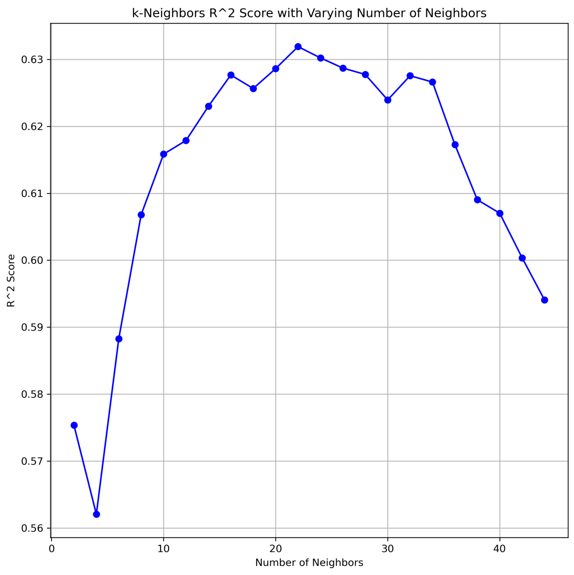
We modeled the microturbulence data $[Vmic]$ from the JINA Database in relation to reduced dimensions obtained from t-SNE and tested several parameters for all our models to achieve optimal performance.



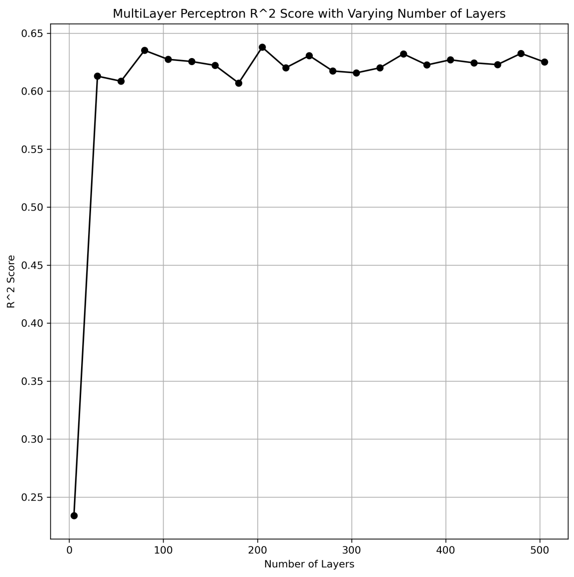
Random Forest Performance: The model achieves peak performance when the number of estimators is approximately 30. Also the model performs best with a maximum depth of 7.



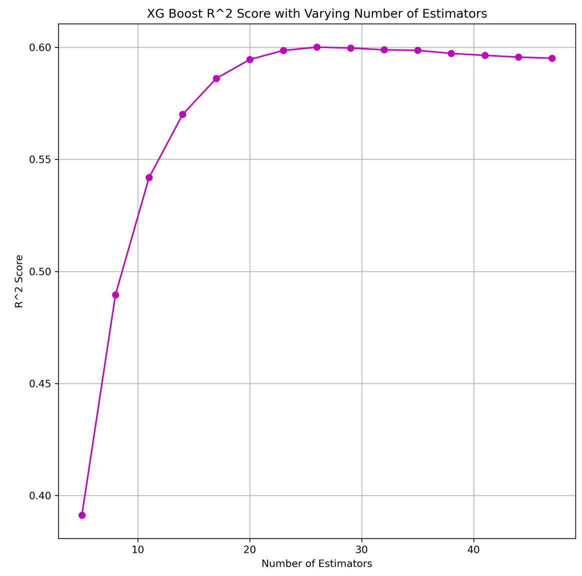
Support Vector Performance: The model performance stagnates after the regularisation parameter reaches 125. And the model performs best when $\epsilon = 0.1$, and with a custom $Matern() + WhiteKernel()$ kernel.



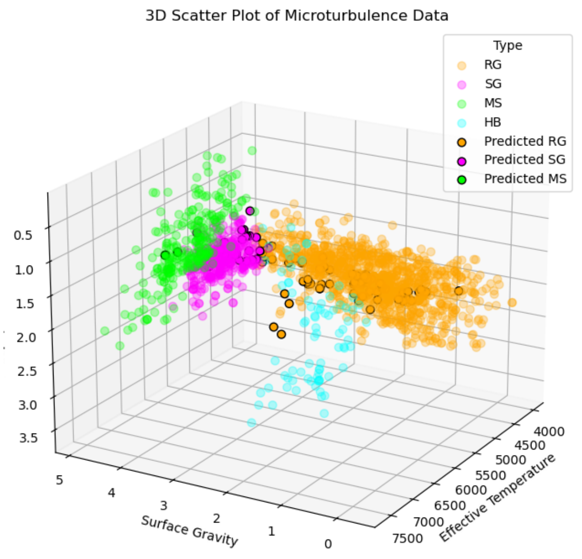
k-Nearest Neighbors Performance: The model performance stagnates when the number of neighbors is around 22 and quickly deteriorates. Also the model performs best with uniform weights.



Multi-layer Perceptron Performance: The model achieves optimal performance when the size of hidden layers is approximately 80. Also the model performs best when the learning rate is 0.1, with a $relu$ activation layer, and with a Limited-BFGS solver.



XGBoost Performance: The model achieves optimal performance when the number of estimators is approximately 25. Also the model performs best when the learning rate is 0.1.



Teff vs logg vs Predicted Vmic This 3D scatter plot visualises the relationship between $PredictedVmic$, $Teff$, and $logg$. The x-axis represents logg, the y-axis represents Teff, and the z-axis represents Vmic. The predictions made by the best model are highlighted in black, and as we can see, the predictions snugly fit with the existing data.

	MSE	R_squared
Random Forest	0.068	0.674
Support Vector	0.073	0.646
k-Nearest Neighbor	0.076	0.631
XGBoost	0.083	0.600
Multi-layer Perceptron	0.076	0.635

Errors & Scores Random Forest Regressor showed the best performance among all models. It showed the least Mean-Squared Error of 0.068 and captured the most variance with a R-squared score of 0.674. We believe the models would perform even better if the noise in the data is dealt with during feature engineering.

5.2 Classification

We set out to classify the stars based on Microturbulence $[Vmic]$, Surface Gravity $[logg]$, and Effective Temperature $[Teff]$. And we were able to model really well and achieved great results across all models.

	Precision	Recall	f1	#
HB	1.00	1.00	1.00	23
MS	1.00	1.00	1.00	55
RG	1.00	0.99	1.00	227
SG	0.97	1.00	0.98	63
Accuracy			0.99	368
Macro Avg	0.99	1.00	0.99	368
Weight Avg	0.99	0.99	0.99	368

Random Forest Classifier: Has near perfect classification, with precision, recall, and F1-scores close to or at 1.00 for most classes. Overall, the model has a high accuracy of 99%, with both the macro and weighted averages reflecting similarly strong performance across the test split of 368 instances.

	HB	**MS**	**RG**	**SG**
HB	23	0	0	0
MS	0	55	0	0
RG	0	0	225	2
SG	0	0	0	63

Confusion Matrix: There were 2 misclassifications of RG as SG; aside from that, the matrix suggests the model performed nearly perfect.

	Precision	Recall	f1	#
HB	0.95	0.91	0.93	23
MS	1.00	0.96	0.98	55
RG	0.99	0.99	0.99	227
SG	0.94	1.00	0.97	63
Accuracy			0.98	368
Macro Avg	0.97	0.97	0.97	368
Weight Avg	0.98	0.98	0.98	368

Support Vector Classifier: Is good at classification and could possibly be improved by selecting better parameters. Overall, the model has a high accuracy of 98%.

	HB	MS	RG	SG
HB	21	0	2	0
MS	0	53	0	2
RG	1	0	224	2
SG	0	0	0	63

Confusion Matrix: There are a few misclassifications across HB, MS & RG.

	Precision	Recall	f1	#
HB	1.00	0.83	0.90	23
MS	1.00	0.87	0.93	55
RG	0.99	0.99	0.99	227
SG	0.86	1.00	0.93	63
Accuracy			0.96	368
Macro Avg	0.96	0.92	0.94	368
Weight Avg	0.97	0.96	0.96	368

KNN Classifier: It has the poorest performance among all models, with low recall; however, like SVM, it can be improved by choosing good parameters, which we overlooked. The model has a accuracy of 96%.

	HB	MS	RG	SG
HB	19	0	3	1
MS	0	48	0	7
RG	0	0	225	2
SG	0	0	0	63

Confusion Matrix: There are many misclassifications across HB, MS & RG.

	Precision	Recall	f1	#
HB	1.00	1.00	1.00	23
MS	1.00	0.96	0.98	55
RG	1.00	0.99	1.00	227
SG	0.94	1.00	0.97	63
Accuracy			0.99	368
Macro Avg	0.99	0.99	0.99	368
Weight Avg	0.99	0.99	0.99	368

XGBoost Classifier: also has near-perfect classification and only falls short compared to Random Forest when classifying MS & SG. Overall, the model has a high accuracy of 99%.

	HB	MS	RG	SG
HB	23	0	0	0
MS	0	53	0	2
RG	0	0	225	2
SG	0	0	0	63

Confusion Matrix: There are a few misclassifications across MS & RG.

	Precision	Recall	f1	#
HB	0.96	0.96	0.96	23
MS	1.00	0.98	0.99	55
RG	1.00	0.99	0.99	227
SG	0.95	1.00	0.98	63
Accuracy			0.99	368
Macro Avg	0.98	0.98	0.98	368
Weight Avg	0.99	0.99	0.99	368

MLP Classifier: With similar performance to XGBoost, the MLP Classifier performs well in most classes, except for HB, and the dip in the F1 score might be due to the small data. The model has a accuracy of 99%.

	HB	MS	RG	SG
HB	22	0	1	0
MS	0	54	0	1
RG	1	0	224	2
SG	0	0	0	63

Confusion Matrix: There are a few misclassifications across HB, MS & RG.

6 Inference & Improvements

Random Forest methods emerged showed the best performance for both of our tasks, demonstrating versatility in regression/imputation & classification. This ensemble learning technique effectively combines multiple decision trees to enhance predictive performance, making it highly effective in handling complex datasets with varied features. The results show that Random Forest not only outperformed other models but also provided consistent results across different evaluation metrics, thus making it a reliable choice for our analysis. Overall, the findings suggest that Random Forest methods could prove to be useful in similar future tasks.

To improve the performance of the models, exploring different feature engineering techniques could help in capturing more variance in the data, especially for the regression task. For instance, creating interaction terms between key variables like effective temperature and surface gravity may uncover hidden patterns that are otherwise missed by the individual features. Moreover, incorporating domain-specific knowledge can also help us derive new features or aggregate existing ones, which could lead to more informative inputs for the model. Doing these might enhance the model's robustness, leading to an improved performance in regression analysis.

References

[1] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *Random Forests*, pages 587–604. Springer New York, New York, NY, 2009.

[2] Support vector machines — scikit-learn 1.2.2 documentation. https://scikit-learn.org/stable/modules/svm.html, 2023. Accessed: 17-04-2023.

[3] Cosma Shalizi. CMU Statistics 36-462/662: Methods of Statistical Learning. Lecture Notes 11: k-Nearest Neighbors, 2022. URL: https://www.stat.cmu.edu/~cshalizi/dm/22/lectures/11/lecture-11.pdf.

[4] Introduction to boosted trees — xgboost documentation. https://xgboost.readthedocs.io/en/latest/tutorials/model.html#introduction-to-boosted-trees, 2022. Accessed: 15-04-2023.

[5] Roger Grosse. University of Toronto CSC 411: Machine Learning and Data Mining. Lecture 5: Multilayer Perceptrons, 2019. URL: https://www.cs.toronto.edu/~mren/teach/csc411_19s/lec/lec10_notes1.pdf.

[6] Abdu Abohalima and Anna Frebel. Jinabase—a database for chemical abundances of metal-poor stars. *The Astrophysical Journal Supplement Series*, 238(2):36, oct 2018.

[7] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[8] C. Chatfield. *Problem Solving: A statistician's guide, Second edition*. Chapman & Hall/CRC Texts in Statistical Science. Taylor & Francis, 1995.