Data Augmentation for Stellar Classification Using Various Machine Learning Algorithms

Robert Cervone-Richards
Boston University
robertcr@bu.edu

William Geddes
Boston University
wjgedd14@bu.edu

Niky Popov
Boston University
nikyp@bu.edu

Abstract

This study explores various machine learning algorithms and data augmentation techniques for stellar classification, a critical task for understanding celestial phenomena and the composition of the universe. Employing algorithms such as k-nearest neighbors, gradient boosting machines, support vector machines, and random forests, we address the challenge of classifying stars into the Morgan-Keenan spectral classes: O, B, A, F, G, K, and M. Given the uneven distribution of star classes in typical datasets, we implement data augmentation strategies including the Synthetic Minority Over-sampling Technique (SMOTE) to balance class representation and enhance model performance. Our results demonstrate that these strategies significantly improve classification accuracy across different datasets, with Random Forest and SVM showing particularly strong performance enhancements.

1. Introduction

In this work, various machine learning algorithms and data augmentation techniques are proposed for the important task of stellar classification. Stellar classification involves categorizing a star based on measurements of its properties, such as temperature, mass, radius, luminosity, and absolute magnitude. The most commonly used stellar classification system is the Morgan-Keenan system, which places stars into seven categories: O, B, A, F, G, K, and M. Stellar classification is an important task because it allows scientists to learn more about the universe and make various predictions. For example, knowing how many stars of each category there are in a galaxy allows scientists to better understand the properties of the galaxy, and observing stars of various types allows scientists to learn how stars are formed and make predictions about how those stars will evolve. This being an important and difficult task, we propose the use of four machine learning algorithms for stellar classification: k-nearest neighbors, gradient boosting machine, support vector machine, and random forest.

2. Literature Review

In [12], Zhao, Wei, and Jiang classify stellar spectra from the Sloan Digital Sky Survey (SDSS) using an Ensemble Convolutional Neural Network (ECNN). Their implementation utilizes six distinct CNN architectures as base classifiers, applying a bagging strategy combined with weighted voting for integration. Ensemble CNNs integrate multiple CNN models to form a collective decision-making system. Each CNN in the ensemble is independently trained on the dataset, allowing them to learn different features of the data. The ensemble approach then aggregates predictions from all CNNs, typically through methods like voting or averaging, to produce a final classification. Unfortunately, due to a pivot in our project goals, we were unable to explore this method in our comparison of stellar classification. It remains as potential future step.

In [7], Kuntzer, Tewes, and Courbin classify stars based on spectral types using only the shape of their diffraction pattern in a single image. A catalog of stellar images is created, the pixel information is compressed so that only the most significant features are retained, and an ensemble of classifiers is used. Using an artificial neural network, the outputs from the various classifiers demonstrate a successful classification rate of 99

In [6], Hinners, Tat, and Thorp use representation learning and feature engineering to predict stellar properties from noisy and sparse data. Representation learning was done by using a recurrent neural network to extract the best features for classification according to one physical property at a time, thus removing human-based preconceived notions about what affects a star's classification. The accuracy of the RNN was evaluated using a confusion matrix, but unimpressive results led to the authors exploring feature engineering, which used properties of the data chosen by humans to analyze why certain features predict certain properties. 46 features were extracted and trained on a variety of models, with a Random Forest classifier yielding some of

the best results; this is one method which will be explored in this project.

The foundation for k-Nearest Neighbor, Random Forest Classification, and Support Vector Classifier will be taken from [9], which discusses several algorithms and techniques employed currently in stellar classification. The paper uses these algorithms, among others, to classify stars and galaxies using several features. These features include right ascension angle, declination angle, photometric system ultraviolet, red, and green filters, and more.

3. Problem & Proposed Solutions

3.1. Datasets Used

The original dataset being used for to both train and test the chosen algorithms for this study comes from Kaggle.com [5]. The dataset includes 40 examples of class O stars, 46 B stars, 19 A stars, 17 F stars, 1 G star, 6 K stars, and 111 M stars. Due to the under-representation of certain star classes, namely K and G stars, the performance of each algorithm would vary significantly each time they would be run. This is because of splitting of the data into either training or test point, where if there are no examples for K and G stars being used to help train the algorithm due to the small percentage of them present in the dataset to choose from, those classes of stars will be labeled incorrectly when labeling test points.

This dataset encompasses 240 data points, each with 5 features. These include **Temperature** measured in Kelvin, ranging from 1,939 K to 40,000 K, which reflects the diverse thermal states of stars. **Luminosity**, expressed as a multiple of the Sun's luminosity (L/Lo), varies dramatically from 0.00008 to 849,420, indicating a broad spectrum of star brightness. The **Radius**, measured in solar radii (R/Ro), extends from 0.0084 to 1,948.5, highlighting the significant differences in star sizes. **Absolute Magnitude** (Mv) spans from -11.92 to 20, illustrating the intrinsic brightness range of the stars. Additionally, **Star Color** is described using categorical attributes that denote the apparent color based on spectral characteristics.

In terms of correlations, luminosity and radius exhibit a moderate correlation (0.53), suggesting that larger stars tend to be brighter. Similarly, there are notable negative correlations between luminosity and absolute magnitude (-0.69), and radius and absolute magnitude (-0.61), indicating that brighter and larger stars typically have lower magnitudes (higher intrinsic brightness). (See Figure 1 for more details)

3.2. Problem Formulation

For stellar classification, the data classes are spectral classes (described above: O, B, A, F, G, K, and M). However, roughly 3 out of 4 stars in the universe belong to class "M" in the Morgan-Keenan system, and fewer than 1 in

	Temperature (K)	Luminosity(L/Lo)	Radius(R/Ro)	\
Temperature (K)	1.000000	0.393404	0.064216	
Luminosity(L/Lo)	0.393404	1.000000	0.526516	
Radius(R/Ro)	0.064216	0.526516	1.000000	
Absolute magnitude(Mv)	-0.420261	-0.692619	-0.608728	
Star type	0.411129	0.676845	0.660975	

	Absolute magnitude(Mv)	Star type
Temperature (K)	-0.420261	0.411129
Luminosity(L/Lo)	-0.692619	0.676845
Radius(R/Ro)	-0.608728	0.660975
Absolute magnitude(Mv)	1.000000	-0.955276
Star type	-0.955276	1.000000

Figure 1. Correlation Matrix of Original Dataset Features (ignore 'Star Type').

10,000 stars belong to class "O" [8]. This imbalance was represented in our original dataset and represented a challenge that we needed to overcome. Out of 240 total data points, 111 were "M" stars, 6 were "K" stars, and only 1 was a "G" star; this made techniques such as k-fold cross-validation impossible because if a class has fewer than k examples when trying to perform k-fold cross-validation, this can lead to inaccurate performance metrics. We looked for better datasets, but seemingly none were available, so rather than choose a new topic we decided to explore three different data augmentation techniques to see if they improved the performance of our algorithms. Note all our datasets used were standardized as follows:

$$z = \frac{x - \mu}{\sigma}$$

where x is original value of feature μ is the mean of each feature and σ is standard deviation of each feature.

3.3. Solution Approaches

The first and perhaps simplest data augmentation technique used involved the over-representation of data points from under-represented classes and the under-representation of data points from over-represented classes. Four extra "G" data points were harvested from the Internet by searching Google for the names of stars that fall into the "G" category, and their features (temperature, radius, luminosity, absolute magnitude, and color) were also found online. Then each "K" and "G" data point was copied and pasted once to have at least 10 examples of each class (the number 10 was chosen arbitrarily). This dataset was given a new name: "MK csv #2," whereas the original dataset was titled "MK csv #1". The results that were obtained from these and the other datasets are described in section 5.

The next dataset was titled "MK csv #3" and kept the changes that were implemented in "MK csv #2," while also under-representing the most over-represented class. This was done by deleting 50 random "M" data points so that there 61 "M" stars, 46 "B" stars, 40 "O" stars, 17 "F" stars, 19 "A" stars, 12 "K" stars, and 10 "G" stars. The number 50 was chosen randomly, but the decision to create and utilize this dataset and the one described previously despite the

imbalance in the number of examples per class was intentional, because we wanted to be able to compare the results obtained from the original dataset with the results obtained from datasets which were only slightly different than the original. Again, the results are described in section 5.

The second augmentation technique used was Synthetic Minority Over-sampling Technique (SMOTE). This technique is a method used to address class imbalance by generating synthetic data points.SMOTE works by creating synthetic examples rather than by oversampling with replacement. It does this by selecting samples from the minority class and their nearest neighbors in the feature space. For each selected sample, synthetic points are generated along the line segments connecting it to its nearest neighbors. Mathematically, a synthetic point is created as follows: for a sample x_i and its nearest neighbor x_{nn}, a new sample x_{new} is generated by interpolating between the two, specifically

$$x_{new} = x_i + (x_{nn} - x_i) \times \delta$$

where δ is a random number between 0 and 1. The main strength of SMOTE is that it helps prevent overfitting that typically occurs with simple random oversampling by enriching the dataset with new, plausible examples rather than copies, thereby extending the feature space and enabling better model generalization. We wanted to run our models on the SMOTE dataset to learn more about how effective this method is in generating synthetic data to improve model learning. Results again are in section 5.

Furthermore, to generate a usable dataset that will produce more consistent results, specifically when using cross-validation, artificial data points needed to be generated based on what exists in the Kaggle dataset. This was our third augmentation technique. To do this, the max and min values for each feature in regards to a specific label was taken from the data, where an evenly distributed number of points was then generated within that range, while being given that label. This process was repeated for each star type, where 100 extra data points were created for each class, except for G and K class stars. In contrast, three hundred examples were added for K class stars in an attempt to make each class more evenly represented. This was not possible for G class stars, as there was one example, meaning that no meaningful ranges of feature data could be extrapolated to create more data points with this label. We attempted to make a range that matched the ranges of feature values seen in the other classes, but they all were greatly inconsistent, making this method ineffectual as well. This meant, that despite making every other class well-represented, G class stars were still heavily underrepresented.

To fix the issues seen when generating artificial data points based on minimum and maximum data ranges for each feature for a specific class, the minimum and maximum ranges were sourced from online sources, such as Wikipedia and several astronomy information archives [1–4,10,11]. Using a similar process as seen before, using these sourced ranges of values, an evenly distributed 400 points was created for each class. Although the points generated using this method ensure nothing is unevenly represented, issues still rise from how there is a very distinct separation between labels, where this is no overlap for any feature. This does not reflect too well what is seen in real life, where star classes are not so distinct and some stars that have features relating to that of a certain class have other traits that make it more in line with another.

4. Implementation of Algorithms

4.1. k-Nearest Neighbors

The k-nearest neighbors algorithm (kNN) classifies each data point by observing its k-nearest neighbors in the feature space, where k is an integer parameter in need of tuning, and assigns the majority class label of those k nearest neighbors as the predicted value for the data point. This algorithm was implemented using an instance of the kNeighborsClassifier class from sci-kitlearn.org. After the data was split into training and test sets, and scaled as described above, grid search was performed on possible combinations of the following parameters: the number of neighbors (k) used to classify each data point, the weighting applied to the nearest neighbors, and the distance metric being used to find the nearest neighbors. Due to the relative simplicity and quick runtime of the k-nearest neighbors algorithm, an exhaustive grid search was performed to find the optimal combination of parameters. Every possible odd value (to avoid the need for tie-breakers) for k between 1 and 21 was tested, along with uniform and distance weighting methods and Manhattan and Euclidean distance metrics. When the uniform weighting method is used, each of the k-nearest neighbors are considered equally when classifying a data point, but when the distance weighting method is used, the data points among the k-nearest neighbors which are closer to the data point being considered are given greater weight than those that are farther away. The combinations of parameters which yield the best results for each dataset are shown in Table 1.

Interestingly, the Euclidean distance metric (which involves computing the L-2 norm between data points) only works best for one dataset. This could suggest that Manhattan norm is less sensitive to outliers, since calculating the Euclidean norm involves squaring the distance between points, which amplifies the affect of large differences. Furthermore, the ”Artificial Data Method 2” dataset was compiled without outliers, as described above, whereas the other datasets would have more outliers, such as the relatively

Table 1. kNN Optimal Parameters

Dataset	Best "k"	Best Weighting	Best Distance
MK csv #1	11	Distance	Manhattan
MK csv #2	15	Distance	Manhattan
MK csv #3	19	Distance	Manhattan
Artificial #1	1	Uniform	Manhattan
Artificial #2	1	Uniform	Euclidean
SMOTE	5	Distance	Manhattan

much smaller of "G" examples. Also interesting is that Artificial Data Methods 1 and 2 get the best results when k = 1, but this makes sense since these data points were generated uniformly within pre-defined ranges, so the 1 nearest neighbor would be likely to fall in the same class because it would be likely to have data in the same range for many if not most features. Finally, it also makes sense that the distance scaling method would improve the results except with Artificial Data Methods 1 and 2, since distance scaling can reduce the impact of neighboring points which are farther away and therefore more likely to be from another class. By contrast, the data in Artificial Data Method 1 was generated with little overlap between the ranges of features, and Artificial Data Method 2 was generated with no overlap.

4.2. Gradient Boosting Machine

The gradient boosting machine (GBM) algorithm is an ensemble learning method, meaning it combines the predictions from multiple models to improve overall performance. Each model in the ensemble is known as a weak learner, meaning its only requirement is that it is at least as good as the best constant predictor. For example, any model that correctly predicts the flip of a coin 50% of the time could be considered a weak learner (if each flip of the coin wasn't an independent event). In order to improve performance beyond 50%, multiple weak learners are combined linearly, with each new learner trained to correct the errors made by the previous models. This approach gradually improves the overall model's accuracy.

This algorithm was implemented using an instance of the GradientBoostingClassifier class, also from scikitlearn.org. A good combination of parameters were found using 20 iterations of a random search of the following: Number of boosting stages, learning rate, maximum depth, minimum samples to split a node, and minimum samples at a leaf node. Unlike a grid search, a random search is not exhaustive, but it can find a nearly-optimal combination of parameters in a fraction of the time it would take a grid search to find the optimal set of parameters. With GBM being a more complex algorithm than kNN with a slower runtime, a random search was used in order to save time.

4.3. Support Vector Machine

We also implemented a Support Vector Machine (SVM) model to classify stars based on their spectral characteristics. SVMs are typically used for binary classification, but we used the All-Pairs strategy for multi-class classifcation. This approach involves training a classifier for each pair of classes and deciding the final class based on the majority vote from these classifiers.

We chose a linear kernel for our SVM after testing different options because it effectively managed our dataset's dimensionality without overfitting. We opted for a soft margin SVM, allowing some misclassifications to improve the model's ability to handle real-world data. Using GridSearchCV, we found the best setting for the C parameter was 100, balancing classification accuracy and margin width.

This setup allowed our SVM to perform well on the test data, for which the results can be seen in the next section.

4.4. Random Forest

5. Experimental Results

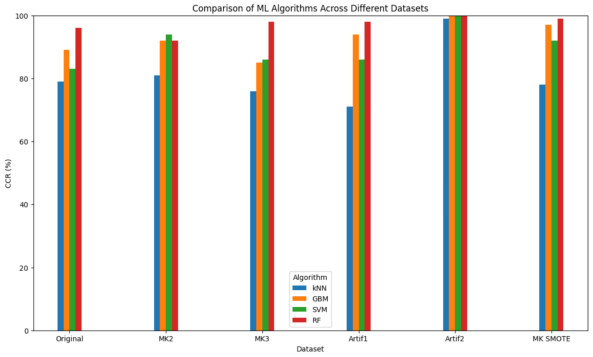

Figure 2. CCR performance metrics for each algorithm in regards to each studied dataset.

Shown above, Figure 2 summarizes the test data correct classification rate (CCR) results for all four algorithms applied to all six datasets. CCR is calculated by counting the number of data points whose labels were correctly predicted by the model and dividing by the total number of points; this is also known as accuracy.

$$\text{CCR} = \frac{\text{\# of Correct Predictions}}{\text{Total \# of Predictions}} \quad (1)$$

Overall, random forest was the best-performing algorithm in terms of accuracy, with a minimum CCR of 88%, and kNN was the lowest-performing algorithm, with a maximum CCR of 81% (besides the Artificial Dataset 2, which had extremely high CCR values across all algorithms, for reasons discussed below). The application of SMOTE on

the MK SMOTE dataset appears to enhance the classification accuracy for all algorithms except kNN when compared to the original dataset. This improvement underscores the effectiveness of SMOTE in balancing class distribution, which in turn enhances model performance. Both GBM and SVM exhibit increased CCRs in datasets that have undergone extensive processing. This performance boost suggests that these algorithms are particularly sensitive to feature distributions and can effectively leverage complex transformations to enhance classification accuracy.

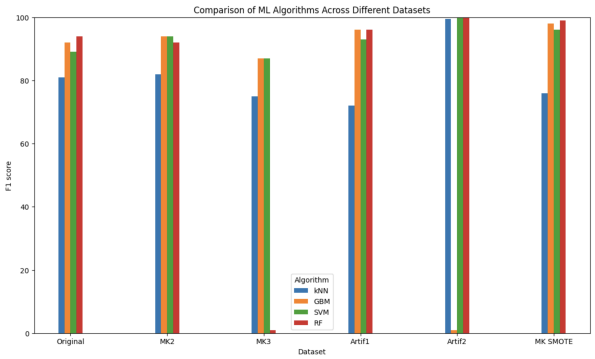

Figure 3. F1-Score performance metrics for each algorithm in regards to each studied dataset.

Shown above, Figure 3 summarizes the test data F-1 scores for all four algorithms applied to all six datasets. F-1 score is a metric which is computed as the harmonic mean of precision and recall, where precision is the proportion of positive predictions that are actually correct and recall is the proportion of actual positives that are correctly predicted. The equations for these quantities are shown below.

$$\text{Precision} = \frac{\text{\# True Positives}}{\text{\# True Positives} + \text{\# False Positives}} \tag{2}$$

$$\text{Recall} = \frac{\text{\# True Positives}}{\text{\# True Positives} + \text{\# False Negatives}} \tag{3}$$

$$\text{F-1 Score} = 2 * \frac{\text{Precision * Recall}}{\text{Precision + Recall}} \tag{4}$$

F-1 score can be a better performance metric than CCR when the dataset has an uneven distribution of classes, as was the case with more than one of our datasets (particularly the original). For example, if a disproportionate number of data points fall into one class (as was the case with the ”M” class in our original dataset) then the model could correctly predict all of those data points but none of the others and have a misleadingly-high CCR. Further, if one class is under-represented in the dataset (as was the case with the ”G” and ”K” classes in our original dataset) then the low-frequency class(es) could be under-represented in the CCR but better accounted for with the F-1 score.

6. Conclusion

This study demonstrated the efficacy of various machine learning algorithms and data augmentation techniques in addressing the challenges of stellar classification. By employing techniques such as over/undersampling, SMOTE and experimenting with diverse algorithms including k-nearest neighbors, gradient boosting machines, support vector machines, and random forests, we systematically compared the classification accuracy across differently processed datasets. Notably, the use of Random Forest and SVM yielded robust performance improvements, underscoring the importance of appropriate algorithm selection and data preprocessing in achieving reliable classification outcomes.

7. Description of Individual Effort

Billy: Data Augmentation Technique #1, kNN, GBM

Robert: Data Augmentation Technique #2, Random Forest

Niky: Data Augmentation Technique #3 (SMOTE), SVM

*each wrote same respective sections

References

[1] B-type star. http://astro.vaporia.com/start/bclass.html. 3

[2] Class a star. https://beyond-universe.fandom.com/wiki/ClassAstar. 3

[3] Class m star. https://beyond-universe.fandom.com/wiki/ClassMstar. 3

[4] Spectral type characteristics. http://astro.phy.vanderbilt.edu.html. 3

[5] Deepraj Baidya. Star dataset to predict star types, 2019. Data retrieved from Kaggle, https://www.kaggle.com/datasets/deepu1109/star-dataset/data. 2

[6] Trisha A Hinners, Kevin Tat, and Rachel Thorp. Machine learning techniques for stellar light curve classification. *The Astronomical Journal*, 156(1):7, 2018. 1

[7] T Kuntzer, M Tewes, and F Courbin. Stellar classification from single-band imaging using machine learning. *Astronomy & Astrophysics*, 591:A54, 2016. 1

[8] Glenn Ledrew. The Real Starry Sky. , 95:32, Feb. 2001. 2

[9] Tanvi Mehta, Nishi Bhuta, and Swati Shinde. Experimental analysis of stellar classification by using different machine learning algorithms. In *2022 International Conference on Industry 4.0 Technology (I4Tech)*, pages 1–8. IEEE, 2022. 2

[10] Wikipedia contributors. O-type main-sequence star — Wikipedia, the free encyclopedia, 2024. https://en.wikipedia.org/w/index. 3

[11] Wikipedia contributors. Stellar classification — Wikipedia, the free encyclopedia, 2024. https://en.wikipedia.org/w/index. 3

[12] Zhuang Zhao, Jiyu Wei, and Bin Jiang. Automated stellar spectra classification with ensemble convolutional neural network. *Advances in Astronomy*, 2022:1–7, 2022. 1