Spaceship Titanic Classification Model Analysis

Jett Tipsword
Rafael Hernandez

1. ABSTRACT

In this problem, we are trying to figure out the cause behind the disappearance of passengers on a spaceship through a spacetime anomaly. To do this, we employed various machine learning techniques. The first step was to fill missing values in the dataset and then create new features while dropping some unnecessary ones. Categorical columns were encoded by creating dummy variables for each unique category and normalizing the dataset using the Normalizer() function. Finally, we trained four different machine learning classifiers: KNN, Decision Tree, Gradient Boosting, and Random Forest. The results showed that the Gradient Boosting classifier had the highest classification accuracy of .8076, indicating that it was the best model for this problem. This project demonstrated the effectiveness of using machine learning in solving complex problems by applying data preprocessing techniques and trying different models to identify the most suitable one.

2. INTRODUCTION

The story of the Spaceship Titanic is a compelling and intricate one that has captivated scientists and researchers alike. Launched just a month ago, this interstellar passenger liner was tasked with transporting thousands of passengers to three newly habitable exoplanets. However, during its voyage, the ship collided with a mysterious space time anomaly hidden within a dust cloud while rounding Alpha Centauri. As a result, almost half of the passengers were mysteriously transported to an alternate dimension, leaving us with more questions than answers.

The scope of this cosmic conundrum is staggering, and it requires us to not only understand the fate of the passengers but also to develop preventative measures for future occurrences. To crack this mystery, we must utilize cutting-edge data science techniques and technologies. From analyzing the data transmitted by the Spaceship Titanic to creating predictive models, we must harness all the resources available to us to uncover the truth.

While this task may seem daunting, it is also exhilarating. The resolution of this problem could have significant implications for space exploration and our understanding of the universe. The journey ahead is bound to be a challenging one, but it is one that we must undertake if we hope to unravel the mysteries of the cosmos.

A tool that could prove useful in solving this problem is machine learning models. These models are very well suited to analyzing large amounts of complex data and show hidden patterns that may be unclear to the human eye. Using these techniques, we can process and analyze the data transmitted from the spaceship, potentially revealing any clues that could provide insight into the fate of the missing passengers. Also, the predictive modeling capabilities of machine learning can be leveraged to develop strategies for avoiding similar incidents in the future. The flexibility of accuracy of well-tuned machine learning models makes it a useful asset for solving complex problems within this domain.

3. METHODS

3.1. Necessary Imports

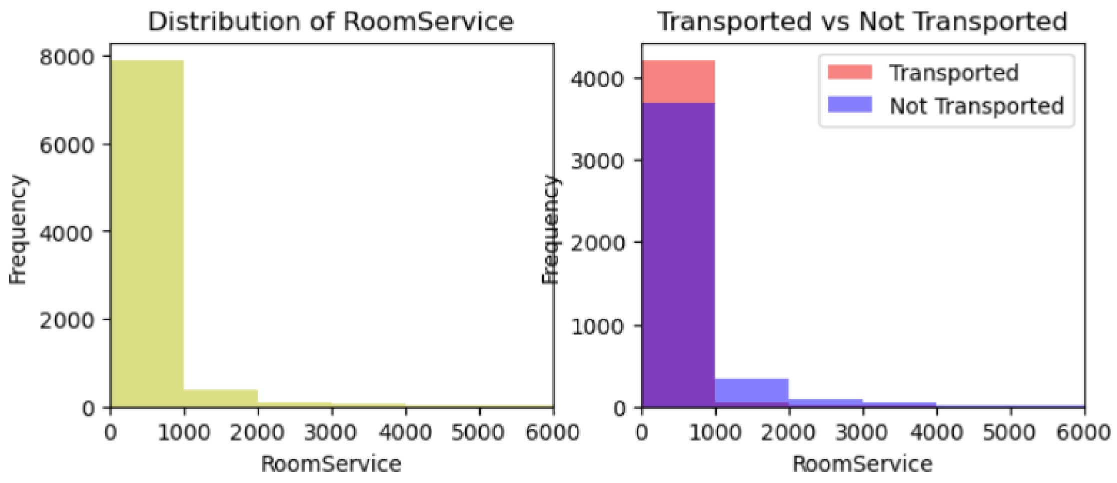
To start, our program begins by importing necessary libraries for NumPy, Pandas, Matplotlib, Seaborn, and OS. Then, the provided train and test comma separated value files are read using the read csv function. The files contain data related to passengers on the Titanic spaceship, with 'train_set' used for training a machine learning model, and 'test_set' for evaluating the model's performance. The data is stored in a tabular format with rows representing individual passengers and columns representing various attributes such as age, sex, and class, and 'header=0' allows the first row of each file to contain the column header for each feature class.

3.2. Exploratory Data Analysis

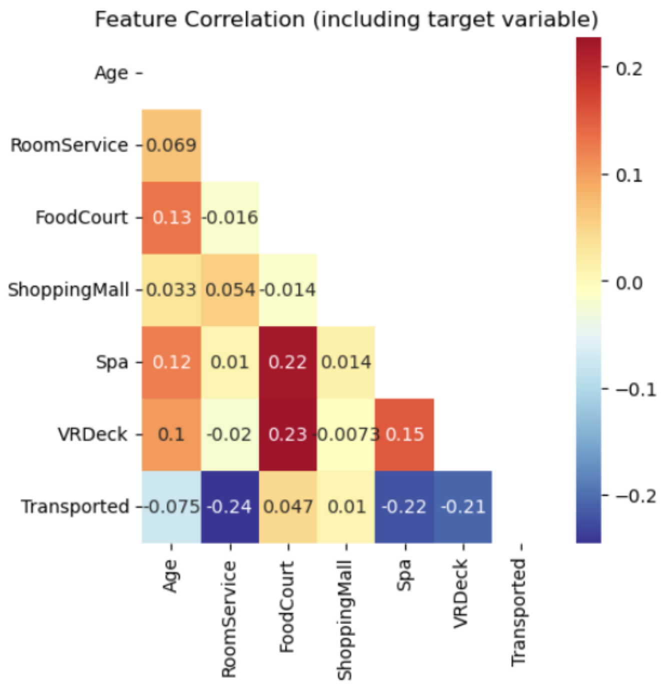
We perform exploratory data analysis by printing the first few rows of data from the training and test sets to get an idea of the data and features they contain. The head() function is used to print the top few rows of a DataFrame, which represents a table of data. The shape of both datasets is printed to provide information about the number of rows and columns, which is important for identifying potential issues in the data such as inconsistent data types or missing values. Understanding the size of the dataset can also help inform decisions around modeling strategy or feature selection. Knowing the shape of the dataset is a crucial first step in data analysis and preprocessing as it establishes a foundation for further analysis and modeling.

Then, the next step is to print the data types of various features in the training set to ensure the data is in the proper format and to identify features that require encoding or transformation. The number of unique values for each feature is printed to inform decisions around feature engineering, preprocessing, and identifying categorical features that require transformation for machine learning models. Understanding the distribution of unique values in the dataset can also provide insights into the data's structure and potential problems such as class imbalance. This step is crucial for data analysis and preprocessing, allowing for better data understanding and processing.

Now that we have observed various amounts of information about our dataset we can continue with our data exploratory analysis, in which we perform initial investigations of the data. The goal here is to see if we can spot any patterns, anomalies, and correlations in the data we have been given. Methods such as data visualizations, data cleaning and preprocessing, and feature engineering will be some of the techniques we used to explore our data. To begin, we generate a histogram for each numeric feature in the training set, showing the distribution of values and how they are related to the Transported target variable. The first histogram shows the distribution of the feature across the entire dataset, while the second histogram splits the data into two groups based on whether the individual was transported. These two histograms work together to show us the distribution of values between our numerical features and whether they were transported or not. This visualization can be useful for identifying any patterns or correlations between the features and the target variable, which can inform feature engineering and selection for machine learning models. It can also help to identify any outliers or unusual distributions that may require further investigation or preprocessing. An example of the Room service histograms is shown below.

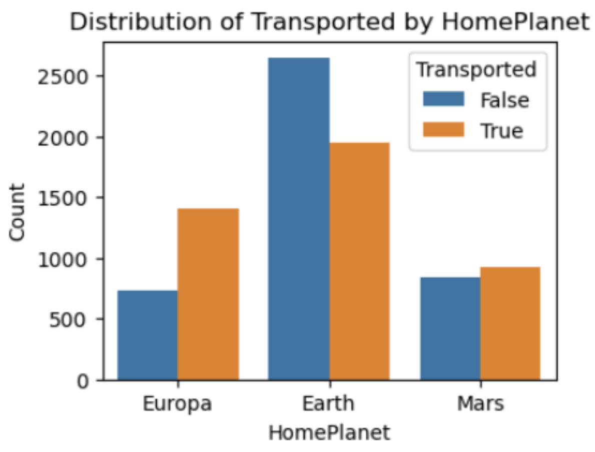


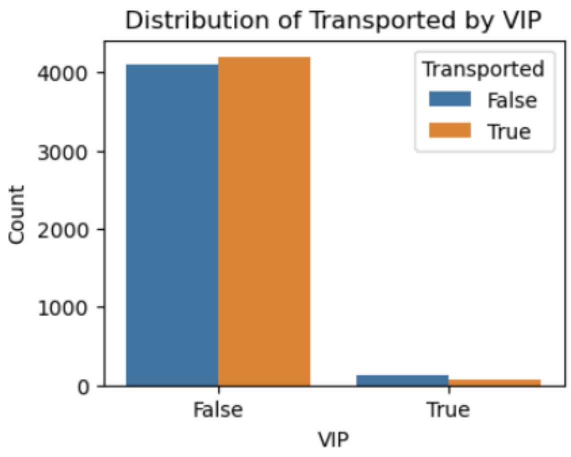
Next, we use another data visualization technique to further explain our data. This time we are using a heat map plot using seaborn. This heat map plot of the correlation matrix (numerical features) provides information about the linear relationships between pairs of features in our training set, as shown below.



	Age	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	Transported
Age							
RoomService	0.069						
FoodCourt	0.13	-0.016					
ShoppingMall	0.033	0.054	-0.014				
Spa	0.12	0.01	0.22	0.014			
VRDeck	0.1	-0.02	0.23	0.0073	0.15		
Transported	-0.075	-0.24	0.047	0.01	-0.22	-0.21	

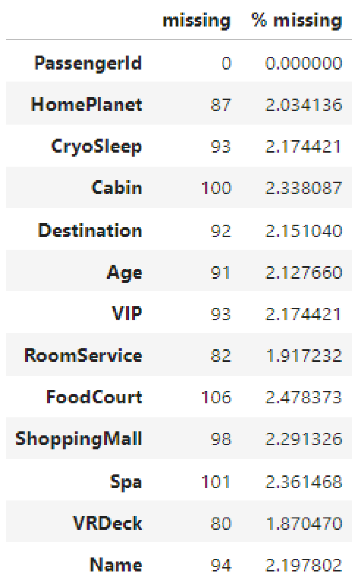
One of the reasons different types of plots are used to analyze data is due to the information that they gave. Histograms are great when wanting to see the distribution of data. Here, we use a heat plot to view correlations between features, and the target value. The color scale you see represents the magnitude of the correlation coefficient. Dark shades in red show positive correlations between features, and white shades show negative correlations between features. We can learn a lot from looking at our data in the heat plot. For example, we can identify which pairs of features have a high correlation to each other, meaning having higher correlation coefficients. As you can see, we have some darker shades of red between the Food Court and VR Deck, and Food Court and Spa. This may be because passengers who use and spend money at the Food court, also do so at the Spa and VRDeck. Our goal is to see if we have any unusually high correlation between features that may introduce problems in our performance as well as interpretation of the models which we don't have. Additionally, we can also view whether any of the features have a high correlation with our target feature "Transported". As you can see, our target feature does not have much correlation with our other features, as shown by the pale blue and white shades of color. We look for correlation between our target features and non-target features because we want to identify high correlations for later use as highly correlated features are more informative as they will likely affect the models more than other features. Next, we use "train_set.desribe()" to further describe and view some data in our training set. Using .describe() will help us spot any outliers, and ranges of the features in our train_set. Doing so will allow us to decide whether we should scale or normalize some of the data. In our case, we will normalize some of the data in a later cell. In the next cell we use count plots to provide us information on the distribution and relationships between our categorical features like Home Planet, CryoSleep, Destination, and VIP, which are all True False values. These count plots can help us identify inconsistencies in our data and view the distribution of transported by each feature. As you can see from our plots, we added the colorization scheme so that we can also see whether the passengers in each respective categorical group were transported or not. One of the count plots that really stood out was that most passengers on the ship were not VIP.





3.3. Data Pre-Processing

Now that we have performed some visual analysis of the data, with the help of the graphs from above, we will now perform some data preprocessing to “filter” our data that we have read into dataframes. Our first step is identifying all our missing values in the data set. To do this, we display a table of all missing tables. In doing so, we can see how we can proceed with missing values.



	missing	% missing
PassengerId	0	0.000000
HomePlanet	87	2.034136
CryoSleep	93	2.174421
Cabin	100	2.338087
Destination	92	2.151040
Age	91	2.127660
VIP	93	2.174421
RoomService	82	1.917232
FoodCourt	106	2.478373
ShoppingMall	98	2.291326
Spa	101	2.361468
VRDeck	80	1.870470
Name	94	2.197802

Since in total there are a good number of missing values, we don't want to delete whole rows because that would result in too large of a loss of information. This brings us to the next two cells in which we take care of data pre-processing using two functions. Clean data takes a data frame as a parameter and fills empty cells in our data for most features. To fill some of the features, we went ahead and calculated the median of the preexisting features and set the missing values equal to that. For the features remaining, we found which feature was occurring the most and simply set the missing features in these to the one that appeared the most. For example, most VIP and Cryosleep values were false, so we just set them to false. After these missing values were taken care of, we decided to create new features called cabin_1 and cabin_2, this was because we wanted to split the values in the Cabin feature to only use the first and last values only and not the number that was originally in the middle. We did this because we felt like using all 3 values separated by dashes wouldn't be very meaningful, so we separated them into these new two features where only the cabin letter and side was saved. We also created a new feature called “Spent” because we felt as if the total money spent from a passenger would be more informative than the amount spent at each place. After the data was cleaned, we noticed that we had some categorical data such as HomePlanet, Destination, Cabin_1, Cabin_2, that could be one hot encoded to be represented as binary values. The reason we did this is because we knew by transforming this data our models would be able to make predictions and analyze patterns more accurately. To do so we use the get_dummies method. To further some of our pre-processing even more we decided to normalize our data. We did this by using sklearn Normalizer because some algorithms such as KNN rely on magnitudes of input features to compute distances. Normalizing the data scales each row of the input features to have a unit norm, while still preserving the direction of the original data. We have finished the pre-processing steps of this assignment, next, we can move into the training step.

3.4. Model Tuning

Now that our data is preprocessed and we have performed data analysis, we are ready to train our models. We start by making our train test split with a train test split of .2. We wanted to use a multitude of models to explore improvements in accuracy scores, have different perspectives, and eventually compare the accuracy scores that the models gave us. While training, we used a total of 4 models, and followed a similar structure in training each one. Each model trained was trained by using grid_search to hyper tune the models and eventually use the best parameters. Then, the models were trained on the best hyperparameters found previously in grid_search. X_train and y_train features were then fit. We wanted to save the score that the grid_search had performed into a variable so that at the end we could graph the accuracies of each result. Using grid_search did come with a downside. We were faced with slightly here run times, but this is understandable since grid search will try multiple combinations of hyperparameters until it finds the best ones to train the model. To continue, we trained the models on the best params that grid search was able to find, and stored the predictions of the model on the X_test features in y_pred. The confusion matrix was printed for further analysis of the results of each model.

4. RESULTS

4.1. Tabulation of Accuracy Scores and Optimal Hyper Parameters

Below are the accuracy scores of the 4 classifiers, rounded to the nearest 4 decimal places, and the confusion matrix for each classifier is located below the table. As you can see from Table1, each model had various optimal hyper parameter values based on limited hyper parameter tuning method.

	KNN	**Decision Tree**	**Gradient Boosting**	**Random Forest**
Accuracy Score	.7893	.7916	.0.8076	.8017
Best Hyper Parameters	Num neighbors : 11 Weights : uniform	Max depth : 7 Max features : none Min sample leaf : 8 Min sample split : 4	Learning rate : 0.1 Max depth : 5 Max features : log2 Min sample leaf : 2	Max depth : none Max features : sqrt Min sample leaf : 2 Min sample split : 6 Num estimators : 100

Looking at the best model by classification accuracy, it turns out to be Gradient boosting. We take a closer look at what each Best Hyper Parameter means and why it turned out to be the best amongst the others.

Learning rate parameter controls the step size at each iteration when updating the weights of the decision trees. A lower learning rate can help prevent overfitting but can also increase the number of iterations required to reach convergence. In this case, a learning rate of 0.1 was found to be optimal, which suggests that a moderate step size was appropriate for this dataset.

Max depth parameter controls the maximum depth of the decision trees used in the gradient boosting model. When the tree is deeper, the model can capture more complex relationships in the dataset but can also lead to overfitting. In this case, a maximum depth of 5 was found to be optimal, which implies that a moderate depth was appropriate for this dataset.

Max features parameter controls the maximum number of features to be considered when splitting each node in the decision tree. Using a subset of the available features could help prevent overfitting and improve the model's performance. In this case, using log2 of the total number of features was found to be optimal, which suggests that a subset of features that is roughly half the total number of features was appropriate for this dataset.

Finally, the min sample leaf parameter controls the minimum number of samples required to be considered a leaf node in the decision tree. A higher value for this parameter can help prevent overfitting but can also lead to underfitting if the dataset is too small. In this case, a value of 2 was found to be optimal, which suggests that a relatively small number of samples per leaf was appropriate for this dataset.

In summary, the best combination of these parameters was found to result in the highest accuracy score for the Gradient Boosting model on this specific dataset. However, it's important to note that the optimal parameter values may vary for different datasets and classification tasks, and so parameter tuning is often an iterative process of experimentation and evaluation.

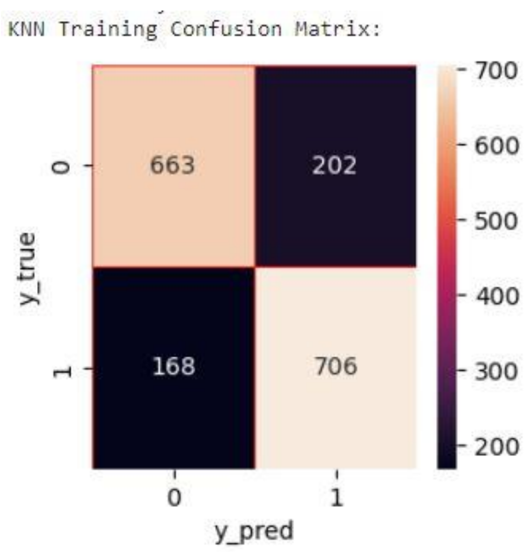
4.2. Confusion Matrices

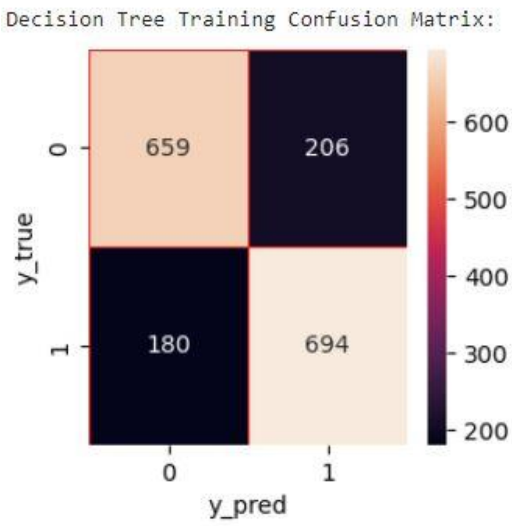
Below is a table that demonstrates how to interpret the 2 x 2 squares that represent our confusion matrices.

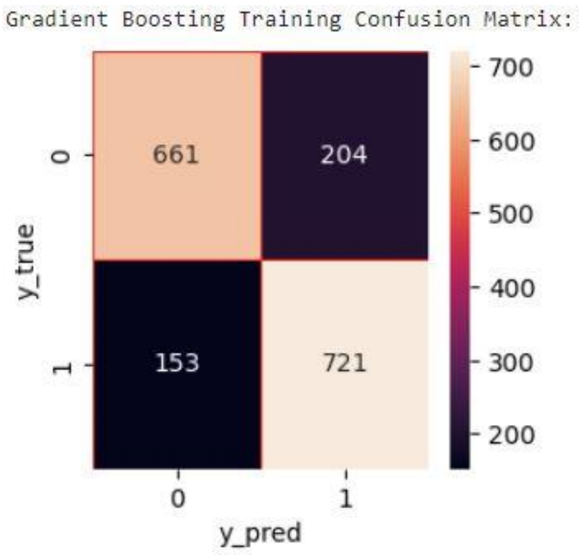
	Negative (TN)	**Predicted Negative (TN)**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

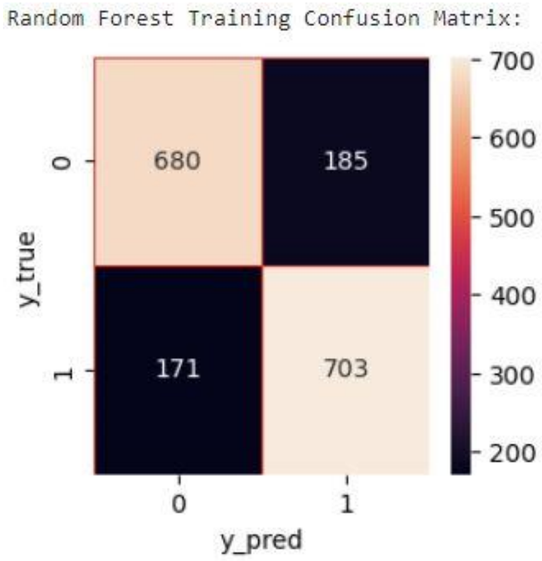
The rows represent the actual class labels, while the columns represent the predicted class labels. The true positive (TP) represents the number of instances that were correctly classified as positive by the model, while the false positive (FP) represents the number of instances that were incorrectly classified as positive. The true negative (TN) represents the number of instances that were correctly classified as negative, while the false negative (FN) represents the number of instances that were incorrectly classified as negative.

These confusion matrices provide useful information for evaluating the performance of our classification model, such as the accuracy for each classifier. For example, the accuracy of the model can be calculated as the sum of the true positives and true negatives divided by the total number of instances. As we can see when we apply this method to all the models, we get their confusion matrix as shown below.

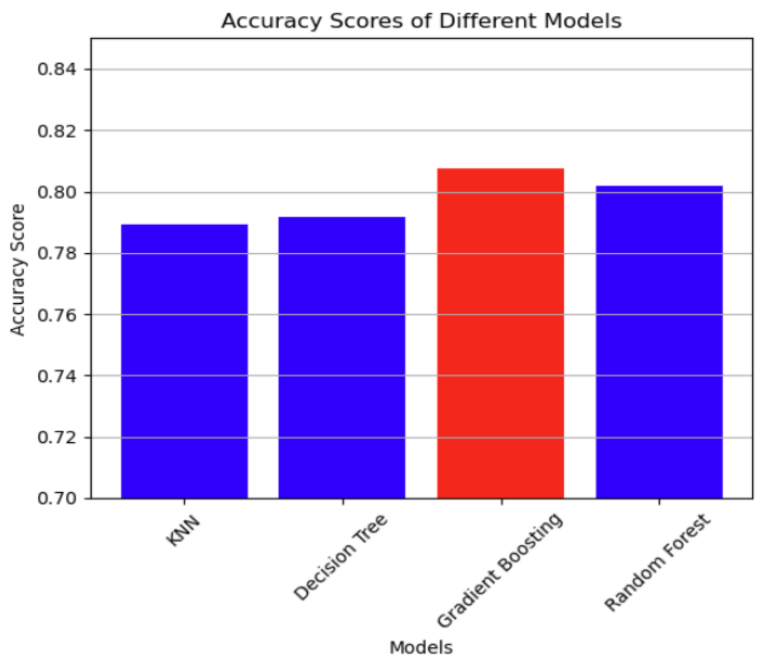








4.3. Results Analysis



In the figure above, we have each model's classification accuracy plotted with respect to their accuracy score. The results show that Gradient Boosting has the highest accuracy score of 0.8076, followed closely by Random Forest with an accuracy score of 0.8017, and then Decision Tree with an accuracy score of 0.7916. KNN has the lowest accuracy score of 0.7893.

A possible explanation of these results is that the dataset for the spaceship titanic may have nonlinear relationships between the features, which can make decision trees and ensemble methods (such as Gradient Boosting and Random Forest) better suited for the classification task at hand. Decision trees can capture nonlinear relationships between features by recursively splitting the data into smaller and smaller subgroups based on the features, while ensemble methods combine multiple decision trees to improve their accuracy and reduce overfitting.

Also, something to consider, KNN is a distance-based algorithm that relies on the assumption that similar features tend to belong to the same class. Therefore, it may not perform as well if the dataset has many unimportant or noisy features, or if the instances are not well clustered.

These classification algorithms can also be affected by the choice of hyperparameters used. Our hyperparameters determine how the algorithm behaves and how well it generalizes to new data. Therefore, choosing the best hyperparameters for each algorithm can greatly impact its accuracy score. If the parameter grids were set up in a different way than we had them, the results could vary slightly, possibly with a different highest accuracy score. Overall, the choice of the best algorithm for a given classification task depends on various factors such as the size and complexity of the dataset, the desired level of accuracy, and the computational resources available.

5. CONCLUSION

In summary, this project aimed to identify the cause behind the disappearance of passengers on a spaceship through a spacetime anomaly by employing machine learning techniques. Our approach involved filling in missing values, creating new features, encoding categorical columns, and normalizing the dataset.

After preprocessing the data, we trained four different machine learning classifiers: KNN, Decision Tree, Gradient Boosting, Random Forest. These classifiers were evaluated based on their classification accuracy to determine which one performed the best. The results of the classification showed that the Gradient Boosting classifier had the highest accuracy of 0.8076, indicating that it was the most suitable model for this problem. This outcome suggests that the Gradient Boosting model was able to identify patterns in the data that other models were not able to capture, making it the most effective in predicting the cause of the disappearance.

In summary, this project demonstrates the effectiveness of using machine learning techniques to solve complex problems by applying data preprocessing techniques and trying different models to identify the most suitable one. The success of this project is a testament to the importance of data preprocessing and model selection in creating effective machine learning solutions.

6. REFERENCES

[1] v-m137, “cs4347-spr2023,” GitHub. [Online]. Available: https://git.txstate.edu/v-m137/cs4347-spr2023.

[2] “6.3. Preprocessing Data,” scikit. [Online]. Available: https://scikit-learn.org/stable/modules/preprocessing.html.