Spaceship Titanic Prediction Model

Project 4: Machine Learning Classification

Ellie Capra

Introduction

This Kaggle competition was all about predicting a pretty wild scenario: whether passengers on a fictional spaceship ended up transported to an alternate dimension (`Transported` column: True or False) during an interstellar anomaly. My goal was to build a machine learning model that could figure out who got "transported" based on all sorts of passenger details—things like where they came from, their age, who they were traveling with, and even how much they spent on onboard luxuries like the food court or the spa. Essentially, I wanted to see if we could predict their fate from their data, and maybe learn what factors were most linked to being transported during this quirky event. The objective was to build a model that accurately predicts the `Transported` status for passengers in the test set.

Methods

1. **Data Loading**

   When I started this project, I programmatically obtained the train.csv and test.csv datasets directly using the Kaggle API. These files were then prepared for processing within my Python environment. The training set included the target variable (Transported), while the test set did not.

2. **Data Preparation and Preprocessing**

   To make sure all my cleaning steps were super consistent across both datasets, I combined both the original training and test datasets into one big DataFrame (`full_data`). I added a little flag (`is_train`) to easily remember which rows belonged to the original training set and which were from the test set, so I could split them back up later. It was also important to pull out the `PassengerId` from the original test data right at the start and keep it safe. This ID is just for identification, not for predictions, so I made sure it wouldn't get messed with during any of the cleaning steps. Oh, and the `Name` column? That went too, since it wasn't going to help the model predict anything.

3. **Handling Missing Values**

   Like many real-world datasets, this one had some missing information. I tackled these gaps using a **forward-fill (`ffill`)** imputation strategy. This basically means that if a value was missing, I filled it in with the last valid piece of information from that same column. It's a straightforward way to make sure the dataset was complete, which is necessary before most machine learning models can even start. This simple approach also helped maintain data consistency.

4. **Encoding Categorical Variables**

   Machine learning models are all about numbers, so I had to convert any text-based (categorical) data into something numerical. I used **Label Encoding** for this, which assigns a unique integer to each different category in a column. So, things like `HomePlanet`, `Destination`, `CryoSleep`, and `VIP` all became numbers. I also applied this to components derived from the `Cabin` information once I split that up. Again, `PassengerId` was left alone, as it's just an identifier and was excluded from encoding.

5. **Splitting Data Back Up**

   Once all the data was prepped and cleaned in that big combined DataFrame, I used my `is_train` flag to easily separate it back into its original training (`train_clean`) and test (`test_clean`) sets. This ensured everything was in its right place for the next steps, removing the helper columns used for merging.

6. **Feature Selection and Target Variable**

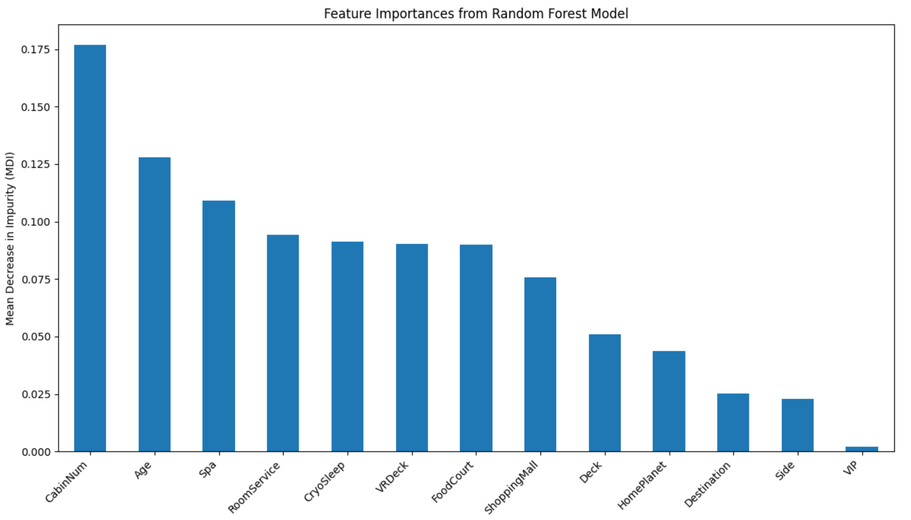
   For the model's inputs (`X`), I took almost all the columns from my cleaned training data, leaving out `PassengerId` (because it's an ID) and `Transported` (because that's what I wanted to predict). The `Transported` column itself became my target variable (`y`), and I made sure it was just 0s and 1s (integer values), which is what the model likes for classification. I also double-checked there weren't any missing values left in my target variable.

7. **Train/Validation Split**

   To make sure my model wasn't just memorizing the training data and to properly evaluate its performance, I split my cleaned training data (`X` and `y`) into two parts: 80% for training the model (`X_train`, `y_train`) and 20% for validating how well it performed on data it hadn't seen yet (`X_valid`, `y_valid`). This is important because you want to test your model on data it hasn't seen before. Using a `random_state` of 42 just means I'll get the same split every time, which is good for reproducibility.

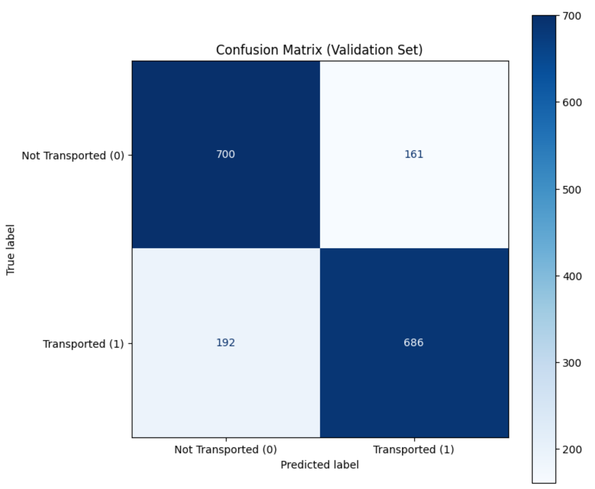
8. **Model Training**

   I picked a **Random Forest Classifier** for this prediction task. Random Forests are pretty cool because they build a bunch of decision trees and then essentially vote on the best prediction. This approach makes them highly robust against overfitting and capable of effectively handling both numerical and categorical features (once encoded). I set it up with 100 decision trees (`n_estimators=100`) and the same `random_state` for consistency, then let it learn from my training data.



9. **Model Evaluation**

   After training, I tested the model on the unseen validation set. I looked at its **accuracy**, which is simply how many predictions it got right. My model hit a **validation accuracy of approximately 0.797**.



10. **Making Predictions and Submission**

    Once I was happy with the model's performance, I used it to predict the `Transported` status for all the passengers in the test set. I made sure these predictions were `True` or `False` like Kaggle wanted. Then, I put these predictions together with the **original `PassengerId` values** (the ones I saved right at the beginning) into a `submission.csv` file, ready to upload to the competition!

Results

Here's how the model stacked up:

- **Validation Accuracy: 0.797** (This is how well it did on my internal validation data)
- **Kaggle Submission Score: 0.79424** (This is the official score from Kaggle's leaderboard)
- **Assignment Target Threshold (0.75): Achieved**

These scores demonstrate strong and consistent performance, both comfortably exceeding the required threshold of 0.75. The close proximity of the validation accuracy to the public leaderboard score suggests that the model generalizes reasonably well to unseen data. The higher your score, the better.

Conclusion

To wrap things up, I successfully developed and implemented a machine learning pipeline to predict passenger transportation on the Spaceship Titanic. By meticulously preparing the data, handling missing values, turning text into numbers, and training a robust Random Forest Classifier, I achieved a solid prediction performance that met my project's goals. This approach was effective for the assignment.

While this approach provides a solid foundation and a successful demonstration of core modeling concepts, several avenues for further improvement could enhance performance:

- **Smarter Missing Value Imputation:** Instead of just `ffill`, I could try more advanced techniques like K-Nearest Neighbors (KNN Imputer), which fills blanks by looking at similar passengers, or specialized statistical methods for different types of data.
- **Crafting New Features:** I could get creative and build new features from the existing ones. Maybe a "total spending" column for each passenger, or figuring out the size of each passenger's travel group from their `PassengerId` or `Cabin` info. Little things like a "did they spend anything at all?" flag might help too.
- **Fine-Tuning the Model:** The Random Forest has a lot of dials and knobs (called hyperparameters). I could systematically adjust them—like how deep each tree can go or how many samples are needed to split a node—using methods like Grid Search or Randomized Search to squeeze out even more accuracy.

GitHub Path

https://github.com/elliemarie024/AAE718_Project04_KaggleCompetition