Spaceship Titanic Problem - Kaggle

Jeremy Michael Becker, Benjamin Comer, Adit Mehta - CS273P Project Report

Our submitted notebook can be read here:
https://github.com/BenComer98/kaggle-spaceshiptitanic

Introduction *(Ben)*

We tried to model the Spaceship Titanic problem on Kaggle. This problem has a dataset of passengers on a spaceship who are potentially transported to an alternate dimension, and many attributes assigned to those passengers that we can use to predict whether they were transported or not. We considered and used many models for this task, and landed on ensembling over many common models, each with very good performance (finishing in the top half of the leaderboard). In this report, we'll cover the models we chose, our implementation specifications, our results, and potential areas of improvement.

Throughout this report, sections with a body will be suffixed by a name of the person or people who wrote that part of the report. For details on implementation tasks, see the last paragraph.

Models

Pre-Processing *(Adit)* Before training the models, we had to do some pre-processing on the data. Firstly, we dropped some columns we thought were not useful to predict the output. Next, there were some missing values in the data. For the numerical values we replaced the missing with the average of that column (except for age, which we replaced using the mode). For the categorical column we did One-Hot Encoding to better handle categorical data. We also split up the Cabin string into three, including the cabin side and deck, discarding the regular string, to retrieve data in a more useful format.

Neural Networks *(Jeremy)* Neural Networks, more specifically multilayer perceptrons, were created to solve a classification problem. Six different neural network models were trained with the Spaceship Titanic dataset provided by Kaggle. The models consist of three sets of two models to showcase the various activation functions commonly used in neural networks such as the ReLu activation function, tanh activation function, and the logistic activation function. Afterwards these sets of models were broken down into two models each with 2 and 4 hidden layers respectively.

Support Vector Machines *(Adit)* Support Vector Machines were also used to solve the Spaceship Titanic dataset's classification problem. SVMs are usually good for handling high-dimensional data and can successfully separate the data by identifying the best hyperplane that maximizes the margin between classes. Two different SVM models were trained, using the Radial Basis Function (RBF) and Sigmoid kernels to analyze their respective impacts on classification performance.

Naive Bayes ***(Adit)*** A Naive Bayes classifier was additionally used to create a probabilistic model for the Spaceship Titanic dataset. Built on the assumption of feature independence, this model efficiently computed class probabilities using Bayes' theorem.

Decision Trees, Random Forests ***(Ben)*** Decision Trees and Random Forests were implemented to compose a relatively simple model of the Spaceship Titanic problem, at least mathematically. In doing so, we could also easily view the best features for Entropy analysis, and construct visuals of the tree, which were implemented in previous versions.

Ensemble Models ***(Ben)*** To make use of all of the above work, we utilized two ensemble models to increase safety against overfitting and increase model diversity, Voting and Stacking classifiers. The Voting classifier is a simple majority vote aggregator of the above models, and Stacking combines all those models in a way to make the best predictions, based on its fitting.

Implementation

Neural Networks ***(Jeremy)*** For the six neural network models we used sklearn’s MLPClassifier. We created the six models by changing the hyperparameters where we used different activation functions and a different amount of hidden layers.

Support Vector Machines ***(Adit)*** For the Support Vector Machine models, we used sklearn’s SVC classifier. We created two models using two different kernels, the RBF and the sigmoid. We also tried cross validation with 10 splits and standard train test split to see which performed better on both the models. Both of these models’ specifications were supported by that tuning.

Naive Bayes ***(Adit)*** For the Naive Bayes models, we used sklearn’s GaussianNB classifier to build a probabilistic classification model. We also tried cross validation with different split values and standard train test split to see which performed better for this model.

Decision Trees, Random Forests ***(Ben)*** We simply used sklearn’s decision tree and random forest model frameworks. On decision trees, we depth-limited the tree to 5 nodes down. As such, we were able to prevent overfitting while serving adequate complexity to capture the finer details of the model. We only depth-limited the random forests by 13 for the opposite reason, because we wanted to capture individual points in the forest. Both of these approaches were also supported by hyperparameter tuning.

Ensemble Models *(Ben)* Once again, we simply used sklearn's models, this time the voting classifier and stacking classifier models. Because of some of our models' non-probabilistic nature, we weren't able to do soft voting, but we simply passed in all of our trained models as voters and let it take care of the results.

Results *(Jeremy)*

Model	**Accuracy**	**Recall**
2-Layer ReLu MLP Classifier	0.79	0.77
4-Layer ReLu MLP Classifier	0.80	0.84
2-Layer Tanh MLP Classifier	0.79	0.81
4-Layer Tanh MLP Classifier	0.79	0.79
2-Layer Logistic MLP Classifier	0.78	0.73
4-Layer Logistic MLP Classifier	0.78	0.76
Naive Bayes Classifier	0.74	0.90
SVM RBF Classifier	0.79	0.88
SVM Sigmoid Classifier	0.73	0.73
Decision Tree Classifier	0.78	0.89
Random Forest Classifier	0.79	0.77
Voting Classifier	0.81	0.84
Stacking Classifier	0.80	0.81

Table 1. Accuracy Comparison

When testing out each of our models we chose to focus on two main metrics: accuracy and recall; Provided in Table 1 are the results for each of our models. Notably, accuracy and recall weren't very close in many cases, and only some models had similar accuracy and recall. For example, we saw closeness in our accuracy and recall for all of our neural network models, SVM sigmoid classifier, random forest classifier, voting classifier, and stacking classifier. The largest difference happened in our naive bayes classifier, SVM RBF classifier, and decision tree classifier. These specific models could be predicting true negatives incorrectly, moreso giving false positives than false negatives. Choosing whether to focus on accuracy or recall ultimately depends on what our dataset is, and in this case I believe accuracy should be centered while using recall to understand the nature of our misses.

For our accuracy, we observed that the majority of our models tend to have an accuracy between 78% and 81%, with the exception of our Naive Bayes model and our SVM Sigmoid Classifier. The model that performed the best is our voting classifier with an accuracy score of 81%. However, if we focus on non-ensemble models, then our best performing model is the 4-Layer neural network with ReLu activation function, with an accuracy of 80%. Overall our models performed well with this dataset, as our accuracy range was between 73% and 81%, especially given that the top performers (aside from cheaters) achieved about 82% on the test data.

Areas for improvement *(Ben)*

There are too many models to consider here, but the idea of further tuning our ensemble methods is appealing. I believe that given time, we could have adjusted our selection of models for the voting classifier, or used a larger ensemble with Stacking, using models with different hyperparameters, to further churn out percentage points of test accuracy.

Additionally, in regards to our validation process, we used regular training and validation splitting for our analysis (for our submission, we removed this to get the most training data possible), when we all could have used cross-validation. Notably, cross-validation was used to tune the SVM and Naive Bayes classifiers, but not for the others. This would have brought better tuning to our hyperparameters, and while it would not improve our submission, it could have brought with it ideas for future models to ensemble.

Contributions *(Ben)*

Jeremy Michael Becker: All neural network implementation, confusion matrix code, collection of analysis data and reporting

Ben Comer: Decision tree and random forest implementation, removing validation data for submission

Adit Mehta: Naive Bayes, SVM implementation, cleanup of notebook for reporting analysis of individual models

Pair Programmed: Data preprocessing, ensemble methods, report submission