Thomas Lamont
CS3270 p2 – phase 1

Last project Nick and Dillon, and I used several, more primitive, forms of machine learning to predict whether passengers on the Spaceship Titanic were transported away from a wormhole. My goal is to implement deep learning, a more sophisticated tool, to make these predictions. With an established direction towards a Long Short-Term Memory model, the best place to start on reworking the project was at the beginning. To further improve my past models, I had some ideas for feature engineering and preprocessing that were underexplored in the first iteration.

To begin, I downloaded a fresh version of the test and training datasets from Kaggle. I combined the test and training datasets so that I could perform the same preprocessing steps on both the training data and the eventual test data for making a prediction. The first step was to split out the unwieldy 'Cabin' feature back into three features, the cabin number, deck letter and side of the ship. This initial step was carried over from the initial project and was important for the next preprocessing steps. It was also important to pull out the group numbers from the 'PassengerId', I also added a new feature, group size, which was filled with the total number of group members between anyone with matching group numbers across the test and training data. No passengers were missing their Id, so the data is now in a state where it needs to start guessing the values of missing features, based off other features.

Feature:	**# Passengers missing data**
PassengerId	0
HomePlanet	288
CryoSleep	310
Cabin	299
Destination	274
Age	270
VIP	296
RoomService	263
FoodCourt	289
ShoppingMall	306
Spa	284
VRDeck	268
Name	294

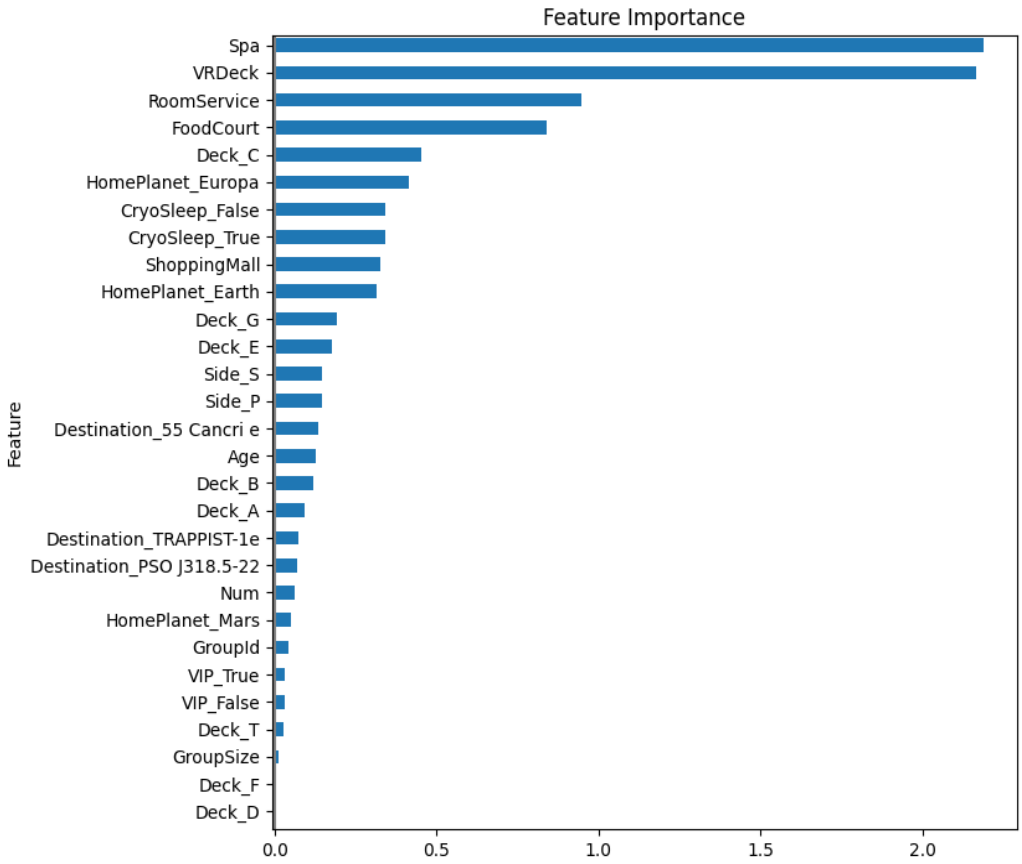
Thomas Lamont
CS3270 p2 – phase 1

About 300 data points were missing from each category, this is valuable information for training and if the model were to ignore it, or infer it incorrectly, it would lead to a loss of accuracy. The first step was to fill in CryoSleep and spending. This was done similar to how it was done in the first project, anyone in CryoSleep with missing spending data had the missing values set to 0, likewise anyone missing their CryoSleep data that had 0 spending was assumed to be in CryoSleep. This left every CryoSleep value filled in, but some spending columns were still missing for the awake passengers. In the first iteration, these missing values were just filled with the overall median value for the feature, which did not consider that the median had been influenced by the large number of 0 spending, a consequence of being in CryoSleep. For this iteration of the project, I took a slightly more sophisticated approach and set the missing spending values to be the median of the non-zero values, this way the value is much closer to the actual median for awake passengers. After CryoSleep was filled in, I wanted to be more sophisticated in how I filled in the other missing values, rather than just using the median values through one hot encoder, I used the data of group members to inform the data. I assumed that anyone in a group would have the same VIP status, Home Planet, Destination Planet and Cabin information as the others in their group, to do this I gathered the group data into a data frame, then filled in any missing values I could from the data frame. Finally, once the more educated guessing was out of the way, I filled in the rest of the missing values, for solo passengers, with the median values of the overall data. Once this step is completed, there is no longer any passengers missing data, in the future steps, I will not need to include any further encoding, since no values are missing.

The next step in my plan of action was to use get dummies to change categorical features, such as Cabin side, deck, home and destination planet into binary features, for instance rather than having a single Cabin deck feature, the decks were split to true and false values for whether a cabin was on a specific deck or not. After splitting the data, I updated the old logistic regression model to work with the new data and rand the same grid search and k-fold testing to compare accuracies, there was about a 1% increase at this point, which is substantial and was a promising outcome from the preprocessing.

Creating dummies and splitting the features simplifies the individual features, but increases the number of features, and thus the dimensionality of the problem. The new data has 30 features, up from 17 before the split. An idea to combat this dimensionality blow up is another step in the feature engineering process, maybe a subset of the features is as good at predicting the solution, or even better, than looking at the entire set. Some of these features could be noise that the models are mistakenly taking into account despite them just being a way to overfit on the training data. It would take too long to try dropping out every combination of features and test every subset, so I created 11 sets of features

and tested every subset of features to see if the dimensionality could be reduced. The feature sets were age, spending, group id, group size, home planet, destination planet, CryoSleep, vip, cabin side, number, and deck, many of these features had been expanded out using get dummies so it made sense to group them back together for this step. Again, using logistic regression as a baseline, every subset was dropped, and its results analyzed compared to the base case with all features. For reference, before any features were dropped the logistic regression model had a ~79.282% accuracy, once the various subsets of features were dropped, the highest performing model dropped the cabin decks and cabin numbers, showing a slight increase in accuracy to ~79.512%. This increase was on the borderline of significance in my opinion, so I attempted all future models both with and without dropping these features. Some things I took away from this step is that it is likely the preprocessing for the cabin numbers and decks did a worse job than I would have liked at filling the proper values. It is of course also possible that this data is noise, or maybe that some deck values and numbers would have been more telling if I had fully tested every subset of features, rather than grouping them together. To highlight the importance of the different features, I plotted a graph that trained a logistic regression model, standardized the features and plotted each feature with the absolute value of its coefficient, and hence importance for the model training.



It is clear that the spending features are crucial to the model's prediction with things like destination planet and cabin numbers being taken less into account. I was surprised to see how little VIP status was considered by the model.

With these improvements on the previous project being implemented, it was time to move to deep learning. I started with setting up a Long Short-Term Memory model. I used tensorflow's LSTM model. Implementing LSTM is not as straightforward as other models, due to the need for samples, features, and timesteps. This required reshaping the features to fit the model. The timesteps would normally be a sequence length, informing the model how many steps are sent to the network for a complete forward and backward pass. Since there is nothing in the model that is a time series the data is reshaped to have one timestep per sequence, essentially treating each feature vector as a single timestep in a sequence. Once the data was reshaped it could be used in the model. The initial accuracy of the LSTM model was ~78.032%, a sizable decrease from the logistic regression, this was with one set of parameters chosen arbitrarily. TensorFlow's LSTM model was not compatible with grid search that I had used in the simpler models, so I implemented my own version of grid search for the model. I set up a parameter grid of model parameters then used nested for loops to try each combination of parameters, this was an easy way to get the same effect as grid search, and after the grid search, the best performing LSTM model had an accuracy of ~78.834%, closer to the numbers seen from logistic regression. Since the reshaping and grid search was already set up for TensorFlow's model, I decided to quickly implement a normal recurrent neural network, and got slightly better results, ~79.241% accuracy.

Both deep learning models performed fairly well but fell a little short of the numbers seen by the simpler models. With everything set up I moved from just using the training data to making real predictions, using the Kaggle test data without known classifications. Our final model from project one was the ensemble prediction taking a majority vote between a neural network, random forest, and logistic regression model. This model reached 80.710% accuracy. I started with pure deep learning, making a prediction with just the RNN, the accuracy on the test data was 79.869%, so the RNN performed similarly on the training and test data, a good sign there was not too much overfitting on the train data. Next was time for the ensemble prediction. I took the previous three models and added my LSTM and RNN to the majority vote, without dropping the decks and cabin numbers, the ensemble had an 80.289% accuracy, and with the features dropped it had 80.313% accuracy.

In the end the final ensemble failed to reach the heights of the previous ensemble. I have a few theories on why this may have been the case. I think its likely that the RNN and

Thomas Lamont
CS3270 p2 – phase 1

LSTM were overengineering the problem, since there was no natural time step in the data, having to hack the data to fit the model was not a boon to the accuracy. The traditional models are best at capturing relations and making predictions based on the individual features, while the deep learning models excel at capturing patterns from more sequential data. Deep learning models are powerful tools to keep in mind in the future for predicting data with sequential elements. Overall, it seems like overfitting was not a problem with the models I implemented, the model's accuracy was always pretty close between the testing and training data, so I can be pretty confident that each model's accuracy is as stated, rather than being boosted by using training data.