If you’ve ever heard of Kaggle before, I can practically guarantee that you’ve heard of the Titanic challenge. The premise is simple: given a set of passengers and a set of attributes like their age, gender, and ticket class, can you predict which of them survived? Before we begin, I’d like to preface by saying that it is difficult to apply any particular conclusions from this data to other data science problems you may face in the future, since the Titanic dataset is very much a toy dataset. However, at the end of this seminar, you should expect to gain a greater understanding of various data preprocessing and machine learning techniques. Hopefully, I will be able to make you reconsider some things you may have taken for granted in your thought process. The Titanic dataset will merely be the sandbox I use to demonstrate and compare and contrast various techniques. So, let’s begin.

==============

The data comes in two datasets; namely ”train.csv” and ”test.csv”. There are 891 datapoints in the training set and 418 in the test set. The small size of this dataset means we can afford to run some procedures that would otherwise be computationally expensive, like 10-fold cross validation to estimate test error. Each row in the datasets corresponds to a unique passenger with a ”PassengerID” identifier column and 10 features/predictors. The values of these predictors are a mix of both numerical and categorical, so it’s an excellent way to get started in dealing with different kinds of data. In the ”train.csv” dataset, there is an additional column called ”Survived” which contains values in the set {0, 1} describing whether that passenger survived the sinking of the Titanic (0 if they didn’t, 1 if they did).

===

Arguably, the most important part of any machine learning procedure is to ensure that your data is properly preprocessed. No machine learning algorithm will give you good results on unclean data no matter how cutting-edge it is. The principle I abide by is; make sure you understand the data first before you feed it into any sort of model; otherwise you may make error-prone conclusions. Now, since machine learning algorithms tend to accept numerical data, and as such, I had to ensure that there were no non-numerical values anywhere in the training and test datasets. I will outline the approach I used to transform the training dataset into a ”machine-learning-ready” state. The steps involve:

1. Cleaning data
2. Feature engineering
3. Diagnostic plots

===

The first thing I did was to check which columns of the data contained missing, as in NA, values. In the training dataset, I noticed 177 missing values in ”Age”, 687 values missing in ”Cabin”, and 2 values missing in ”Embarked”. The general approach I adopted was to fill in missing data according to the distribution of the remaining data.

In the ”Embarked” section, I noticed a correlation between fare price, Pclass and Embarked. As such, I evaluated the median fare price for each Pclass/Embarked pair, and filled in the missing values according to whichever pair’s fare was closest to the fare of the missing values’.

===

In the ”Age” section just under 20% of the ages are missing. As we can see from the distribution of the ages, if we use a more naive strategy like mean or median replacement, they are heavily centered around the middle, which is inconsistent with the original distribution.

Therefore, I needed to implement a way to fill in the missing values such that the resultant distribution is at least more similar to the original. The naive way would be to compute the original distribution function, then assign ’Age’ values to missing values randomly according to that distribution. However, that fails to take into account the idea that the age distribution is likely correlated with some other predictors. For instance, an individual with ”Mrs” in their name you would not expect to be under 18, but the naive approach would not take that into consideration.

Therefore, I applied regression on each missing age value. Each missing Age value would be defined to be the response variable Y in some regression involving predictors drawn from other columns of the training set, and I repeated this regression procedure until all the missing values are filled. To do this, I chose to implement the MICE (Multivariate Imputation by Chained Equation) procedure to predict a distribution and randomly select values from it to use as the missing values. This method takes into account the relations between the OTHER predictors that may be correlated with different ages, and should therefore perform better than the naive approach. Note that this is just one approach and there are other approaches to missing data imputation, including class-based imputation or the aforementioned median-based or mean-based imputation. As you can see from the graph, the purple curve, which represents the data with MICE input, is much more similar to the blue curve, representing the distribution of the existing data.

====

Since there were so many distinct values in the Cabin column, I concluded there was no information to be gained from the specific cabin number. Instead, I decided to extract the Deck each passenger was situated at. As the majority of the “Cabin” data was missing, it would not be prudent to fill in missing data according to the distribution of the existing values. Instead, all missing values were filled in with an X.

The Deck and Embarked variables were encoded in indicator columns. The reason I chose not to encode the Deck/Embarked variable in a single column was because I could not safely assume that fixed increment between categories. For example, regression based models might perform poorly since there is no evidence to suggest that the increment in chance of survival between two categories, say embarked at C and embarked at Q, is the same as the increment in chance of survival between another two categories like embarked at Q and embarked at S.

An alternative method to encoding the Deck feature is setting an indicator column=1 if it was part of the original data, and 0 otherwise. However this would result in loss of information (which Deck a passenger belonged to), and a passenger’s location on the ship might be relevant to determining whether they survived.

Since the Name feature is useless, I decided to extract the title of each individual, to generate the Honorific column. Honorifics range from Mr, Mrs, Countess, etc. I created two features from this: isMarried, which equaled 1 for married women and 0 otherwise, and Unique to denote if an individual had a rarer titles like Dr, Rev., etc.

====

A correlation heatmap of the features show that the most important features in the model (highest absolute value of correlation) are Sex and Survived (-0.54), isMarried and Survived (0.34), Pclass and Survived (-0.34) and Fare and Survived (0.26). None of the correlations are particularly strong, but we observe a high negative correlation between values of the various indicator columns (which makes sense, since if a value is 1 in Embarked 0, it must be 0 for all the other Embarked indicator columns.) Surprisingly, I also observed that there is a strong positive correlation between Deck X and Pclass (0.73) which is slightly reflected in the correlation between Deck X and Survived (-0.32). However, I wasn’t assured of the reliability of Deck X as a predictor since Deck X denoted the missing values which made up a large proportion of the dataset.

The second plot showed the survival rates by gender in each age bracket. You will notice that females over the age of 20 are disproportionately more likely to survive than males of the same age range. However, males and females under the age of 20 are approximately equally likely to survive (likely on account of them being children/minors and given priority.)

Now, based off the high correlations indicated in the correlation map, I also compared the number of those that survived vs those that didn’t for various features. The ratio of individuals that survived to those that didn’t is the highest for Pclass=1, for married women, and for individuals with unique honorifics. Additionally, those that had one or more relative were also relatively likely to survive compared to if they didn’t.

Now, since there is also a high correlation (0.73) between PClass and Deck X, to avoid collinearity, we should remove one of these features. I ended up removing Deck X and Embarked\_2. This gives us a total of 17 features, all in numeric form, to be fed into our model.

====

I tried 3 different model structures for prediction - KNN, logistic regression, and random forest. We ignored single decision trees since they are susceptible to changes in training data and may overfit. It’s worth noting I did not experiment with Naive Bayes since I believed it to be simple, and did not bother with neural networks since I believed the model would be too complicated for this dataset. However, this assumption may be false.

====

The KNN model basically chooses which class that point belongs to based on the majority vote of that point’s k closest neighbours. I tried the KNN model with hyperparameters K=1 up to 29. The initial training accuracy of the KNN model was 0.945, which initially seems extremely good. However, this heavily implies that the k-NN model overfits to the training data, because when K=1 the probability of one sample is estimated ONLY on ONE OTHER sample, which by definition is very susceptible to noise and outliers. As such I could not assume this model’s training accuracy to be indicative of its performance on the unknown test set. Indeed, the model had high training accuracy the lower the value of K. Many sources online debate on the optimal value of K but a common choice for a dataset with n points is the square root of n, and in our case n = 29. Cross validation was used to estimate the test error, and the 10-fold mean cross validation score was 70.9%.

=====

In logistic regression, we once again used 10-fold cross validation to estimate the test error, and the average validation accuracy of the model (using the L2-norm and all features) was 82.5%, which was a significant improvement on KNN.

====

In random forest model, I arbitrarily chose the hyperparameters and implemented backwards subset selection to select the best list of predictors. ‘Importance’ of a predictor is given by its Gini importance, which is the mean decrease in Gini index. Larger values of ‘importance’ indicate a more important predictor. As we can see, the most important features are Sex, Ages, Pclass, isMarried, SibSp. For the remaining features whose importance was less than 0.05, I deemed them relatively insignificant and conducted backwards subset selection, dropping the least important feature and computing the 10-fold mean CV. I then attempted to choose the simplest model with a 10-fold mean CV 1SD within the highest 10-fold mean CV score, but all of the models actually satisfied this condition, so I decided to drop all predictors except Parent/Child since there is a significant increase in importance between it and the other predictors. The final model has 10-fold mean CV score of 83.5%, which is a small improvement on the logistic regression model and also has the benefit of not overfitting to the less significant features.

===

Now, the random forest model seems decent, but I found that the test accuracy was only around 71%. As such, there is clearly much room for improvement. We’ve already removed many features from the model by considering them insignificant, so model either underfitting without them (unlikely) or the original hyperparameter choice caused overfitting.

The solution is to further tune hyperparameters, such as the depth of the trees in the random forest, and/or explore alternative models. For instance, AdaBoost is a boosting learning model that heavily penalizes misclassified points since the loss function is exponential and I believed that it would help improve the model.

======

I chose to implement grid search for hyperparameters on the AdaBoost classifier with Decision Tree with maximum depth 1 as the base model, using the same features as in the random forest model. During intermediate stages of refining, I actually did try exploring learning rates over 1 and found that around the 1.7 threshold, the training accuracy started to decrease to the mid 60%s, hence they are not included in this section. A search of the hyperparameter space revealed that the hyperparameter set specifying the simplest model within 1SD of the maximum 10-fold mean CV score was learning rate =0.7 and number of estimators=10. I also noticed that a higher number of estimators corresponded to a slightly higher 10-fold mean CV score - I will demonstrate how this is misleading later on. On the full set of predictors, this model had 71% accuracy on the test set, slightly lower than our original random forest result.

Understandably, I was unsatisfied and tried to add backwards subset selection in conjunction with grid search for hyperparameters to see if I could improve the accuracy, but found that the simplified model seemed to perform even more poorly, with a test accuracy of 59.33%, suggesting that in a boosting model, the features which the random forest deemed as irrelevant may have actually been relevant. This is quite important since the base model for my AdaBoost implementation used a decision tree with a maximum depth of 1 and excluding many features would likely cause it to perform much more poorly, especially given that the irrelevant features were only unnecessary in a random model because the model had sufficient depth.

===

To counteract the lack of features, I attempted to use the full set of predictors in Adaboost with the highest 10-fold mean CV score, namely with learning rate=0.7 and n\_estimators=50, but only got a test accuracy of 40%. Something to be learned from this is that good mean CV score is not necessarily indicative of good test performance since this model essentially has 50 estimators stacked on top of one another. Indeed, this matches with the conclusion in the lecture notes that an overly large M (number of estimators) can lead to overfitting of the training data and hence suboptimal performance on the test set.

Ultimately, AdaBoost seems like a dead end since removing predictors causes underfitting, and increasing number of estimators causes overfitting. As such, I decided to backtrack to existing models.

===

Returning to our logistic regression model, I evaluated my baseline model in Part (5) with full set of predictors and found it does surprisingly well, with 72.01% accuracy on the test set, and is marginally better than our original random forest model. I then asked if it would be possible to try and simplify the logistic regression model to prevent the model from overfitting on irrelevant features and avoid collinearity, just like in the random forest case. To do that, I calculated the variance inflation factor or VIF, which is the ratio of variance in estimating some parameter in a model to the variance of the model constructed with only one term. The most common threshold indicating multicollinearity is if any one of the VIFs is >10. In this case, none of them do, so to verify this, I ended up using backwards subset selection again and found that removing predictors does seem to have a statistically significant effect on the model, since the test accuracy is reduced to 63%.

As such, there’s not much to be done to improve logistic regression model either short of redesigning the features or perhaps transforming some existing features to accept quadratic terms etc.

=====

Returning to our random forest model on a subset of predictors, I conducted hyperparameter grid search my baseline model, by generally using shallower trees so as to avoid overfitting and trees being too collinear with each other. The model with the best 10-fold mean CV score was with 60 estimators and max\_depth=5, but I ended up choosing the simplest model with 30 estimators and max\_depth=5. Test accuracy of this model was 68%, a slight drop off. As such, this indicated that the maximum depth parameter in random forests tends to be extremely sensitive towards data. As such, I opted to select another model whose 10-fold mean CV score was within 1SD of the maximum with 150 estimators and max depth of 2. This ended up performing well on the test set with accuracy of 77%, a significant improvement over the baseline. Generally, if the max depth is too high, then the model is extremely prone to overfitting and performing poorly on the test set.

Furthermore, the training accuracy seems to improve as the number of estimators increases; that is to say, the amount of overfitting does NOT increase with more trees, only when the trees are deeper.

===

Future analysis

77% is actually a fairly reasonable place to stop especially on such a small dataset as this. Although the Kaggle rankings show a lot of people getting 100%, it is very possible to cheat this competition by just flipping bits in the submission since no code has to be submitted. Furthermore, besides the concepts, there is not much carryover of a tuned Titanic model to other data science problems.

However, if analysis were to continue, additional feature engineering could be employed. Perhaps the Name category could be used to detect husband-wife pairs with the same last name and make the isMarried category include both married men and women, although this would leave out spouses who chose to travel alone or those men whose wives did not take their husband’s name. Also, the title “Master” actually used to refer to young boys/men, and as we have seen individuals under the age of 18 tend to have higher chances of survival in both genders, so this is very likely to be a relevant feature. Finally, we could separate SibSp and ParCh into features Sibling, Spouse, Parent, Child to see if there is additional correlation (those that had parents travelling them were likely children and hence more likely to survive.)

Other forms of subset selection could be used, like backward-forward elimination.

Finally, I decided not to use the F1 score as a metric because that should only be used where there is significant class imbalance (there is only a 2:1 ratio in this dataset) and true positive detection of one class is much more important than false negatives of the other class.

===

In the end, machine learning is not a one-and-done process. Just like software development, continual refinement and tuning of the procedures you employ can not only yield better results but help you become a more well-rounded data scientist who is skilled in multiple areas. I have only demonstrated a small subset of the nearly infinite tools and techniques that you can use on your data, and if you believe your current results aren’t up to your standards, just try another approach. I hope this has been informative and eye-opening to some degree and that you have gained a greater appreciation for not just the accuracy of your machine learning models, but understanding and evaluating the importance of the assumptions that you’ve made to get there. Thank you for attending my presentation.