Decision Management Community Challenge August 2021

Titanic Booking Service: Perfect, Near Perfect and Imperfect Solutions

(Bob Moore, JETset Business Consulting, 10th September 2021)

1 Problem Statement (from the web site)

*Kaddle is a popular competition website for Machine Learning (ML) professionals. Its legendary Titanic competition is quite simple: use ML to create a model that predicts which passengers survived the Titanic shipwreck. Kaddle offered two CSV files: **train.csv** – a list of 891 passengers with their various characteristics like age, sex, ticket class, fare, and more; **test.csv** – a list of other 418 passengers with similar characteristics. Both lists include the field "**Survived**" that contains 1 if the passenger survived or 0 – if not. The first list should be used as a training set to discover the survival rules, and the second list should be used only to test the accuracy of the discovered survival rules. Note that not all characteristics are known, e.g. for some passengers age is not specified. Many good solutions have been provided since 2012.*

In our challenge, we want you to download and use the same two csv-files to create and test a special decision service called "Titanic Booking Service". First, you should create a decision model with rules that produce one of the following advices for each passenger: 1) Bon Voyage; 2) Go at your own risk; 3) Don't do it!

You can create rules using any ML tool or manually following your own analysis of the data and your interpretation of the Titanic tragedy ("hints from the future"). You even may borrow some rules discovered by Kaddle's competitors. What you cannot do is to feed your ML tool with passengers who should be used for testing only.

At the end your decision model should apply these rules against every of 418 test-passengers, giving them a booking "advice", comparing them with what actually happened to each passenger, and producing the summary with total numbers of Good and Bad advices.

2 A perfect solution and one which is almost perfect

At first sight the challenge sounds plausible, but it quickly becomes obvious that there are problems with what we seem to be asked to do. A quick look at the data shows that of the 891 passengers in our training data, only 38% of them survived (and only 36% of the passengers in the test data set[1]). No one in their right mind would set sail on a boat with these odds of survival. So, we can reasonably propose the following solution:

Solution 1:

Decision Service has following rules:

IF true
THEN advice = “Don’t do it!”

Total number of good advices for test cases = 418 (100%)

It’s perfect because it always gives good advice – don’t go. Even the passengers who beat the odds and survived had a very traumatic experience generally losing their possessions and, in many cases, their loved ones too. But for this to make sense, the decision service needs to know *in advance* that the Titanic will sink.

So, what’s the alternative? Well, if back in 1912, I had been planning to take a trip from Southampton to New York, I could reasonably have put a good degree of faith the assertion that the Titanic really was ‘unsinkable’, in much the same way that today I have faith that I will survive if I book myself on a flight from London to New York. No ship is truly unsinkable, but we could reasonable assume this might mean the chance of *not* surviving a trip on the Titanic was small - say 1 in 1,000[2]. So, there is no good reason to *not* buy a ticket (on the basis of survivability). If we build a decision system built on this analysis, we get the following solution:

Solution 2:

Decision Service has following rules:

IF true
THEN advice = “Bon Voyage”

Total number of good advices for test cases ≈ 418 (99.9%)

This solution is ‘almost’ perfect, because the information we have at the point we give the advice is so strongly weighted in favour of going, there is no justification not to[3].

[1] There is good reason to be decidedly dubious about the test set by the way. It took me a long time to notice, but finally I discovered there is a significant peculiarity about it. In it, every female passenger survives, and every male passenger dies, so it is clearly not a random sample!

[2] For comparison according to FlyFright (https://flyfright.com/statistics/) the chance of dying in an air crash is less than 1 in 3 billion, but people are still scared of flying, Conversely travel safety at sea in the early twentieth century was not very high, but plenty of people still risked it.

[3] It is also worth noting that the advice here is quite independent of the data for any passenger. We are only looking at one random event, the ship sinking or not, so we give the same advice to everyone.

3 Towards an Imperfect (but More Interesting) Solution

One might feel the arguments above are a little spurious, but the point here is that if we are building a decision system, we need to understand what kind of decisions we can reasonably make. We *cannot* make use of a predictive model unless we get to a point where we have the data. And we cannot see into the future. The challenge seems to posit we know which people died before the ship set sail. But no one who boarded the Titanic for its maiden voyage *knew* it would sink. If they had known, they would not have walked up the gangplank. Equally it seems likely that virtually none of the passengers considered the ship sinking as anything other than very unlikely when they embarked.

So, if we are going to get as far as doing any Machine Learning, we need to consider the problem in a slightly different way. But, before we get onto that, let think about how we imagine this decision service is going to work. It seems plausible that we expect the service to advise '*Bon Voyage'* if our chances of surviving the journey are high, that is to say above some (as yet) unspecified threshold. Likewise, it should tell us '*Go at your own risk'* if our chances of surviving the journey are moderate and *'Don't do it!'* if our chances of surviving the journey are unacceptably low. If this is what we expect the decision service to do, we need to work out a passenger's chances of survival.

Now let us make the following assumptions:

1) We have developed a classification 'model' which predicts if a passenger will survive if a ship 'like' the Titanic sinks[4]. So, we can consider this model to be a function $\mathcal{M}$, such that if x, is the data for a particular passenger (a row in the test or training files) then $\mathcal{M}\,(x)$, will either be 1 if the passenger is predicted to survive, or 0 if they are predicted to die.

2) Based on past shipping history we have established a reasonably good estimate of the probability of a ship 'like' the Titanic sinking on a voyage from Southampton to New York. From the arguments in favour of Solution 2, this should be 'quite' small (whatever that might mean).

These assumptions should allow us to calculate our survival probabilities and hence build a decision service which makes some kind of sense. But it does mean our decision service is not really for booking on the Titanic itself. It is for booking on a ship 'like' the Titanic (making a voyage in almost identical circumstances to the Titanic's maiden voyage). And that the decision service is based on the historical records of which passengers survived previous voyages of ships 'like' the Titanic (including one which sank).

Currently, we have neither the model of assumption 1, nor the probability of assumption 2. We will address the first omission in the next section but will only 'guess' at an

[4] Of course, any model we come up with, based on the Kaggle (not Kaddle by the way!) data sets will have baked in a huge number of contingent conditions, for example that the sinking takes place in the middle of the night in the North, rather than (say) at midday in New York Harbor – when survival rates would probably have been much higher.

answer to the second[5]. But before we try to create a model, let us think about how to use it. The model predicts if a given passenger will survive given that they are on a ship ‘like’ the Titanic and that ship sinks[6]. We are not going to get a perfect prediction. On the Kaggle website it is suggested that any models having an accuracy of 80% or more on the training set, are likely overfitting, and so are unlikely to work as well as that on the test set. The trained model should give us estimates for the probability of it making an accurate prediction (true-positive and true-negative) and of an inaccurate prediction (false-positive and false-negative). With these probabilities, the data for an individual passenger and the probability of the ship sinking in the first place, we should be able to come up with an estimate of the probability of that passenger surviving the trip based on the model prediction combined with the actual chance that the ship sinks.

How do we go about this? Well, we have five pieces of data which are independent of any individual passenger, namely the four marginal probabilities we estimate from our model accuracy, and the probability of the ship sinking in the first place. We also have one piece of data which is passenger dependent, namely the prediction of the model. Let’s put these together and go for a formal argument. We have three ‘events’ of interest:

1) The passenger survives – denoted AS (actually survives)
2) The model returns 1, (so it predicts the passenger survives) – denoted PS
3) The ship sinks – denoted SINKS

We are interested in the probability of the event AS. Since the only passenger dependent input to this is the outcome of the model, we only have two cases to consider, one where the model predicts survival, and one where it does not. In the normal notation of conditional probabilities, we can express what we are looking at as:

$$P(AS|PS) = P(AS \mid PS, SINK) * P(SINK) + P(AS \mid PS, \neg SINK) * P(\neg SINK)$$

$$P(AS|\neg PS) = P(AS \mid \neg PS, SINK)P(SINK) + P(AS \mid \neg PS, \neg SINK)P(\neg SINK)$$

Now if the ship does not sink, the model prediction is irrelevant – it is reasonable to assume the passenger survives. So $P(AS \mid PS, \neg SINK)$ and $P(AS \mid \neg PS, \neg SINK)$ both are equal to 1. On the other hand when the ship does sink, the fact of sinking is baked into the model, so $P(AS \mid PS, SINK)$ is simply the probability of the model giving a true-positive prediction while $P(AS \mid \neg PS, SINK)$ is correspondingly the probability of a false-negative so we now have:

$$P(AS|PS) = P(\text{true-positive}) * P(SINK) + P(\neg SINK) \text{ – denote as } P_{tp} \quad (1)$$

$$P(AS|\neg PS) = P(\text{false-negative}) * P(SINK) + P(\neg SINK) \text{ – denote this as } P_{fn} \quad (2)$$

It is perhaps worth re-emphasising that these probabilities do not vary for any passenger the decision system offers advice to. They are functions of the model and the inherent risk of sinking. The only contribution of the passenger to the advice process is whether the model predicts survival or not.

[5] An exhaustive trawl though the records of Lloyds Registry would, no doubt, provide the answer, but that is a very long-term research project! See footnote 12 for a contemporary estimate.

[6] And in the middle of the night, in the North Atlantic, etc. etc.

Now we need to define our risk thresholds. For the moment we don't need explicit values but let us call these:

T_{go} – if the probability of survival is above this value, the advice will be "*Bon Voyage*"

T_{stay} – if the probability of survival is below this value, the advice will be "*Don't do it*!"

If the probability is between these the advice would obviously be "*Go at your own risk*".

We are now in a position to outline another solution which if far from perfect, does make use of the additional data we have. All we need to do is to select which probability to use (P_{tp} or P_{fn}), based on whether the model predicts survival or not, then apply the appropriate rules:

Solution 3 (outline):

Decision Service has following logic:

For passenger **with data** x **compute** $\mathcal{M}(x)$,

And apply the rules:

IF $\mathcal{M}(x) = 1$ and $P_{tp} >= T_{go}$
THEN advice = "Bon Voyage"

IF $\mathcal{M}(x) = 1$ and $P_{tp} < T_{go}$ and $P_{tp} >= T_{stay}$
THEN advice = "Go at your own risk"

IF $\mathcal{M}(x) = 1$ and $P_{tp} < T_{stay}$
THEN advice = "Don't do it!"

IF $\mathcal{M}(x) = 0$ and $P_{fn} >= T_{go}$
THEN advice = "Bon Voyage"

IF $\mathcal{M}(x) = 0$ and $P_{fn} < T_{go}$ and $P_{fn} >= T_{stay}$
THEN advice = "Go at your own risk"

IF $\mathcal{M}(x) = 0$ and $P_{fn} < T_{stay}$
THEN advice = "Don't do it!"

Total number of good advices for test cases ≈ ???

It looks like we have six rules here, but this is an illusion, the reality is there are only two. Because P_{tp}, P_{fn}, T_{go} and T_{stay} are all fixed, only one of the first three rules and one of the second three can ever fire. Indeed, if we (reasonably) assume that $P_{tp} > P_{fn}$

(which will be the case if the model is any good) and $T_{go} > T_{stay}$ (it would be nonsense for this not to be the case) it is possible we really only have one rule. This is the case if $P_{tp} < T_{stay}$ – when the solution reduces to solution 1 (never go), or $P_{fn} >= T_{go}$ when we get back to solution 2 (always go).

This gives us a problem. If we only have two rules, we can only give two pieces of advice, but the challenge asks for three. A moment's thought tells us why we have a problem. Our model only gives us a binary outcome. What we really want is not a yes/no answer from the model (a classification), but a probability of survival. We'll come back to this later. But now let's get onto the exciting bit – some Machine Learning.

4 <u>Some Predictive Models for Survival</u>

4.1 Preliminaries

If one spends a little time looking at the posts on the Kaggle site with regard to the Titanic ML competition (or indeed have spent any time looking into the way predictive analytic teams work), one will know the process of building a good model is very time consuming. The data set must be analysed in detail. Account must be taken of missing values, outliers, correlations between different attributes. Attributes which are categorical need different handling to those which are numeric. Then there are a vast range of different approaches to machine learning. Apart from a host of what one might term 'algorithmic' approaches such as nearest neighbours & support vector machines, one can also use deep learning mechanisms. This in mind, I felt I had three alternative options:

a) Spend weeks on doing it right
b) Directly lift a model from the Kaggle site[7]
c) Do the absolute minimum to get something which works better than nothing

I have effectively settled for c) with a bit of b) thrown in.

4.2 A Little Data Analysis - Which Columns to Use?

The data sets have eleven input columns. Some of these are clearly irrelevant – like the ticket number, passenger name and passenger id, so can be dropped at once. It seems unlikely that the Cabin number is of much use, and it is only available in 23% of the cases anyway. It seems hard to think that the port of embarkation would have influence on survival. So if we ignore these we only need to work with six input columns.

The 'Number of Siblings/Spouses' and 'Number of Parents/Children' columns present a few issues. Should we treat these as simple numeric fields? Is the difference between having 7 or 8 siblings aboard the same as having 1 or 2? A similar issue arises for 'Fare' with the additional complication that some of the fares are missing. Some Kaggle solutions suggests creating ranges here, such as travelling alone, travelling with one or two companions, travelling with more than two. A look at the data suggests travelling with no companions or more than two damages your survival chances.

[7] Or as Tom Lehrer put it in his immortal "Lobachevsky" song "Plagiarize, only be sure always to call it please 'research'"

Age does seem likely to be an important factor in survival. The very young and very old, one would imagine are less likely to survive[8], however we have a problem with the data set, namely that for 177 passengers in the training data we do not have this information (this is about 20% of training data, a similar portion of the test data is missing this information as well). What can we do about this? The simplest solution is probably to assume that if a passenger's age is not given, they will have an 'average' age. The mean age of the training set is about 30. One might try to get a better estimate by factoring in the sex of the passenger and things like if they have a spouse/sibling with them, but the standard deviation of ages is high (~14) compared to the mean, so it is unlikely to make big difference.

As to the two remaining fields, 'Passenger Sex' and 'Passenger Class' are categorical fields, which most machine learning algorithm implementations do not directly support. There are quick fixes for this, but as noted in some Kaggle submissions, it makes sense to consider 'Passenger Class' as an ordinal field as in some sense there is an ordering of the classes (first class > second class > third class), and this very clear in the mean survival rates (from the training set, 63% of first-class passengers survived but only 24% of third-class passengers[9])

4.3 Machine Learning Options

There are lots of approaches for building a model using machine learning. From the Kaggle submissions, the 'Random Forest' and 'Support Vector Machine' (SVM) approaches generally seem to be the ones which come out on top, so I decided initially to try these – but read on.

To implement the models, I have used the tried and tested Python libraries, which pop up in a lot of the Kaggle submissions, namely Pandas for the data wrangling and Scikit-Learn for the machine learning side of things. These make things seem deceptively easy. Once you have cleaned and tidied your data, you can build a model in literally only a couple of lines of code.

4.4 Model Evaluation

As noted above a 'perfect' model which accurately predicts who lives and who dies is not going to come out of the exercise. So how do we measure 'accuracy'? In the context of machine learning classification problems, '*accuracy*' has a very specific meaning, it is the proportion of cases that the model correctly classifies. For example, the simplest model is one that assumes everyone dies. This is the correct assumption 577 times out of 891 cases so has an accuracy of 577/891 = 0.648. A more sophisticated model assumes all the women survive, but all the men die. This model is correct 701 times out of 891, so has an accuracy of 0.787 (and it has an accuracy of 1.0 on the test cases!!). The overall accuracy is only one aspect of performance. If one category is rare compared to another, a model may have an accuracy close to 1, but still be of no use, as typically false-positives swamp the true-positives (this problem is prevalent in diagnostic tests for rare conditions). Fortunately, it is not a problem with the Titanic disaster, even though it was a major tragedy many people still survived. For our

[8] Or perhaps not. All 7 babies in the training set (passengers with a specified age under 1) survived, as did the oldest passenger

[9] Any data analyst worth their salt would point out there were considerably more third-class passengers, but numerically as well as proportionally more first-class passengers survived than third-class ones

purpose, the principal interest is in the true-positive, true-negative, false-positive, and false-negative rates which are given by something termed the ‘confusion matrix’[10].

4.5 First Model – Random Forest

The Random Forest algorithm is an evolution from the decision trees created by algorithms like ID3 and CART. It creates decision trees by selecting random data samples, the trees then ‘vote’ on what is the correct classification. This leads to shallower trees and less likelihood of overfitting at the cost of being computationally more expensive. My first attempt with this came up with the following results:

```
Using columns: "Age" "SorS" "PorC" "Fare" "Class" "Sex_female" "Sex_male"
For data set Training using model RandomForestClassifier(random_state=42)
Confusion Matrix for data set Training is:
                              Model Predicts
                             Die      Survive
                     Die     545         4    Neg Pred Value 0.993
 Actual outcome: Survive      12       330         Precision 0.965
                                                    Accuracy 0.982
                        Sensitivity  Specificity
                           0.978        0.988
```

This looks wonderful, 98% accuracy! But the test set results are disappointing:

```
For data set Test using model RandomForestClassifier(random_state=42)
Confusion Matrix for data set Test is:
                              Model Predicts
                             Die      Survive
                     Die     229        37    Neg Pred Value 0.861
 Actual outcome: Survive      36       116         Precision 0.763
                                                    Accuracy 0.825
                        Sensitivity  Specificity
                           0.864        0.758
```

Clearly there is overfitting. There are any number of things one can twiddle with to improve matters, but my first attempt at doing this gave a very creditable result. As hinted above I was dubious about the ‘Fare’, the ‘Number of Siblings/Spouses’ and the ‘Number of Parents/Children’ columns. What happens if you just remove them from the model? Well, the accuracy of the training set declines as one might expect:

```
Using columns: "Age" "Class" "Sex_female" "Sex_male"
For data set Training using model RandomForestClassifier(random_state=42)
Confusion Matrix for data set Training is:
                              Model Predicts
                             Die      Survive
                     Die     511        38    Neg Pred Value 0.931
 Actual outcome: Survive      69       273         Precision 0.798
                                                    Accuracy 0.880
                        Sensitivity  Specificity
                           0.881        0.878
```

But the accuracy of the test set jumps up to meet it:

[10] There are lots of descriptions of evaluation method for machine learning model available on the web, one I liked is https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/

```
Using columns: "Age" "Class" "Sex_female" "Sex_male"
For data set Test using model RandomForestClassifier(random_state=42)
Confusion Matrix for data set Test is:
                              Model Predicts
                             Die      Survive
                     Die     236         30   Neg Pred Value 0.887
 Actual outcome: Survive      23        129        Precision 0.849
                                                    Accuracy 0.873
                         Sensitivity  Specificity
                            0.911        0.811
```

The accuracy of the model working on the test set is 87% vs 88% on the training set. Given we know there is a going to be a huge number of random factors in play which are not in the data set but influenced the individual survival prospects of each passenger, the close match suggests our model is capturing most of the predictive power in the data. We are predicting with close to 90% accuracy with only three pieces of data. We might be able to do better than this, but not much, I think.

4.6 Some other models – SVM, GaussianNB, DecisionTreeClassifier and GradientBoostingClassifier

Having packaged up the code to create and evaluate a model I went to town and tried several more[11] – but generally didn't have a lot of luck in improving on my first attempt!

This is not so much an issue with the algorithms, as a lack of effort on my part with the data analysis. In particular some feature engineering on the various attributes would have helped as is abundantly clear from many of the submissions on the Kaggle site.

4.6.1 Support Vector Machine (SVM)

This did not go well. The default version only gives an accuracy of 70% scarcely better than assuming everyone dies. I tried adding back in the 'Fare', the 'Number of Siblings/Spouses' and the 'Number of Parents/Children' columns. This makes the performance even worse:

```
Using columns: "Age" "Class" "Sex_female" "Sex_male"
For data set Training using model SVC(random_state=42)
Confusion Matrix for data set Training is:
                              Model Predicts
                             Die      Survive
                     Die     526         23   Neg Pred Value 0.958
 Actual outcome: Survive     239        103        Precision 0.301
                                                    Accuracy 0.706
                         Sensitivity  Specificity
                            0.688        0.817
```

[11] Understanding how these algorithms work is left as an exercise to reader

```
Using columns: "Age" "Class" "Sex_female" "Sex_male"
For data set Test using model SVC(random_state=42)
Confusion Matrix for data set Test is:
                                Model Predicts
                               Die      Survive
                      Die      256         10   Neg Pred Value 0.962
 Actual outcome: Survive      114         38         Precision 0.250
                                                      Accuracy 0.703
                          Sensitivity  Specificity
                             0.692        0.792
```

Modifying the out of the box version of the model to use a linear kernel boosts the accuracy up to 78.7% – but something strange happens with the test data!

```
Using columns: "Age" "Class" "Sex_female" "Sex_male"
For data set Training using model SVC(kernel='linear', random_state=42)
Confusion Matrix for data set Training is:
                                Model Predicts
                               Die      Survive
                      Die      468         81   Neg Pred Value 0.852
 Actual outcome: Survive      109        233         Precision 0.681
                                                      Accuracy 0.787
                          Sensitivity  Specificity
                             0.811        0.742
```

```
For data set Test using model SVC(kernel='linear', random_state=42)
Confusion Matrix for data set Test is:
                                Model Predicts
                               Die      Survive
                      Die      266          0   Neg Pred Value 1.000
 Actual outcome: Survive        0        152         Precision 1.000
                                                      Accuracy 1.000
                          Sensitivity  Specificity
                             1.000        1.000
```

The test data is predicted perfectly!! What is going on here? Well as noted at the beginning (see footnote 1), the test data set is very odd: all the women survive; all the men die. And the accuracy of the model on the training data is the same as the naïve model described in section 4.4, which predicts survival on the basis of sex. It looks as if the model is basing its decision purely on whether the passenger is a man or a woman.

4.6.2 Gaussian Naïve Bayes

Although the logic behind the training algorithm is different this gives essentially identical results to the SVN model:

```
Using columns: "Age" "Class" "Sex_female" "Sex_male"
For data set Training using model GaussianNB()
Confusion Matrix for data set Training is:
                                Model Predicts
                               Die      Survive
                      Die      468         81   Neg Pred Value 0.852
 Actual outcome: Survive      109        233         Precision 0.681
                                                      Accuracy 0.787
                          Sensitivity  Specificity
                             0.811        0.742
```

```
Using columns: "Age" "Class" "Sex_female" "Sex_male"
For data set Test using model GaussianNB()
Confusion Matrix for data set Test is:
                              Model Predicts
                             Die      Survive
                     Die     266         0   Neg Pred Value 1.000
 Actual outcome: Survive       0       152        Precision 1.000
                                                   Accuracy 1.000
                         Sensitivity  Specificity
                            1.000        1.000
```

Once more the model prediction seems simply based on sex. Adding back in the ‘Fare’, the ‘Number of Siblings/Spouses’ and the ‘Number of Parents/Children’ columns gives a very small boost to the performance.

4.6.3 Decision Tree

This algorithm is based on the CART algorithm for generating decision trees, and so is the flip-side of the Random Tree algorithm, fast but with deep trees and vulnerable to overfitting. However, since the data set is relatively small and the number of input columns being used is very small, it is not surprising both algorithms come up with models with almost identical results:

```
Using columns: "Age" "Class" "Sex_female" "Sex_male"
For data set Training using model DecisionTreeClassifier(random_state=42)
Confusion Matrix for data set Training is:
                              Model Predicts
                             Die      Survive
                     Die     518        31   Neg Pred Value 0.944
 Actual outcome: Survive      76       266        Precision 0.778
                                                   Accuracy 0.880
                         Sensitivity  Specificity
                            0.872        0.896
```

```
Using columns: "Age" "Class" "Sex_female" "Sex_male"
For data set Test using model DecisionTreeClassifier(random_state=42)
Confusion Matrix for data set Test is:
                              Model Predicts
                             Die      Survive
                     Die     237        29   Neg Pred Value 0.891
 Actual outcome: Survive      29       123        Precision 0.809
                                                   Accuracy 0.861
                         Sensitivity  Specificity
                            0.891        0.809
```

4.6.4 And the winner is … Gradient Boosting!

I must admit this is a new one on me, which I found while browsing the Kaggle site. The algorithm is based around ‘ensemble’ learning which is one of the driving concepts of the random tree algorithm. The results on the training data are good at 85% accuracy though not quite as good as the tree algorithms. That the model performs better on the test data than the training data is presumably down to the test data’s peculiar nature.

```
Using columns: "Age" "Class" "Sex_female" "Sex_male"
For data set Training using model
GradientBoostingClassifier(random_state=42)
Confusion Matrix for data set Training is:
                              Model Predicts
                             Die      Survive
                     Die     496         53   Neg Pred Value 0.903
 Actual outcome: Survive      76        266        Precision 0.778
                                                    Accuracy 0.855
                        Sensitivity  Specificity
                           0.867        0.834
```

```
Using columns: "Age" "Class" "Sex_female" "Sex_male"
For data set Test using model GradientBoostingClassifier(random_state=42)
Confusion Matrix for data set Test is:
                              Model Predicts
                             Die      Survive
                     Die     246         20   Neg Pred Value 0.925
 Actual outcome: Survive      11        141        Precision 0.928
                                                    Accuracy 0.926
                        Sensitivity  Specificity
                           0.957        0.876
```

However, adding back in the 'Fare', the 'Number of Siblings/Spouses' and the 'Number of Parents/Children' columns with this algorithm does actually give a boost to the performance putting it ahead of the two tree algorithms adding percentage points to the accuracy on both training and test data sets:

```
Using columns: "Age" "SorS" "PorC" "Fare" "Class" "Sex_female" "Sex_male"
For data set Training using model
GradientBoostingClassifier(random_state=42)
Confusion Matrix for data set Training is:
                              Model Predicts
                             Die      Survive
                     Die     526         23   Neg Pred Value 0.958
 Actual outcome: Survive      61        281        Precision 0.822
                                                    Accuracy 0.906
                        Sensitivity  Specificity
                           0.896        0.924
```

```
Using columns: "Age" "SorS" "PorC" "Fare" "Class" "Sex_female" "Sex_male"
For data set Test using model GradientBoostingClassifier(random_state=42)
Confusion Matrix for data set Test is:
                              Model Predicts
                             Die      Survive
                     Die     249         17   Neg Pred Value 0.936
 Actual outcome: Survive      28        124        Precision 0.816
                                                    Accuracy 0.892
                        Sensitivity  Specificity
                           0.899        0.879
```

5 'Go at your own risk' – the Probability of Survival

When building 'Solution 3', one of the problems inherent in the way the challenge was posed became evident. If we have a model which simply predicts if a passenger survives, we can only ever give two pieces of advice, never three. However, with some of the models rather than ask it to classify a case, we can ask what does the model estimate the probability of a case to be? In other words, the models we build can say:

"the estimated probability of this passenger surviving is 0.7"; rather than "this passenger is predicted to survive". So instead of the model $\mathcal{M}$ proposed in section 3, we could have a model $\mathcal{M}'$ which instead of giving a prediction 1 or 0 about survival gives a probability *p* about survival. To take some examples. The Gradient Boosting model predicts that passenger 24 in the training set - Mr. William Thompson Sloper – will survive (he did). However, if we ask the model what the probability of him surviving is, we find it is only 0.534, so the model suggests he was lucky to do so. The model predicted that fellow survivor passenger 349 - Master William Loch Coutts was almost certain to survive (with probability 0.971).

Using probabilities rather than predictions means the analysis in section 3 gets muddled up. We no longer have false-positives and true-positives to work with, but a probability distribution, and we know that the model is not perfect, so the probability distribution is at best an approximation to the true one. But given the model accuracy is high we can at hope that it is a reasonable approximation. To make use of the probability instead of the prediction, we replace equations (1) and (2) by the following equation for the total probability of survival allowing for the fact the ship may not sink:

$$P(AS) = P(\text{survival from model}) * P(\text{SINK}) + P(\neg\text{SINK}) \text{ – denote as } P_{as} \qquad (3)$$

We now get a fourth solution which can actually give three pieces of advice (if we select suitable values for the other parameters):

Solution 4 (outline):

Decision Service has following logic:

```
For passenger with data $x$ compute $P_{as} = \mathcal{M}'(x) * P(\text{SINK}) + P(\neg\text{SINK})$

And apply the rules:

IF $P_{as} >= T_{go}$
THEN advice = "Bon Voyage"

IF $P_{as} < T_{go}$ and $P_{as} >= T_{stay}$
THEN advice = "Go at your own risk"

IF $P_{as} < T_{stay}$
THEN advice = "Don't do it!"

Total number of good advices for test cases ≈ ???
```

We don't have an answer for how much good advice we give, but the problem is the test cases *cannot* be used to give us one. They tell us which passengers survived, but it is on the basis that the ship they were on sank. Our model creation/validation process can use them because the Titanic *did* sink, but our decision system gives advice to people thinking of taking a ship which only *might* sink. And the advice we give depends on the thresholds we set as well as the model predictions. If we are risk averse, we may opt not travel even if the risk of dying is fairly low (say no more than one in ten).

6 Parameterising our Imperfect Solutions

Let's put in some numbers. We will consider three values for the probability of the ship sinking – (1.0, 0.5, and 0.9) – that is to say the ship always sinks, there is a fifty-fifty chance of it sinking or that nine times out of ten it completes its trip safely[12]. We will make use of the Gradient Boosting model, which was the best we found in section 4. This allows us to calculate P_{tp} and P_{fn} for each value:

P(SINK)	P(true-positive)	P(false-positive)	P_{tp}	P_{fn}
1.0	281/(281+61) = 0.822	23/(23+526) =0.042	0.822	0.042
0.5	281/(281+23) = 0.822	23/(23+526) =0.042	0.911	0.521
0.9	281/(281+23) = 0.822	23/(23+526) =0.042	0.982	0.904

Note that if the ship is certain to sink, P_{tp} and P_{fn} are simply the model predictions.

Our next challenge is to pick some values for T_{go} and T_{stay}. How about we say if we always sail if our chances of survival are over 95% (nineteen times out of twenty), and never sail if they are below 83% (five times out of six), we get the following alternative versions of solution 3:

```
Solution 3 (with P(SINK=1.0):
Only rule is:
IF true
THEN advice = "Don't do it!"              # as $P_{tp}$ (0.82) < $T_{stay}$ (0.83)

Solution 3 (with P(SINK=0.5):
For passenger with data $x$ compute $\mathcal{M}(x)$,
Then apply rules
IF $\mathcal{M}(x) = 1$
THEN advice = "Go at your own risk"   # as $T_{stay}$ (0.83) < $P_{tp}$ (0.91) < $T_{go}$ (0.95)

IF $\mathcal{M}(x) = 0$
THEN advice = "Don't do it!"              # as $P_{fn}$ (0.52) < $T_{stay}$ (0.83)

Solution 3 (with P(SINK=0.1):
For passenger with data $x$ compute $\mathcal{M}(x)$,
Then apply rules
IF $\mathcal{M}(x) = 1$
THEN advice = "Bon Voyage"                # as $P_{tp}$ (0.98) > $T_{go}$ (0.95)

IF $\mathcal{M}(x) = 0$
THEN advice = "Go at your own risk"   # as $T_{stay}$ (0.83) <= $P_{fn}$(0.90) < $T_{go}$(0.95)
```

[12] A fairly recent article in the Guardian https://www.theguardian.com/world/2015/jan/10/shipping-disasters-we-never-hear-about suggests perhaps 24 'large' ships out of about 86,000 worldwide will sink in a year – so assuming each makes say 5 trips a year, we end up with a probability of 0.00005 of a ship sinking on a given voyage. Even if things were a thousand times worse a century ago, a 0.1 chance of a sinking is still a little pessimistic

As noted in section 3, we get a maximum of two rules and only two types of advice. It is interesting that even if there is a fifty-fifty chance of sinking, some brave souls may be prepared to risk it.

For Solution 4, we get the more concrete implementation:

Solution 4:

Decision Service has following logic:

For passenger **with data** x **compute** $P_{as} = \mathcal{M}(x) * P(\text{SINK}) + P(\neg\text{SINK})$

And apply the rules:

IF $P_{as} >= 0.95$
THEN advice = "Bon Voyage"

IF $P_{as} < 0.95$ and $P_{as} >= 0.83$
THEN advice = "Go at your own risk"

IF $P_{as} < 0.83$
THEN advice = "Don't do it!"

But with this solution the outcome depends on the individual passenger, not how the model classifies them. Let us consider Mr Sloper and Master Coutts again, and also introduce Passenger 631 - Mr. Algernon Henry Wilson Barkworth - who at 80 is the oldest person in either the test cases or the training cases and also was a survivor. The model gives him a probability of surviving of 0.838. If we assume the probability of sinking is 0.1, using Solution 3, we see that since all three passengers are predicted to survive by the model the service will both offer them all the same advice – "Bon Voyage". If the probability sinking is 0.5 the advice, they all get is "Go at your own risk", and if it is 1.0 it is "Don't do it!"

On the other hand, if we look at solution 4, we find the advice for the passengers differs depending on the probability of sinking:

Passenger	Estimated probability of survival on sinking	Survival probability for P(SINK) = 1.0	Survival probability for P(SINK) = 0.5	Survival probability for P(SINK) = 0.1
Sloper	0.534	0.534	0.767	0.953
Barkworth	0.838	0.838	0.919	0.984
Coutts	0.971	0.971	0.986	0.997

Except when the probability of sinking is low (when they all are recommended to go), each passenger gets different advice. Sloper's value of P_{as} is below 83% so he is advised not to go. Barkworth's value of P_{as} is between the thresholds so is advised to go at his own risk, Young Master Coutts' value of P_{as} is so high he is always recommended to go.

7 Last Thoughts

I have mixed feelings about the challenge. It does make it clear there is a big difference between a decision service which incorporates a machine learning model and the model itself. For the challenge as posed, the model is subordinate to something we have no good information about, how likely a ship is to sink. Furthermore, such estimates as we do have, suggest the probability is so low it is likely we will always advise passengers to travel.

Working on the challenge also makes it clear that the availably of libraries like Scikit-Learn make machine learning seem so easy that people will rush off and do ‘Machine Learning’ without really understanding what they are doing and the limitations and real challenges to be overcome in doing it properly.

The main conclusion is I need to learn more and try a bit harder.

Appendix – Source Code

Source code is in the form of a Jupyter notebook. I still haven’t got around to setting something on GitHub, but I will e-mail it on request.