Loyola COMP 479
Nate Boldt
Sept 14, 2021

Report on Machine Learning Algorithm Attempt

My implementation of a "machine learning" algorithm to predict Titanic survivors was mildly effective, showing a 68% accuracy of predicting a group of survivors. This accuracy number comes from the Titanic challenge submission process. The approach I created involved reverse engineering which features and values within those features occurred often among the survivors in the training set.

In more detail, the approach starts by finding which features might be important to surviving by counting how many survivors have that value. In other words, we can group the survivors into different values for each feature. For example, if 80% of the survivors had an "Embarked" value of "C", this should indicate that it is an important aspect to survival. Therefore, we can rank each value by what percentage of survivors had it. Then, we can add up the "points" for each survivor by summing their percentage values for each feature.

Once we have a "points" value for each survivor, we can start making decisions on who survivors. One way to do this is to take the percentage of the training set who did survive. Assuming our training set is sufficiently large, we can then apply the same percentage to the test set on which passengers could survive. Therefore, we make a cutting index at the percentage (~38%) in this context where survivors who have more points than the cut survive and those with fewer points do not survive. This would only work when using the algorithm on a test set – if we wanted to predict each case individually, we could simply take the points value at the cutting index instead and apply it in each case.

This approach would not be my first choice in an actual application, but it does perform better than 50% in this example. The benefits of this algorithm include unbiased rankings of features across the board – instead of honing in on a few more obvious traits like "Sex" or "Age", it takes a wholistic view. It also only creates weighted values based on actual data as opposed to the biases of the code writer.

Other possibilities that I considered included grouping the values into categories that might promote better rankings, for example the "Age" attribute. In this current implementation, we use the "Age" feature as individual ages whereas it might make more sense to group survivors based on decades like "30's or 40's". Additionally, features like "number of siblings" might really only matter if the survivor in question is not an adult, so that might be another area for increased accuracy.

Overall, this approach helped me think more critically about defining and ranking features, as opposed to just picking the top X# of features and basing the entire ranking around them. Additionally, this implementation might be useful to look at before starting another project as it helps the user see which traits are actually potentially meaningful – in which traits have high percentages of survival for a certain value. Nevertheless, this simplistic reverse engineering from existing data provided a decent result and several avenues for improvement upon itself.