Survivor Group Project

Zack Ballam, Chase Bledsoe, Ryan Lee, Matthew Sheppard
CS 201R, Winter 2024
Department of Computer Science
Brigham Young University

Abstract

As reality TV shows become more complex and dynamic, the task of predicting their outcomes becomes more difficult. This project attempts to create a model to predict which contestant would be voted off of each episode of Survivor utilizing data from all 45 seasons of the show. By meticulous data processing and many attempts at feature engineering, we have gained new insights on what features more greatly impact the outcomes of Survivor. However, we were not able to produce an accurate model. The results of this knowledge can be used to create more accurate models in the future, and it can help the producers and directors of Survivor gain and retain viewers throughout the entirety of each new season.

1. Introduction

Survivor is a reality TV show that has been airing since the early 2000s. Today there have been 45 seasons. Each season takes a group of people who are tasked with working together, creating alliances, and completing challenges. At the end of each episode, one person is voted off, therefore eliminating them from the show. Our task was to train a model to predict the winner of each season. This first approach came with some serious problems, as our dataset would have been severely limited with only 45 instances of winners and hundreds of instances of losers. We decided to change the focus of our project to instead predict who would be voted off the show at the end of each episode. This helped us to have a more balanced training set (more on this in Sections 2a and 3f).

We implemented data found from a [1] Survivor dataset that contained data on each contestant–or each survivor as they're known in the show–for each of the 45 seasons. The data included demographic information (race, age, personality type, ethnicity, etc), information about which survivors won challenges, the number of times a survivor appeared alone before the camera (known as a confessional), and who was voted out in which episode. The data was easily accessible, but we needed to do a lot of feature engineering to organize the data in a way that would produce our desired result. During the process, we ran into a few problems, mainly handling the imbalance of our dataset and creating new and viable features to train on.

2. Methods

a. Sampling Techniques

As our team looked over our problem and dataset, we quickly recognized that we had a huge gap between the number of samples for our two classes (those who were voted out versus those who stayed each episode). This is because each Survivor season begins with at least 15 contestants, and usually only one is voted out per episode. So, for each episode (n), there are n-1 instances of contestants staying and 1 instance of a contestant leaving. To remedy this issue, we explored oversampling and undersampling techniques.

The oversampling technique our team applied is SMOTE (Synthetic Minority Oversampling Technique). Through SMOTE, the minority class of the dataset is oversampled until it reaches an equivalent amount as the other class. Because our class ratio was approximately 12,000 to 1,000 (for staying on versus voted out respectively), SMOTE oversampled our voted-out class randomly until it had reached 12,000 samples, giving us 24,000 samples to train on overall. To properly attain and apply this oversampled data, we first split the regular data into a training set and a test set. We then applied SMOTE on the training set exclusively so our model's test results would be representative of a real application environment.

Our team also investigated a random undersampling technique, which would randomly sample the majority class (staying on in this case) until it had reached an equivalent number of samples to the minority class (voted-out). However, this only yielded 1,000 samples of each class, with a total dataset of 2,000 samples. Our team felt this was not enough data to train on, and decided to pursue the SMOTE technique instead. However, exploring the use of this sampling method for our problem could be highly beneficial in the future.

b. Feature Engineering

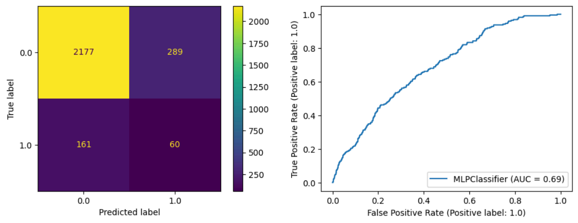
As mentioned in the introduction, we encountered difficulties in organizing our data. The dataset we used was separated into many different relational tables. To organize the data we began pulling information from all of the tables and organizing them by each episode. This meant we had rows of data that showed information on each survivor in each episode. We hypothesized that as the episodes got further into the season we could accumulate some of the data from the other episodes. We wrote a function that would create a new feature with cumulative data for that survivor from all of the prior episodes. This helped us to more correctly predict who would be voted off in later episodes. We also added information about whether each survivor was in their original tribe or not because we knew this to be an important factor in the show (from the alliances that are made in original tribes early on). However, we soon realized this only marginally helped us. We then created a new feature that kept track of the percentage of survivors that remained from their original tribe. This feature ended up being much more influential on the overall prediction. Finally, we added a feature that would tell us what percentage of the remaining survivors shared the same gender. This feature also ended up being very effective.

3. Results

Note: The following models were trained using an oversampled training set and tested using the original, unmodified dataset (as discussed in Section 2 SMOTE).

a. Multi-layer Perceptron

Our team began our training by using an MLP and discovered that it was most accurate with a single hidden layer of 64 nodes, an initial learning rate of 0.01, and a momentum of 0.9.

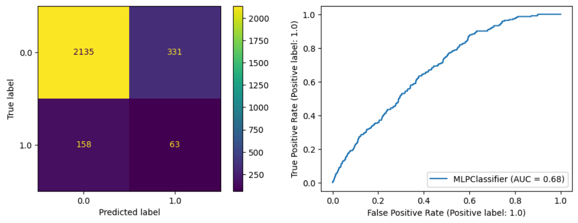


This model received the following F1 scores:

```
F1 Score: [0.90632806 0.21052632]
F1 Macro Score: 0.5584271878697576
F1 Micro Score: 0.8325269817640492
F1 Weighted Score: 0.8490998554619562
```

b. Grid Search Multi-layer Perceptron

We found the most optimal hyper-parameters for the above MLP through a few different grid searches. Here is one such instance below where we found the best learning rate (of 0.01) and the best momentum (of 0.9).

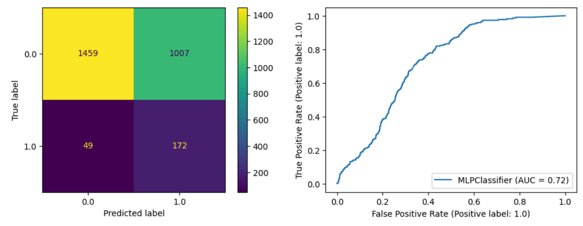


This model received the following F1 scores:

```
F1 Score: [0.89724732 0.20487805]
F1 Macro Score: 0.5510626848231079
F1 Micro Score: 0.8180126535169334
F1 Weighted Score: 0.8403014298605782
Number of iterations: 117
```

c. (Super) Multi-layer Perceptron

As a final approach to tuning an MLP, our team set the model to run until 1000 max iterations (provided the model was still improving in accuracy). Surprisingly this model converged in fewer iterations than we expected (about 200-300) and correctly predicted more true negatives, however, the accuracy of true positives decreased significantly.



This model received the following F1 scores:

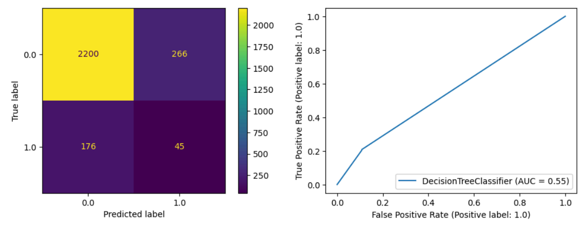
```
F1 Score: [0.73427277 0.24571429]
F1 Macro Score: 0.48999352936947305
F1 Micro Score: 0.6069966505396353
F1 Weighted Score: 0.6940898829258166
Number of iterations: 52
```

d. Decision Tree

We wanted to test out a Decision Tree to see how well it would perform, and to see if we could get any insights into which parameters had the most impact. We trained at several max-depths, as well as letting the model train as far as it needed.

i. Default Parameters

The default parameters resulted in a tree with a max-depth of 26. We ended up with scores similar to our best MLP model.

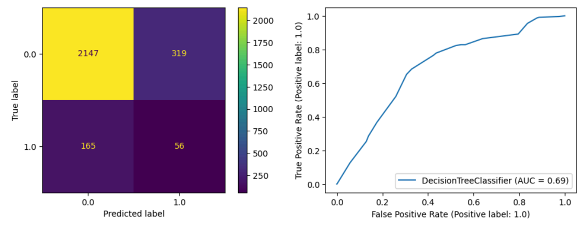


This model received the following F1 scores:

```
F1 Score: [0.90871541 0.16917293]
F1 Macro Score: 0.5389441695937489
F1 Micro Score: 0.8355042798660216
F1 Weighted Score: 0.8478896208982741
```

ii. Max Depth of 7

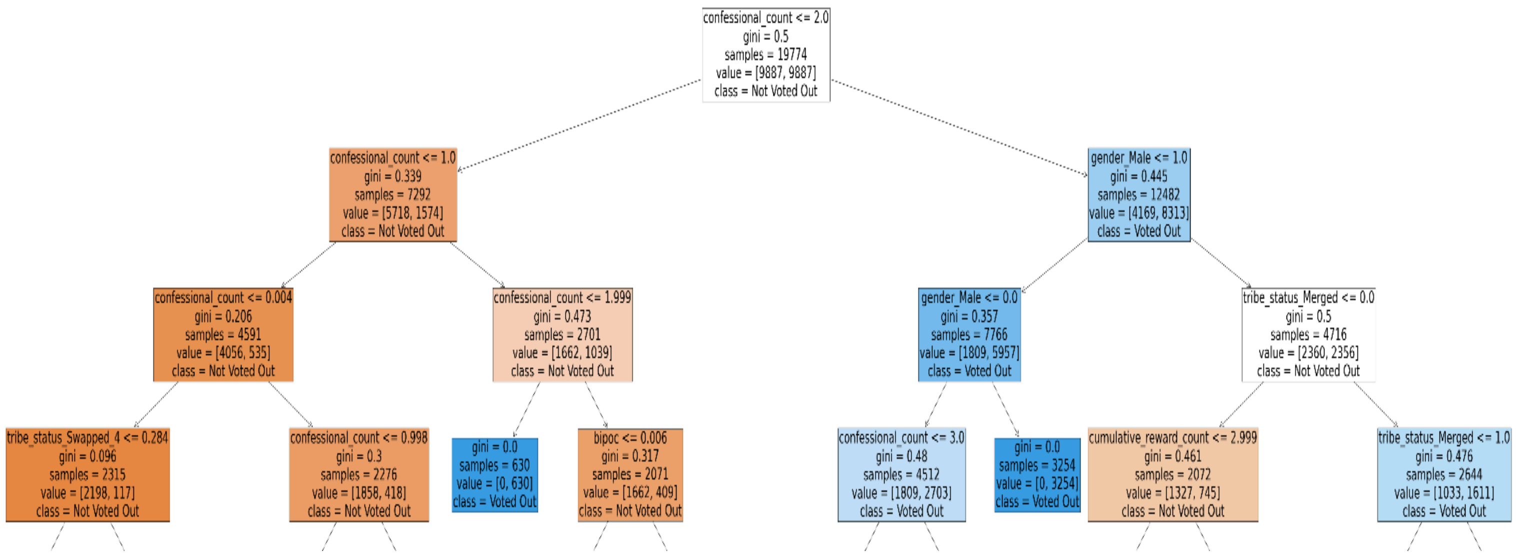
A max-depth of seven provided similar results to the default parameters. There were slightly more false positives and slightly less false negatives. The top layers for the trees look very similar, and we were able to find some interesting incites that will be gone over in the discussion section.



This model received the following F1 scores:

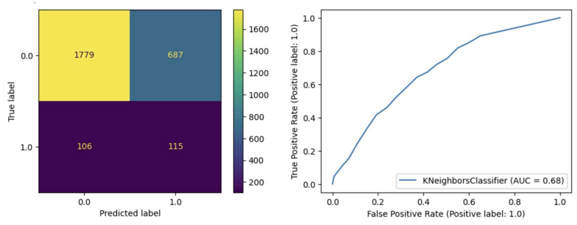
```
F1 Score: [0.89870239 0.18791946]
F1 Macro Score: 0.5433109245113932
F1 Micro Score: 0.8198734648306661
F1 Weighted Score: 0.8402420115591064
```

The first 3 layers of the decision tree

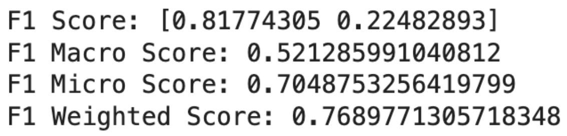


e. K-Nearest Neighbors

We decided to test it on a K-Nearest Neighbors model to see if it performed any better. After running it with different P values and different numbers of nearest neighbors we settled on a P value of 1 and 17 nearest neighbors.



This model received the following F1 scores:



```
F1 Score: [0.81774305 0.22482893]
F1 Macro Score: 0.521285991040812
F1 Micro Score: 0.7048753256419799
F1 Weighted Score: 0.7689771305718348
```

f. SMOTE

As shown in the model results in this section, every model struggled to reach high and consistent classification accuracies. Although most models had a high training set accuracy (above 95%) and an average test set accuracy (around 80%), our team found that these numbers did not tell the whole story. We believe the high training set accuracy came as a result of overfitting the training data set, which is undesirable. We then found that our models were achieving their test accuracy by guessing a disproportionate amount of the staying-on class, which had significantly more natural samples. All models had a high false-positive and false-negative rate, with a small minority of correctly classified voted-out instances.

4. Discussion

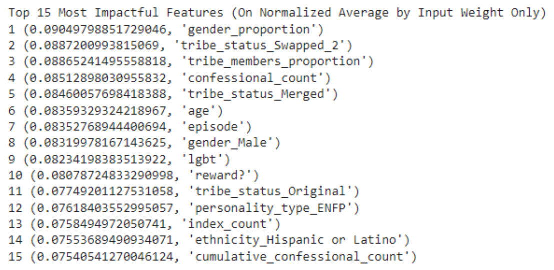
a. Multi-layer Perceptron

Unfortunately, the original issue of having an unbalanced dataset continued to have a huge impact on our model, despite our attempts at utilizing the SMOTE technique to train. As denoted by the F1 scores, the model performs well at classifying the staying-on class, which makes up most of the test set instances. This provides the bulk of the test set accuracy score. However, a look at the associated confusion matrix shows that only a small minority (about 25%) of the voted-out instances were classified correctly.

b. MLP Weights

After training this MLP model, our team wanted to observe which features in the dataset were most influential (on average) in the model. To accomplish this, we decided to only look at the weights connecting the input features to the hidden layer of nodes (for simplicity). Our team normalized all of these weights collectively. We then took the absolute value and mean of all the weights connected to an input feature for each input feature. Finally, we associated these weights with their counterpart labels from the dataset and sorted them in descending order. *Note: Our team recognizes that this approach is unusual, and only gives an estimate into the way this model is interpreting the input features.*

By conducting this analysis, our team found that the input features relating to gender and tribe proportion were very influential, which impressed our team. Initially, we had believed the number of interviews (confessional_count) would be the determining feature for our models. Although this feature was very impactful, we found that other features had equal or greater influence on this model's classification process.



```
Top 15 Most Impactful Features (On Normalized Average by Input Weight Only)
1 (0.09049798851729046, 'gender_proportion')
2 (0.0887200993815069, 'tribe_status_Swapped_2')
3 (0.08865241495558818, 'tribe_members_proportion')
4 (0.08512898030955832, 'confessional_count')
5 (0.08460057698418388, 'tribe_status_Merged')
6 (0.08359329324218967, 'age')
7 (0.08352768944400694, 'episode')
8 (0.08319978167143625, 'gender_Male')
9 (0.08234198383513922, 'lgbt')
10 (0.08078724833290998, 'reward?')
11 (0.07749201127531058, 'tribe_status_Original')
12 (0.07618403552995057, 'personality_type_ENFP')
13 (0.0758494972050741, 'index_count')
14 (0.07553689490934071, 'ethnicity_Hispanic or Latino')
15 (0.07540541270046124, 'cumulative_confessional_count')
```

We found it interesting that the proportional features that we made through our feature engineering ended up being very impactful within the model (Note: adding more of these could increase the accuracy of the model). It was also very interesting to us that gender made such an impact on whether or not a survivor was voted off. At first we thought that our prediction was wrong. However, after some research we found that about 61% of all Survivor winners are male which gives credibility to the interpretation of feature weights listed above.

c. Grid Search Multi-layer Perceptron

To find the best parameters we ran a few different grid searches. Because running grid searches is highly time intensive, we weren't able to try as many combinations as we would have liked, but we were able to find some important details. First, we created a custom scorer that predicted the best model based on how many class 1 instances were correctly identified (that is for the voted-out class). This helped us to find the best parameters for predicting who would be voted off each episode and not just who would stay. We also noticed that making more complex hidden layers only helped to some extent and that making the model more complex didn't necessarily help our model become more accurate. However, the grid search helped us to find the best momentum and learning rates for our model.

d. (Super) Multi-layer Perceptron

In addition to the default MLP and Grid Search MLP, our team wanted to ensure our models were not getting stuck in a local minimum in the training process. Interestingly, this model predicted class 1 (the voted-out class) much more frequently than the other MLPs. Unfortunately, this increased prediction rate did not come with increased accuracy. This served as another clue to our team that more work needed to be conducted in massaging our dataset, as the models up to this point were not able to make accurate classifications. We noticed that the reason this model was able to predict more class 1 instances (the voted-out class) was because it was classifying more instances as 1s. This increased the number of correctly identified 1s, but it also drastically increased the number of incorrectly identified 1s.

e. Decision Tree

Training a decision tree performed similarly to our best MLP model. However, this meant the decision tree also had the same difficulty accurately predicting the voted out class. Printing out the decision tree gave us some interesting insights into which features were most important. The root node for the tree, and several nodes in the first three layers are based on confessional_count. Other features that were considered in the first three layers include the following: gender_male, tribe_status_merged, cumulative_reward_count, and tribe_status_swapped. This group of features appears to align with what the players generally consider when voting.

f. K-Nearest Neighbors

While the K-Nearest Neighbors model didn't prove to be better than many of our other models, we did notice that we had a similar result as the (Super) Multi-layer Perceptron. We noticed that the number of survivors that were correctly identified as being voted off was higher. However, the number of survivors that were incorrectly identified as being voted off was drastically higher as well. We hypothesize that this is because our data wasn't very separable in the format we had organized it. This also occurred because this model predicted that survivors would be voted off more often than the other models.

g. SMOTE

Overall, our models were not able to reach any strong accuracies in the time frame we had to work on this project. We believe this is a product of our dataset. Although SMOTE can work in some situations, we do not believe we have enough data of voted-out contestants to allow SMOTE to be an effective process in this setting. Oversampling the small voted-out class twelve times over created a huge amount of repeated data, which we believe was not helpful for our models. It is also possible that there are other issues with our dataset that would need more time to investigate. For example, a deeper exploration of which features are contributing, if we have enough data for each of those features, and how intermeshed the two classes of data are could yield greater insight if we were to pursue this problem further.

5. Conclusion

In conclusion, our team set out to build an ML model to reliably predict which contestant would be voted out in any given episode and season of Survivor, given all of the data on the contestants up to that point. Our team took multiple different approaches to transform and modify the data our model would use, including generating new features from the data set, joining in features from various different dataframes, and applying the SMOTE oversampling technique to have an equivalent number of instances of voted-out and not voted-out (stayed-in) samples. We then tested several different model architectures to see which type would yield the highest classification accuracy. Unfortunately, the modifications we made in the dataset were not sufficient to enable any model architectures to find clear classification patterns. This may signal a need for more diverse input features, or more samples overall to counteract the lack of equality in classification instances in the initial dataset. With more time and data, our team expects better classification accuracies to be possible.

This project would be valuable for the directors and editors of Survivor because the model would tell them what types of Survivors are more likely to do well. This could help the show's directors and editors to choose contestants that would perform better, thus making the show more interesting. This project also helps us predict who will be voted off by how many times a Survivor appears each episode. This knowledge can help the directors edit each episode in order to prevent viewers from easily predicting who will be voted off each episode. Both of these facts will help Survivor gain viewers and keep them until the end of the season. If the contestants weren't interesting nobody would be interested in watching, and if it was too easy to predict the winner (or who was voted off each episode) viewers wouldn't continue watching throughout the whole season.

6. Future Work

There is much to be said of the future work that could be done in this field, and as our team developed our models and worked with the dataset, we discovered several different approaches that, if implemented, could improve the accuracy of this project. The main issue that we encountered is that by nature, the problem of Survivor prediction is a comparison between players and most of our data deals with aspects of a single player. As we tried to fine-tune our original approach we were able to better recognize the complexities of this problem and also some potential solutions which will be discussed in this section.

The first potential method that could have a positive impact on the outcome of the results is a manipulation of the data. As previously discussed, our team added features to the data that was a proportional value of tribe members and gender, rather than just using the feature of tribe or gender on its own, and those two features quickly became some of the most helpful in the model predictions. Thus, an area of future work in this project would be to convert more or the features into proportions so that the model can take into account the relationship between the shared attributes among players rather than just the attributes of a single player.

The second potential method that could be used in future work to improve results would be a complete change in the structure of the project to use regression instead of classification. Instead of classifying whether or not a person would be voted off, instead the model could assign a float value of the chance of a person being voted out. Then, those values could be compared and the person with the highest value assigned by the model would be classified as voted out.

In conclusion, there are several approaches that could potentially solve the problems that our team faced, and there is much more that could be explored to improve classification accuracy in the future.

7. References

[1] doehm. (n.d.). survivoR. GitHub. https://github.com/doehm/survivoR