Using Multinomial Naive Bayes to Predict the Outcome of a *Slay The Spire* Run

Kent Torell
University of North Carolina at Chapel Hill

November 30, 2018

1 Introduction

Slay The Spire is a deck-building, rogue-like, turn-based role playing game that has been gaining popularity since it was first released as an early access game in November of 2017. The game is fairly difficult, with only about 15-20% of all runs achieving victory, and, while a large portion of the success chance comes from tight gameplay, an even larger portion comes down to luck and deck composition. This project focuses on predicting the outcome of the final fight of each run (which at the inception of the project was the Act 3 boss) based solely on the composition of the deck going into the fight.

2 Data

With permission from MegaCrit, the developers of *Slay The Spire*, I was able to obtain run data for approximately 188,000 runs. From these runs I only included runs that reached the boss of Act 3. This gave about 30,000 runs with which to train and test a model. While there is no limit to the number of cards that can go into a deck, most runs usually have between 20 and 40 cards in them. The data that was provided for this project had runs consisting of card counts between 1 and 279 with a mean deck size of 29.9 cards and median of 30. The features used for this project were the counts of cards in any given run. The number of unique cards implemented into the game when data was collected was 283; each card has an upgraded version as well, bringing the final count to 566 features.

3 Model

Each run is a sparse set of counts that produce a binary outcome (either a win or a loss). Because of this, the model chosen to predict the outcomes was a Multinomial Naive Bayes (MNB). Naive Bayes (NB) models are based on Bayes theorem which states that:

$$p(a|\mathbf{b}) = \frac{p(a)p(\mathbf{b}|a)}{p(\mathbf{b})}$$

The numerator is equivalent to $p(a, \mathbf{b})$, which written out is:

$$p(b_1|b_2, b_3, \ldots, b_n, a)p(b_2|b_3, b_4, \ldots, b_n, a)\ldots p(b_n|a)p(a)$$

The naive portion of NB refers to the assumption that all the features are independent of one another. Under this assumption the following equation holds:

$$p(b_i|b_{i+1}, , b_n, a) = p(b_i, a)$$

which implies that

$$p(a|b) \propto p(a)\prod_{i=1}^{n} p(b_i|a)$$

For the purposes of predicting outcomes in this model the following notation will be used:

x_{ij} = Card i in run j, where $i \in \{1, 2, \ldots, n\}$ and $j \in \{1, 2 \ldots, p\}$

x = The vector of card counts for a single run

$p(\{V, L\})$ = Probability of a {victory, loss}

$C(\{V, L\})$ = Number of {victories, losses} in training set

$C(r_i)$ = Number of card i in run r

$C(x_i, \{V, L\})$ = The total count of card i appearing across all {victories, losses}

Rewriting the NB model in the context of this project leads to:

$p(V|x) \propto p(V) \prod_{i=1}^{n} p(x_i|V)$

where predicted outcome for run $r =$

$$\underset{o \in \{V,L\}}{\text{argmax}}\ p(o) \prod_{i=1}^{n} [p(x_i|o)^{C(r_i)}]$$

with

$p(\{V, L\}) = \frac{C(\{V,L\})}{n}$

$p(x_i|\{V, L\}) = \frac{C(x_i,\{V,L\})+1}{\sum_i C(x_i,\{V,L\})+n}$

Note the 1 in the numerator, this was included to stop the probabilities from tending to 0 when faced with a run that does not include card i. The n in the denominator offset the 1s in the numerator and kept the individual probabilities between 0 and 1.

4 Results

In order to break the data into training and validation sets a random selection was chosen from the overall data. A simple random sample was taken with 80% of the data going towards training and 20% going towards validation. A random sample was acceptable because each run was completely independent of all other runs. This gave a training matrix of 23,534x566 and a validation set of 5,884 runs. After running the training data through the model to learn $p(V), p(x_i|V), p(L)$, and $p(x_i|L)$ the validation set was tested. Comparing the outcomes from the validation set to the true outcomes, the percentage of runs that were correctly predicted by this model was 66.553% (misclassification rate of 33.447%). Compared to guessing victory for all runs, which gives a misclassification rate of 29.094%, the MNB model appears to be worse than random weighted guessing. Please see the readme file associated with the code located at $https://github.com/kenttorell/MLFinal\_StS$ for information about how to reproduce these results (and more) on your own.

5 Discussion

There are a few reasons why I believe the MNB produced a higher misclassification rate. The first is that there are many confounding variables which were not considered for this model. Some of these variables, which I will explain shortly, include: skill of the player, health going into the final fight, relics obtained throughout the run, which boss appears, which character is being used for the run, and many other unmeasurable variables such as dumb luck.

5.1 Health

Health is the deciding factor between victory and defeat. Once a players health reaches zero their game ends in a defeat. The amount of health the player possesses when entering the final fight is not guaranteed and can change from run to run. It would stand to reason that the more health the player starts with, the more likely the chance for success.

5.2 Relics

In addition to collecting cards with each run, the player also collects special passive improvements called relics. These give special bonuses such as increasing maximum health, increasing the players decks ability to deal damage or block damage, along with many other beneficial effects. These relics can change the effectiveness of certain cards and thus can determine not only the power level of the players deck, but also affects their chances for victory.

5.3 Bosses

When data was collected for this project, three different bosses could appear as the final fight. These bosses have strategies associated with them. Not only that, but certain characters are strong against certain bosses while much weaker against others.

This means that a player who could easily beat one of the bosses might still lose if presented with a boss that the players character was weak against.

5.4 Characters

Each run collected from the data has one of three characters associated with it. These character choices determine multiple aspects of the run including card choices, relic choices, starting deck, and the starting relic. Each of the characters is associated with a color; red is The Ironclad, green is The Silent, and blue is The Defect. Each card included in the game is also associated with a particular color: red, green, blue, or colorless. While normally each character may only choose from cards that correspond to their player color (with colorless cards being available to all characters), there is a relic in the game which allows the characters to pick from any card pool, regardless of color restrictions.

5.5 Card Dependence

Perhaps the biggest reason for why MNB fails is because NB models rely on the independence of the parameters. *Slay The Spire* decks require synergy to perform well. So much so that the best card in one deck may be completely useless and actually detrimental if placed into a deck that doesnt synergize well with it. Because of this, the independence assumption of MNB fails; meaning that MNB is not a good prediction model for this data.

6 Other Models and Future Work

In an attempt to find a model that fits better than MNB other models were tested. The code used to test these models was often times already written in packages like scikitlearn or xgboost. Please refer to the GitHub link provided earlier in the paper to view the code used to produce the following results. Below are misclassification rates for some other models when run on the same training and validation sets with default settings unless otherwise specifieds.

(For reference) MNB:
33.447%

(For reference) Guess all Victories:
29.094%

Logistic regression with l1 regularization, C=1:
28.127%

Logistic regression with l1 regularization, C=100:
28.229%

Logistic regression with l2 regularization, C=1:
28.229%

Logistic regression with l2 regularization, C=100:
28.246%

XGBoost:
28.442%

C-Support Vector Classification with C=1:
28.900%

C-Support Vector Classification with C=100:
28.246%