Shane Fuller

Professor Read

CS376B: Applied Data Science

April 10th, 2019

Data Quality Plan

Summary

In this plan, I will discuss the data collection process, the modification of the files discovered, and the steps taken that resulted in my final ABT regarding the Super Smash Brothers Melee data that was collected from the Philadelphia Tournament, The Gang Steals the Script. This project could not have been completed without the work done by the Project Slippi team, and each member was extremely helpful in responding to any questions that I had presented with them. A special thank you to their passion and dedication to provide the community with new data analysis tools.

Data Collection

The data collection process took place on Saturday, March 9th at Swarthmore College in Philadelphia, Pennsylvania. The data was collected using the Project Slippi framework, which consisted of a network of 28 consoles. After each game was played on these marked setups, the .slp was uploaded to slippi.gg, where the data and stats could then be viewed. A week later, on March 15th, Project Slippi then released a JSON dump of all the .slp files in their database, so data analysis could then be done. For this project, the final data set was constructed from this JSON metadata dump.

Data Extraction

For this project, the final goal was to get the needed data from the JSON format to a CSV format, that way it would be easier to import into RStudio. Not only did the format need changed, but the data supplied in the JSON dump had to be reformatted and cleaned up. After the first download, the first step was to look at an individual .slp file, which was the metadata for an individual game, and see what was necessary to keep and what was not. Each .slp file was formatted into multiple documents, including root, stocks, conversions, moves, action counts, overall, successful conversions, inputs per minute, openings per kill, damage per opening, neutral win ratio, counter hit ratio, and beneficial trade ratio. For the desired analysis that the project was interested in, much of this data was either unhelpful or redundant. Therefore, after

examining the data, it was deemed that the only documents that were worth extracting variables out of were root, action counts, and overall.

From this step, the next choice was in discovering what variables would be needed from those three documents. For this analysis, there wasn't a heavy interest in specific combo data, but rather character, stage, movement, and conversion data were the focus of this analysis. Therefore, below show the following data that was extracted from each JSON document.

Document	**Variables Extracted**
Root	characterId stageId
ActionCounts	wavedashCount wavelandCount airDodgeCount dashDanceCount spotDodgeCount rollCount
Overall	inputCount conversionCount totalDamage killCount inputsPerMin openingsPerKill_count openingsPerKill_ratio damagePerOpening_ratio neutralWinRatio_count neutralWinRatio_ratio counterHitRatio_count counterHitRatio_ratio beneficialTradeRatio_count

Data Manipulation

Once this list was created, the data was then extracted into a flat file from a function used in Mongo, and then transferred into a CSV file, which was then prepared to be manipulated. One major issue that had to be accounted for was in the format of the .slp file, that had two players for each row. This is a result of the nature of a game, which is the competition between two individuals. Therefore, when the data is transferred over, the information for player one and player two are intertwined in the same row. Therefore, the next step in the data manipulation was to eliminate this trend, and create unique rows for each player from each game. This process involved writing code in Java to read the current CSV file, extract the data, then create two separate arrays for player one data and player two data. A screen shot of the code used is found below.

```java
import java.io.BufferedWriter;

public class TestScanner {

  public static void main(String[] args) throws UnsupportedEncodingException, FileNotFoundException, IOException {
    String fileName = "C:/Users/Shane/Downloads/slippiDataFlat.csv";
    File file = new File(fileName);

    // this gives you a 2-dimensional array of strings
    List<List<String>> lines = new ArrayList<>();
    Scanner inputStream;

    try {
      inputStream = new Scanner(file);

      while (inputStream.hasNext()) {
        String line = inputStream.next();
        String[] values = line.split(",");
        // this adds the currently parsed line to the 2-dimensional string array
        lines.add(Arrays.asList(values));
      }

      inputStream.close();
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    }

    // the following code lets you iterate through the 2-dimensional array
    int lineNo = 1;
    for (List<String> line : lines) {
      int columnNo = 1;
      for (String value : line) {
        System.out
            .println("Line " + lineNo + " Column " + columnNo + ": " + value);
        columnNo++;
      }
      lineNo++;
    }
     String[][] playerOne = new String[3067][23];
     String[][] playerTwo = new String[3067][23];

     for (int i = 0; i < 3067; i++) {
       // System.out.println(lines.get(i).get(0));
       playerOne[i][0] = i + "";
       playerOne[i][1] = lines.get(i).get(2);
       playerOne[i][2] = lines.get(i).get(40);
       playerOne[i][3] = lines.get(i).get(4);
       playerOne[i][4] = lines.get(i).get(5);
       playerOne[i][5] = lines.get(i).get(7);
       playerOne[i][6] = lines.get(i).get(10);
       playerOne[i][7] = lines.get(i).get(11);
       playerOne[i][8] = lines.get(i).get(13);
       playerOne[i][9] = lines.get(i).get(15);
       playerOne[i][10] = lines.get(i).get(16);
       playerOne[i][11] = lines.get(i).get(17);
       playerOne[i][12] = lines.get(i).get(18);
       playerOne[i][13] = lines.get(i).get(19);
       playerOne[i][14] = lines.get(i).get(22);
       playerOne[i][15] = lines.get(i).get(25);
       playerOne[i][16] = lines.get(i).get(26);
       playerOne[i][17] = lines.get(i).get(27);
       playerOne[i][18] = lines.get(i).get(28);
       playerOne[i][19] = lines.get(i).get(30);
       playerOne[i][20] = lines.get(i).get(32);
       playerOne[i][21] = lines.get(i).get(36);
       playerOne[i][22] = lines.get(i).get(37);
     }

```

```
    for (int i = 0; i < 3067; i++) {
      // System.out.println(lines.get(i).get(0));
      playerTwo[i][0] = (3067+i) + "";
      playerTwo[i][1] = lines.get(i).get(29);
      playerTwo[i][2] = lines.get(i).get(39);
      playerTwo[i][3] = lines.get(i).get(12);
      playerTwo[i][4] = lines.get(i).get(38);
      playerTwo[i][5] = lines.get(i).get(1);
      playerTwo[i][6] = lines.get(i).get(9);
      playerTwo[i][7] = lines.get(i).get(31);
      playerTwo[i][8] = lines.get(i).get(14);
      playerTwo[i][9] = lines.get(i).get(24);
      playerTwo[i][10] = lines.get(i).get(3);
      playerTwo[i][11] = lines.get(i).get(33);
      playerTwo[i][12] = lines.get(i).get(0);
      playerTwo[i][13] = lines.get(i).get(34);
      playerTwo[i][14] = lines.get(i).get(23);
      playerTwo[i][15] = lines.get(i).get(8);
      playerTwo[i][16] = lines.get(i).get(20);
      playerTwo[i][17] = lines.get(i).get(21);
      playerTwo[i][18] = lines.get(i).get(36);
      playerTwo[i][19] = lines.get(i).get(30);
      playerTwo[i][20] = lines.get(i).get(6);
      playerTwo[i][21] = lines.get(i).get(28);
      playerTwo[i][22] = lines.get(i).get(35);
    }

    try{
      PrintWriter writer = new PrintWriter(new FileWriter("slippiData.csv", false));

    for (int j = 0; j < 3067; j++) {
      for (int i = 0; i <= 22; i++) {

          writer.print(playerOne[j][i] + ",");

        System.out.print(playerOne[j][i] + ",");
      }
      System.out.println();
      writer.println();
    }

    for (int j = 0; j < 3067; j++) {
      for (int i = 0; i <= 22; i++) {

            writer.print(playerTwo[j][i] + ",");

        System.out.print(playerTwo[j][i] + ",");
      }
      System.out.println();
      writer.println();
    }

    writer.close();

  }catch(Throwable throwable){
    throwable.printStackTrace();
  }
  }
```

After the two arrays were constructed, and in the proper order in relation to one another, they were then printed to the file slippiData.csv, which then almost held the final ABT that was used. A note from the code above was in the importance of the construction of the arrays, to be sure that the rows where in the same order as its corresponding character id pair. In other words, the decision to grab the rows in the order I did for each was crucial, and if this step was done unproperly, then some rows would have been with the wrong data, completely invalidating the data set. Another note was that a new data row was constructed from this set, in the form of an opponentId. Essentially, it was important in the analysis to know who the character that the player was fighting against, therefore, the opponentId was added to the set of features.

Intermediate Steps

The final steps in the creation of the ABT was quick, simple cleanup of the data that was in the slippidata.csv file. From the excel spreadsheet, I changed the title headings, as they were formatted in a way that only spoke of player one. Therefore, I made the column headings less redundant, and changed the names to the appropriate categories. From this iteration of the slippiData.csv file, I then imported the dataset into R, and began to make the final modifications needed. First, due to the way that the code was written in the step above, the data when imported into R was all as a character type. A large majority of the data is of numeric type, so the first step when in R was converting all the data into a numeric type, that was summary statistics could be calculated. One result of this transition was that the data that had responses of “None”, where converted into “N/A” values. Those initial “None” entries were a result of the calculated ratio features, which were found from counting the number of instances for a specific player over the total number of occurrences. Sometimes, there were no total occurrences, which lead to a division by 0, which was accounted for by the “None” entry. Therefore, this transition was unproblematic. The last simple fix was in the removal of rows 3067 and 1673. Row 3067 was simply a row consisting of all “N/A” values. This was because when the file was rewritten, this included the column headers again. Therefore, this data needed to be removed. The row 1673 showed an entry with a killCount of 5, which is impossible to achieve in a game. Therefore, this data also had to be removed.

Normalization

Due to the consistency of the data collection, there was no normalization of the data required. Each of the units where all standard and where all calculated in a single game, so no unit changes had to be accounted for.

Deriving Features

Again, similar to the normalization topic, there was no need to derive new ratio features, as this data was already included in the data analysis work done by Project Slippi.

Feature Removal

The final step that lead to the final ABT was to reevaluate the features existent, and make sure that they were all necessary. The original pruning of the data came in the data extraction step, however the data that was retained was not guaranteed to be completely useful. After looking through the data in R, it became clear that some of the features were either redundant, confusing, or unhelpful to the final analysis and model creation. Of the 23 features, 5 were removed in the final pruning of the dataset. First, inputsPerMinuteTotal was removed, as the values were confusing and misleading, with especially unclear units. Next, id was removed, as the values were not helpful, and they were causing issues with the model creation. Last, the remaining four variables that were deleted were total counts, which proved to be unnecessary due to the existence of their ratios, which is a more helpful unit of measure for this type of information. These four variables were inputCount, openingPerKillCount, inputesPerMinuteCount, and beneficalTradeRatioCount. With these features removed, this resulted in the final ABT to be used for model creation.

ABT Overview

The final ABT is 17 columns wide, and 6,131 rows deep. This data contains 14 numeric values, and 3 ids. The information ranges in skill of players, characters, stage choice, and time of match. However, when conducting a tournament, there is a wide level of range for every single game that is played, all unique in their own right. Therefore, this dataset is an accurate description of what competitive Super Smash Brothers Melee looks like, and the many different personalities and skills that are presented at events like these.