McKenzie Shores

CS 480 Senior Project

April 22, 2019

Predictors of Catan

The idea for my senior project was to create a set of machine learning algorithms that could predict the winner for any given state of the board in the game *Settlers of Catan*. Abby Winegarden and I chose to work on this project together because we were both interested in doing a machine learning based project and felt that this question was significantly challenging and interesting. When designing the project we broke in it into five distinct sections: research, extracting data from an existing project, generating our own data, data processing, and writing predictive algorithms. I was to focus on research, extraction, and the algorithms while Abby was set to head up our game generator and data processing. We would have prefered to pair program the generator and algorithms since these would be the densest aspects of the project, however due to the complexity we worked mostly independently, meeting once a week to go over areas of interest or places we were stuck.

The research portion was meant to put dedicated time into learning Google's Tensorflow API. Because I would be leading the algorithms, I did a majority of the research in the first few weeks of the project before we started working on getting datasets together. I had very limited experience with Python and neither Abby nor myself had used Tenflow. I spent a couple weekends working through the premade tutorials. I did three step by step projects looking at basic image classification, text classification, and linear regression. After doing these exercises I determined that we should be looking at doing a multiclass classification. This would be based off of a linear regression followed by implementing a Sigmoid function. I chose this because we needed a non-binary output and although the prediction would be based in linear regression, the target feature was discrete. For this reason the Sigmoid function would be ideal for bucketing our data. At the time, I had not seen any Tensorflow projects that were not based in regressions so I started working towards these algorithms.

A major roadblock in working on the algorithms was the lack of explanation and examples that were available. Most everything was the same three tutorials or a slight variation. Additionally, examples tended to use the MNIST dataset. With this absence of variety it was difficult to understand why things were being done a certain way or how particular pieces of code functioned. Although the Tensorflow documentation is very thorough, it did not provide much clarity as to why functions would be used. Most of my time had to be spent tracking down definitions for various parameters to understand the requirements and purpose behind small pieces of code instead of understanding the overall structure of the programs. When we chose to work with Tensorflow we had understood that it would be more complex than other machine learning softwares we had looked at. I anticipated having a very steep learning curve, which is why we proposed an entire section for research, however I was expecting there to be more resources available. This is one of the reasons that I believe that if we had known what we were getting into with Tensorflow we would have chosen a different direction for the algorithms.

The second portion of the project came from a research paper that I found online from some students at Tilburg University. They were trying to determine if the seating order of the players had a measurable impact on who would win a game of *Settlers of Catan* using a Java project they created entitled *Smart Settlers*. They found that the player who went first had a slight edge in the game and their paper can be found in full in our proposal. We were interested in their project because they created a dataset using their own game generator by implementing a Monte-Carlo Tree Search to make decisions and run full games of *Settlers of Catan* very quickly. We had decided that making any sort of artificial intelligence for our generator would be far too complex given our experience and time. We still wanted to create our own generator which would make random decisions however we used *Smart Settlers* as a secondary dataset. Both sets would be run separately and in combination through our predictive algorithms to try and get the best possible accuracy.

When it came to extracting the data from *Smart Settlers* we decided on seven features. Resource strength was defined as the sum of all of a player's tiles's probabilities of being rolled. Probabilites were determined by the values given to each number by *Settlers of Catan*. For example the game gives the number two a score of one, ten a three, and eight a six so a player with these tiles would have a resource strength of ten. Hand strength was said to be the variety of a players hand multiplied by their total number of resource cards. Victory points are the players current score in the game. Cities is the count of how many cities a player has built, we decided this was necessary because once a player starts to build cities the resources they can acquire increases. Development cards is the total amount of development cards a player has drawn over the course of the game. Round was counted as a numeric value and lastly winner was recorded as a binary value of whether or not a player won the game.

For *Smart Settlers* I recorded these values in a two dimensional array of all of the player's features and wrote them to a .csv file after every round, setting all of the array's values back to zero once they had been written. The project's main class was over 2000 lines long with very little use of objects relative to the size, mostly making use of if statements and switch cases.The lack of design and commenting in the code made it difficult to understand and track down the values that I was looking for. Calculating resource strength was particularly difficult because all of the board and player data was contained in a single array which did not hold onto a player's tile values. As opposed to our design which utilized seperate objects for each player with convenient getter and setter methods to easily understand how various data points were being distributed throughout the game. It is worth noting that *Smart Settlers* was not designed to extract data points and it's creators were only interested in which player won each game and not how they obtained their victory points.

While I was working on extracting data from *Smart Settlers*, Abby was working on generating our own game generator. We looked at some of the intelligent aspect of the the existing generator's code but decided to stick with random decisions so that the ours would be completely separate. We decided that Abby should lead this portion because she had more experience in object design than I did. We wanted to make sure that, unlike the existing project, ours would be easier to comprehend and follow design patterns. For the most part my contributions to the game generator were limited to our weekly meetings implementing test code and fine tuning methods. We also worked together on some of the more complex aspects of the design. The largest section being working out the decision making object. Although we wanted all choices to be random, a game would never finish if we let player only build roads or buy development cards instead of building settlements or cities. We determined that the object should always check if it can make a play in the order of buying a development card, building a city, building a settlement, constructing a road, and lastly trading with the bank. This way a player would not be spending all of their resources on a road, worth no points, when they could gain points from the other three options. Additionally, the order of development card, city, settlement, favors decisions that will yield points more quickly than the others.

However the densest part of the decision object was deciding where a player should build roads. We wanted them to be constructed in such a way that we could easily calculate which player has the longest road. Longest road is determined by the player with the longest continuous road and is worth two victory points, thus we wanted to mimic a human player's mindset by not have a bunch of random offshutes that cannot be counted. Initially, we had no way of connecting the hash tables of valid road placements to our settlement placements. This meant that we could see if a road was available but not if it made any sense to our player. To combat this we created a new hashtable that made associations between available road and settlement placements. In hindsight we should have put more consideration into the decision making object when working out the larger design of the game generator. Given more time in the beginning of the semester we should have spent more time working out the details however neither Abby nor myself believed that the generator would become as complex as it did.

Once the game generator was up and working I started working on processing the data in a new Java project. In our proposal this was initially Abby's role however because I had already been working on collecting data from both projects and had a better understanding of what needed to be done it made more sense for me to write it. Both projects had two features that needed to be edited. Round was written to the file as a numeric value but because of the variance in game lengths I converted it to discrete. I calculated the length of each game and then set round to either 1, 2, or 3 depending on which third of the game I was looking at. Winner also needed to be changed from a binary value to the Player ID of the winner. Later we decided that it would be more helpful to run our algorithms with the binary value so I ran all of the data through a second time maintaining the original binary target value. Lastly the data was split into training, 80%, and testing, 20%. Overall this section of the project was very straightforward and felt more like busy work than anything with clear learning objectives.

After all of the data had been processed we started work on our Tensorflow algorithms. Because of the unanticipated complexity of the game generator we only had about three weeks with our data putting on a lot of pressure to get something together by the end of the semester. I had been working on a trial dataset and had a binary classifier using linear regression close to working. It was not until near the end when I realized that I had misinterpreted the output that we needed. My algorithm would output whether or not a player was likely to win but did not consider the other players in the game. At this point we made the decision to work towards a random forest algorithm that would take in all four player's feature as a single instance, with the target feature being the winning player's ID. I found an example of a random forest for Tensorflow online and got it running fairly quickly, however the accuracy was a dismal 46%. random guessing would put us at 25% but I was still very disappointed in our results. In our weekly meeting Abby and I tried to output a confusion matrix or ROC curve to see where we could make improvements but could find no resources for outputting this data in a random forest.

Abby then found a tutorial using boosted trees that outputted a ROC curve and all of the data necessary to create a confusion matrix. It would also give us the the prediction that the algorithm was making for each instance as a numeric value. We decided that although it was not initially how we envision our algorithm functioning it would be better to have an understanding of the data than our trying to improve the 46% accuracy. Abby was already close to having the boosted trees working for our dataset so she finished up that algorithm. Afterwards we took the outputs for each instance to find the maximum value for each round to give us our predictive winner. I went back to working on doing binary classification by linear regression so that we could see which would give us a better accuracy. I am confident that we will be able to output much better predictions from binary classification however at this point it is still a work in progress. Given better time management in the beginning of the project we would have had the time to convert the data that we got into finding the overall winner for a round instead of whether or not a singular player was likely to win.

Although with our random forest algorithm we did answer the question that we initially asked I think our project could have been vastly improved if we had a better understanding of Tensorflow and the complexity of creating a game generator at the beginning of the semester. After using other machine learning softwares not related to the project I believe we would have been better off using something like Weka's API. Tensorflow is an extremely powerful tool however it is best suited for heavy mathematical computations and was not the best fit for what we were trying to do. We should have understood our own questions better and looked into software that was better suited for classification algorithms. This coupled with how long it took to finish the game generator put us in a less than ideal situation for finishing the project. At the start of the semester we should have put more time into understanding how the game generator would function and working out the details of the object design. Because it was supposed to be a completely random generator, we did not think that it would become so complicated. We neglected to think about the interactions between different aspects of the game which lead to a much longer process. Even though we misunderstood how our algorithms would function we still accomplished all of our goals. I greatly expanded my knowledge of Python and Tensorflow. Through working with Abby I learned the value of pair programming as well as a greater understanding of object design, which I had not set as a specific objective for myself.

Grading Scale

(20pts Total) Pre-Existing Game Generator - McKenzie (**20pts earned total**)

- (10pts) Using game generator found on Github: *Smart Settlers,* review code to understand how the game works. (**10pts earned**)
- (5pts) Get *Smart Settlers* to interact with our data collection methods i.e. modify/create code to output necessary data. (**5pts earned**)
- (5pts) Generate and run 10,000 games and extract data. (**5pts earned**)

(65pts Total) Building Game Generator - Abby~~ + McKenzie~~ (**63pts earned total**)

- (5pts) Create board object (**3pts earned**)
    - (3pts) 19 resource hex objects (**3pts earned**)
    - ~~(2pts) 9 port objects~~ (**0pts earned**)
- (5pts) Create resource decks (**5pts earned**)
- (5pts) Create development deck (**5pts earned**)
- (30pts) Create a player object (**35pts earned total**)
    - (2pts) Victory point count (**2pts earned**)
    - (2pts) 15 road objects (**2pts earned**)
    - ~~(2pts) 4 city objects~~
    - ~~(2pts) 5 settlement objects~~
    - (4pts) city and settlements were combined into one object (**4pts earned**)
    - (2pts) Deck/hand (**2pts earned**)
    - (5pts) Player understand rules of game and legal moves (**10pts earned**)
      ***This aspect of the Player was far more complex than anticipated**

      ***This aspect was moved into its own object**
    - (15pts) Decision making (**12pts earned total**)
        - (5pts) Resource use (**5pts earned**)
        - (4pts) Development use (**4pts earned**)
        - (6pts) Basic trading (**3pts earned**)
            - Trading is 4:1 to the board ~~or with any port object the player has access to~~.

- (2pts) Create robber object (**2pts earned**)
- (12pts) Gameplay (**10pts earned total**)
  - (4pts) Initial set up (**4pts earned**)
  - ~~(2pts) Randomly decide which player goes first~~ (**0pts earned**)
  - (2pts) Initial placement is also random (**2pts earned**)
  - (4pts) Resource allocation (**4pts earned**)
    - (2pts) Allocate starting resources (**2pts earned**)
    - (2pts) Allocate resources (if any) to each player on each die roll (**2pts earned**)
- (5pts) At the start of each round collect data from each player (see Data Collection) (**5pts earned**)
- (1pts) Run 10,000 games (**1pts earned**)

(**15pts Total**) Data Collection - ~~Abby~~ McKenzie (**15pts earned total**)

- (8pts) Scripts will be written to collect data from each player at each round in the game (**8pts earned**)
- (3pts) Collect data (**3pts earned**)
- (2pts) General overview of data to remove outliers (**2pts earned**)
- (2pts) Divide data into training set and testing set (**2pts earned**)

(**40pts Total**) Machine Learning Prediction - McKenzie ~~+ Abby~~ (**40pts earned total**)

- (15pts) Tensorflow research (**15pts earned**)
- (15pts) Write prediction algorithm (**15pts earned**)
- (3pts) Make predictions based on each generator individually (**3pts earned**)
- (4pts) Make predictions based on both generators (**4pts earned**)
- (3pts) Make comparisons about accuracy (**3pts earned**)

Total Points earned: 148
Original Grading Scale: 150pts Possible

≥ 135pts	A
134pts - 120pts	B
119pts - 100pts	C
≤ 99pts	D

Recommended Grade: A