Commentator Bot - Absurdly Intelligent

ECS170, Spring 2018

Group Members

Please place all group member names, email addresses, and student IDs.
Saksham Bhalla, iambhalla@ucdavis.edu, 912778394
Yash Bhartia, ybhartia@ucdavis.edu, 912969611
Rohit Nair, rhnair@ucdavis.edu, 912746847
Aakaash Kapoor, aakkapoor@ucdavis.edu, 912719191
Hung Mai. hqmai@ucdavis.edu, 914362274

Problem and Contribution Statement

Currently in StarCraft II, players can watch the replay or look at the post-game stats to see how they matched up against their opponent. By watching the replay, a player can see what actions their opponent take throughout different phrases of the game. But, it may be difficult to determine which actions or phrases in the game that is a key turning point. Our tool can act as a live commentator on their own gameplay and make watching replays more fun; at the same time, assist the player to identify key points in the game. Based on our preliminary research, we believe that no significant work has been done on this problem and the only tool that the new users have right now is watching professional tournaments or streamers on youtube or twitch and learn from there. However, we believe that live commentary on the player own's replay would significantly help them learn to play better.

One of the focuses of artificial intelligence in general is to try and model the human mind and philosophy, and this could be related to the problem we're trying to solve with this bot. Commentating is a job that great Esports casters take years to master, and that too the problem is they would never cast a game that happens between two casual players. A solution would be a bot that is able to provide useful information on impactful events happening in the game. We wonder how accurate we can make a bot that learns how to commentate and what to commentate on games depending on how important an action is. We know that this will prove to be a fun and challenging problem to solve.

We will implement Support Vector Machines that first look at the list of actions that happens throughout the game and figure which actions are more important to provide commentary on. Support Vector Machines are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Since we will have a lot of data to analyze given our replay dataset, a support vector machine would be able to predict whether a task is essential to the game. While training we will classify each task of the winner as a 1 and classify each task of the loser as a 0. This will allow us to predict the important task given more than one task.

Possible alternatives to determine how well a player did are the post-game stats, or an actual StarCraft II caster, however hiring a caster for a casual game is not realistic. Amateur players may find the post game stats to be overwhelming to analyze and pinpoint what aspect of the game to improve on. It also requires that spectators wait until the game is over to see how the players played relative to each other. Our solution, hopefully, will provide action-by-action (both important micro and macro actions) analysis to the user in real time.

Throughout the duration of the project, we were adamant on solving the original problem. We proposed on making a commentator bot that would be able to make intelligent commentary on a real Starcraft II game. Since this was a four week project, naturally we had to scope our project accordingly. At the end of 4 weeks, we were successful in setting up a commentator bot, capable of making intelligent commentary on any given replay. However we did slight modifications to our proposed contributions to solving the problem.

In our proposal, we mentioned that our model would be able to identify key points in the game and providing live commentary on the status of the game. In addition to this, we also proposed a bot that would be able to analyze whether an action is important enough to commentate. However, after analyzing the data, we decided on scoping our project in a slightly different manner. Rather than the bot understanding the importance of an action, we manually picked out actions that we thought would make a difference in the result of the game. This is because once we analysed the format in which we get the replay, we understood that some events happen a lot more often than other events (For example, Camera Events, Selection Events, etc.) and for an AI model determining what led to a victory for a player, these events should not considered of any importance. For other events, even though they could add value on determining the winner, the format of the trigger event that we get from API was too hard to contextualize and commentate on in simple english that users understand.

Other than this slight modification, we were able to make intelligent commentary on a replay. We are able to contextualize our data and not print every event naively. Moreover, we are able to make a biased commentary based on how well a team is doing at a particular instance in the game.

One of the primary reasons we chose this solution given possible alternatives was the fact that this solution was feasible given the time restrictions. Some of the important tasks that were required for the completion of this project were much more challenging than we had anticipated. One particular naive task that we all struggled the most in was successfully being able to parse replay files. Since most of the replay parsers work on previous versions of starcraft and are not updated frequently by developers of the Starcraft community, it was very challenging to find a good replay parser that works on the most recent starcraft generated replays.

Design and Technical Approach

For this project we are using NLP techniques such as normal word processing into natural language and SVM for training purposes. We will have more than one type of Support Vector Machines and we have to choose from within those. To do so, our training data will be split into training and testing , and we will find the best accuracies within each model and choose the best model. Additionally we will not be implementing a neural network because our input layer would have a range of data namely but not limited to the health of each building and health of each monster. Although neural networks would work with a big input layer, it would only work if we have a lot hidden layers and the computation time would thus be slower. Since we want real time commentary on the replays, a slow approach is not ideal.

Hopefully we have access to decent amounts of data to make our model more accurate. We also use natural language generation models to contextualize our results in a way that the users or spectators can understand. The general problem is that human work is necessary in commentating on sports, and that games that don't have commentary may be hard to learn from or understand. Making a bot that can at least recognize important actions in a way that it emulates casting in StarCraft II is a beginning in trying to solve this problem.

Our output layers would be spitting out 1 hot vectors that would trigger different functions. We would have a natural language sentence generated function which would generate sentences around selected sentence blanks associated with word types. For example: "_____ won the game!" The blank is suppose to be a proper noun. The rest of the words are generated to indicate the blank won!

The reason why we feel this bot is relevant is because the bot is converting raw data in language humans will understand. The raw data will be converted by Natural Language Processing for players playing the game to understand in terms of normal english. We are going to train on previous game data and decide which

plays were the best based on PPM or other achievements the player is getting.

The technology stack for this project will use Python modules such as NLTK and TensorFlow for the machine learning and the natural language processing side of the development.

Our technology stack is the best choice for our development in that we will use large amounts of data to mold the perfect/best output to the users' game. NLTK and TensorFlow are developed by Google which uses large amounts of data for training. This will be beneficial for our project with not as much data, but the training functions provided will work best for our goal.

For this project we are going to be using multiple developing assistant softwares which will make the workflow convenient. To write code we will use IDEs where coding guidelines can be set in stone. And for Git interface alongside a work assignment board we will use Gitkraken and Gitkraken Glow.

For this project we will use github as our main code base. This will ensure that each developer can have their code made available to the team all the time and also have a secure backup on a cloud server. We will maintain two main branches: Master and Developer. Our team will push to Master only during team meetings and at other times we will push to the developer branch. Apart from these two each team member will have their own branch which they will use to test their

code and then update the developer branch.

We will also use a pull first and then push format which will ensure that we do not have conflicts and those that we do are checked before submission into Developer branch, by the developer himself.

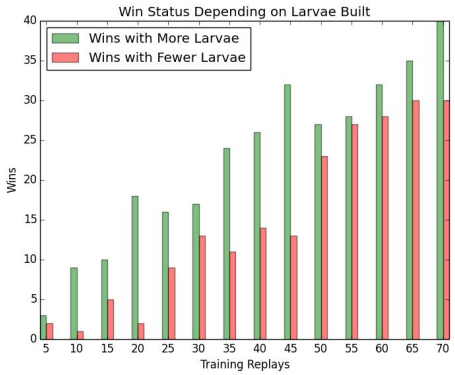
We will follow a strict coding guideline sheet which we have written up. This guideline sheet will allow our code to be readable, reusable and documented at all times. We are trying to figure out if we can make these rules strict on an IDE like atom or sublime text our work would become a lot more reliable and better.

As discussed in the previous sections, getting replay data from the replays was the most time consuming part of the project. It took us almost two weeks to get it up and running. This caused us to prioritize the tasks that we would want to do in the project and which led us to not use NLP in creating sentences. The moment we got the replay data, our priority became to parse the data, and get meaningful events from the gameplay. We were creating a commentator bot and the priority was to get meaningful data from the replay.

Therefore, we had a parser file made that would go through the replays and parse the data as required.

We were mainly focused on the events object in the replay, which contained all the events that happened throughout the game. This object had data from the player info, to the object as well as the time it happened. From the events, we had to take out two major forms of information. a) Hot Vector Data for the SVM to run on and predict the status of the game. b) as well commentator data that can be converted to understandable data.

To run an SVM and predict the status of the game we had to send in data that an SVM can understand and compute. So first we did was to create arrays of events such as, it had the time it occurred, then the a 1 if event occurs and 0 if that event did not occur. Thus, creating one-hot vectors for each event, having a time in the start. This raw data was then passed to SVM Handler, that would create more meaningful data from it. We wanted our commentator not say biased commentary on all stages of the game. But to commentate on its thinking on who is winning the game at each minute. This goal forced us to concatenate the raw data, into an array one hot vector that we would feed in the SVM and give it whether we won or not, thus training the data. Our approach changed from the proposal, since we are now not focusing on what objects are important, but commentating on who is winning the game at each minute, because we could not find a way to prioritize events due to lack of time.



Additionally, our hot vector for the svm changed throughout the project, based on results. Some events were more important than the others. Like camera events in the game, were not as important as unit died events. We prioritized these events based on results that we got from the SVM. For example, Zerg spawning larva was an event that happened quite

often, and the commentator bot would use a lot of comment to convey this general action. In order to justify ignoring this particular action altogether, we tested wins with difference in larvae spawns in Zerg v Zerg matches. We see that more larvae consistently lead to more wins, so the final decision was to not commentate on the fact, but keep it in the one hot vector for future predictions.

For the commentator data, we first wanted to run NLP techniques to create sentences. But we were not able to do that since, we had object names and usernames changing at all points. TrueMaster.py, which is the main file of our project, got the data from the parser and called our print library that would print sentences based on time. We then had a print library, that would take in arguments and create sentences using simple string concatenation. We have 4-5 sentences for each case and a sentence is chosen at random to be printed on the screen, trying to make our AI, as human like as possible.

Additionally, we could not create sentences from code so we focused on user experience. First, we created a animate file that would animate our print functions, as well structure the history of comments that the user sees as well as the time. The structure included a 5 line history for normal comments and 5 line history for AI comments as well as a current sentence space, as well as a timer on the terminal. This created a somewhat interactive experience for the user, other than just simple prints on the terminal .

Scope and Timeline

Although this task can be a bit overwhelming to complete in the span of one month, we believe that with a perfect plan and timeline, we will be able to complete our project and evaluate the feasibility of this product. In the timeline given below, we have set every week's goal. However, we also plan to set smaller goals to ensure completion.

We had to make a considerable amount of changes to our timeline since we learned some tasks took a lot more time than we had anticipated. These tasks also led us to redefine the scope of our project. For a very long time, we were not able to get, readable, easy to understand replay parser. As discussed in previous sections, most of the replay parsers were only able to parse files from previous versions of starcraft which would defeat the purpose of making a commentator bot. Once we learned about sc2reader, we realized that the format in which we get our data is not the way we were expecting it to be. The anxiety associated with these changes in our assumptions led us to redefine the scope of our progress. Once this scope was redefined, we had to expedite our progress in order to come up with a working product. Looking back to the past four weeks, we believe that we finished twice as many tasks in the last 2 weeks as compared to the first two weeks.

Week 1 (Monday, April 30 - Friday, May 4)	• Getting familiar with the gameplay and the data given by the api. • Figure out how to perform a API request in Python to get desired data. • **Learning about different python wrappers around Starcraft II Blizzard s2client-proto https://github.com/Blizzard/s2client-proto.** • **Reading documentation on how pysc2 works. Trying to setup pysc2 and parse replays. https://github.com/deepmind/pysc2** • **Setting up github repo and reading papers on how to get replay data in a readable format.**
Week 2 (Friday, May 4 - Friday, May 11)	• Understand game mechanics and be able to figure out manually what important data is. Discuss and come up with strategies for machine learning algorithms. • Figuring out battle phase and build phase of the game are and how they can be distinguished using what the API gives us. • Figuring out how to tackle the case when both battle phase and building phase is going on simultaneously. • **Rather than parsing a real replay, trying to train on replays predefined in https://github.com/wuhuikai/MSC library. The MSC repo contains 40000+ replays and would've possibly helped us train our SVM model extensively. After almost being able to get a well defined vector with all events that happened in the game, we gave up because we were having major problems with being able to run one of the primary scripts for this repo.** • **Meanwhile, the other half of the group was still working on getting pysc2 to work. We tried to downgrade our version of Starcraft and create replays that we would be able to parse.**
Week 3 (Friday, May 11 - Friday, May 18)	• Run over multiple replays to train the data. • Find what we think is the time when he stopped building.. then we can comment on what happened in the building phase and summarize rather than just bluntly flushing data. • Maybe we can find out what happens different in different games to figure out what leads to a win or a loss. • **We eventually gave up on using PySC2 and MSC dataset and this was the first time we were able to parse a replay using the sc2reader API. https://github.com/GraylinKim/sc2reader** • **After being able to successfully parse the replay, we fast tracked our progress and started building a training and testing dataset.** • **Meanwhile, the other half of of the group was building a library to print out our results in a more human way.**

	They implemented a animated printing library which gives our project a more human like characteristic.
Week 4 (Friday, May 18 - Friday, May 25)	• While this building phase is going on, we can give a pre analysis of different strategies. • We can evaluate the tradeoff between resource loss and power gain to figure out if it was a good strategy. • Implement a Natural Language Generator to summarize the results in a more readable way. • Prepare deliverables and perform rigorous testing. • **Once our training and testing dataset was complete, we started implementing our SVM to predict the winner while the game is still running.** • **The replay parsing library was improved in order to get more characteristics from the replay.** • **The printing library was improved such that it could synchronize with the actual replay timings.**
Week 5 (Friday, May 25 - Friday, June 1) and Week 6 (Friday, June 1 - Friday, June 8)	• **Completed implementation for the SVM and trained it on about a hundred replays to get a better accuracy.** • **The printing library was further improved and was able to distinguish between a naive comment and an intelligent comment.** • **Implemented a graphing library to illustrate results in a more comprehensive way** • **Completed working on all deliverables and performed testing.**

Documentation and Access

We are planning to post our project on to github with a detailed readme. This readme should briefly talk about the problems we're trying to fix and why we believe that our solution fixes the problem. Then we will explain how we structured the project and how to effectively use the product. We haven't yet decided if we're planning to make a website or not since our end product is not something that can be explained better using a website.

We used github and have a readme that details how to use effectively use the project, and a separate website that talks about the problems we're trying to fix and why we believe that our solution fixes the problem, and how it works. The github repo and website links can be found below.

GITHUB REPO: https://github.com/ybhartia/170Starcraft

WEBSITE: https://goo.gl/Aordja

Documentation for other Python Libraries used:

- https://matplotlib.org/
- https://github.com/GraylinKim/sc2reader
- https://gggreplays.com/landing_tour

Evaluation

On severe research we found out that there are almost no bots that do the task that we have taken upon ourselves. The creators of the bots that exists haven't spoken in their documentation about evaluation and just claim to be “good” at commentating. In order to evaluate the success of this project, we will be looking at the outputs of our program over a small number of simple replays. Our evaluation will be based on the grammatical correctness of English sentences, that are the output of our program. At the minimum, we want our program to generate understandable sentences. We will create a scale between 0 and 100 where we will know 0 is below satisfactory and 100 is brilliant. The score the scale generates depends on a variety of things. The list includes but is not limited to things like, number of times same sentence was used, appropriate use of sentences at appropriate times. Hopefully are scale is more generalized and then we can create a far and evenly distributed.

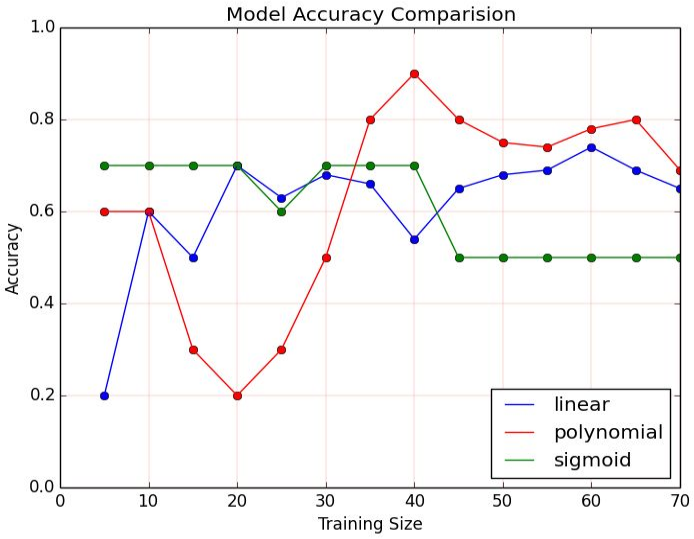
We are using GGGReplays to get replays for real matches from different players of the Starcraft community. Replays had to follow several criteria: none of the users involved had special characters in their usernames so that ascii code conversions would work properly; and there had to be actions coming from both teams, or the project would not learn; there also had to be a clear winner, because there were a few edge case replays which had two winners. We found varieties of replays from different races and race crossovers to better fit our model. We also found different varieties of replays based on how the skill level of the players. More specifically, we got some replays in which the winning team was evident and others were it was a close encounter between the winning team and the losing team. This strategy helped us understand how our model is doing on different replays and it also helped us improve our model to work on more challenging, close encounter games.

Our commenter bot was commenting on creation, deaths, upgrade and different alterations of bots and buildings. We also commented on the fact that which team is winning and when the scales are tipping over. This is slightly different from our original plan of commenting on game changing actions and generating our own sentences. Thus we had to come up with new evaluators. Sometimes it was important to be able to tell who was winning at each point of the game. Sometimes we were concerned if our bot was parsing the data correctly and did not parse incorrectly.

One evaluator we have is a winning predictor evaluator. At each point of the game we check if the commenter knows who is winning based on the events that were triggered. This is important because for the commenter to have a more human like feeling we would like it to maintain biases during gameplay. And like an unethical commentator we will make him favor

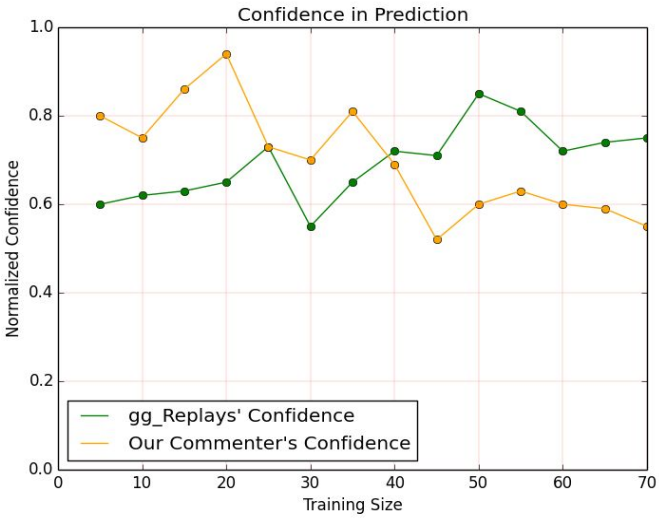
the team that is currently winning.

Data and Analysis



Our SVM was implemented with 3 different functions, linear, polynomial and sigmoid. The graph shown on the left shows the accuracy of our model's final verdict meaning the last prediction that the model made against the actual outcome. As you can see the polynomial function had better predictions given larger amounts of training data, and all functions did relatively well given around 30-40 replays for training.

In the graph to the right, we judged the confidence of our model by comparing the winning predictor values at minute intervals with the values of the ggreplay mineral + gas + supply data on the same replay one same intervals, after doing training on various data sizes. Both of these values were being compared to the outcome of the game. Our commentator bot was more accurate than ggreplay throughout the intervals when trained with around 20 replays. This is the closest we can get to judging the actual accuracy of our AI biased comments.



Results

The comments of the bot look neat and are synced up with the game, so the ability to commentate seems well put together. We found that the ideal training / testing split for our model was 40-60, because it best mimicked gggreplays and had best accuracy at that point. It had seen at that point various sets of play styles and various types of games and races, and it is possible that looking at more games than that would result in overfitting. Overall we were content with our accuracy and confidence because it was difficult to determine whether

events in each minute interval had the lasting impact on the outcome of the game.

Plan for Deliverables

As this project is not a StarCraft Bots, we won't be submitting to a tournament, but we are planning to post this project on Blizzard StarCraft II API forum, which can be found here: https://us.battle.net/forums/en/sc2/22814033/

We believe our project still has room for improvement, and we may work further on it to have a better impact on the subcommunity of Starcraft AI that commentate, that teams in the future can look at and build on before we submit the project to any forums. However we will make the github public so anyone interested can check us out.

GITHUB REPO: https://github.com/ybhartia/170Starcraft

WEBSITE: https://goo.gl/Aordja

Separation of Tasks for Team

Implementing the interface code: Rohit and Hung

Strategizing effective ML algorithms to train data: Saksham and Yash
Implementation and optimization of ML algorithms: Yash and Rohit
Implementing a contextualizer: Aakaash and Hung
Implement a Natural Language Generator: Saksham and Aakaash
Testing and Code Review: Yash and Hung
Evaluating Github Workflow: Saksham and Aakaash
Inline Documentation: Aakaash and Hung
Deliverable Reports: Rohit and Saksham

Implementing the interface code: Yash and Saksham
Creating a Data Set: Rohit and Hung
Parsing the Data Set: Yash and Saksham
Strategizing effective ML algorithms to train data: Yash and Aakaash
Implementation and optimization of ML algorithms: Aakaash and Rohit
Implementing a contextualizer: Rohit and Hung
Testing and Code Review: Yash and Saksham
Evaluating Github Workflow: Yash and Saksham
Inline Documentation: Aakaash
Deliverable Reports: Saksham

Since we had changed the scope of the project during these four weeks. Our tasks also changed slightly.

Saksham Bhalla and Yash Bhartia mostly worked on building the dataset for training and testing purposes. For the scope of this project, we were parsing our replay data using sc2reader library. It was Saksham's responsibility to analyze the dataset and read documentation on all game events that are triggered and recorded by the sc2reader API while Yash worked on creating an Object-Oriented Design for our library. Using this insight, Saksham and Yash worked on creating data vectors and events dataset unique to each team in the game. These data vectors include critical information (such as UnitBornEvent, UnitUpgradeEvent etc.) and formatted according to the needs of the Machine Learning team. Once this information is extracted, this vector is used for training and testing SVM implementation. Additionally, Yash worked on graphing our data and providing the result in a more comprehensive way while Saksham worked on preparing the deliverables.

Rohit Nair did research in early stages on how to identify different player strategies and workout how to determine if a player was doing better than their opponent. Since pivoting, Rohit worked mostly on functioning of commentating, which involved calling comments per event given and syncing in accordance with time and contextualizing the data. Also implemented AI biased comments every minute given the predictions from the SVM. Rohit also downloaded 100 handpicked replays, majority coming from gggreplays.

Hung Mai created different English sentences for different events (such as UnitBornEvent, UnitUpgradeEvent etc.). For each event Hung wrote several lines regarding the type of event happened, the unit involved, etc. A line would be chosen a random. Some events such as UnitTypeChangeEvent covered a broader range of in-game events, including when Zerg's Larva are produced, Terran's and Protoss' units changing modes. Hung also created some scenario specific replays to be used for testing.

Aakaash focused on the SVM part of the project creating the library for that and help writing handlers that would make sense of the data and send it to the SVMs. Additionally he was helping rohit and hung write the print libraries and also wrote the trueMaster file for running the project. Also he was responsible for overseeing the project and making sure all the work was done in order and done on time.