VCT Esports Manager Challenge

Kyle Zhou, Yufan Yang

November 5, 2024

This document is a TL;DR of a full methodology write up also included in this submission. If anything seems to be glossed over or you want more details, it is likely addressed in the full write up. We think there are some very interesting data driven results and encourage the reader to check out the complete version.

1 Methodology

Considering that no one on our team has ever worked with AWS before, has experience building LLMs/transformer applications, and that we are a very late entry into the competition, we have one main constraint that we need to design around: we need to build this product in 2 weeks. We settled on an implementation that would blend well understood unsupervised learning methods with the cutting edge LLM toolbox in AWS Bedrock. This serves to lower the amount of time spent tagging data and wrestling with new, complex concepts as well as to buy some time to learn the AWS tango and create a quality final product. That being said, there are a couple core assumptions that we will operate under, which are detailed in section 2 of the full write up.

Ultimately this project can be divided into two stages: data mining and application creation. The idea is to use GMM and PCA to isolate the best players at each role and from each league and region, generate probability distributions on a per role, per league, per region basis, and sample out of these distributions to generate teams. At this point we can pass the teams to an LLM for context aware filtering and analysis until we end with the "best" team. Then it is a matter of presenting an LLM session to the user with a natural language breakdown of this team consistent with the challenge requirements.

1.1 Data mining

1.1.1 Gaussian Mixture Modeling

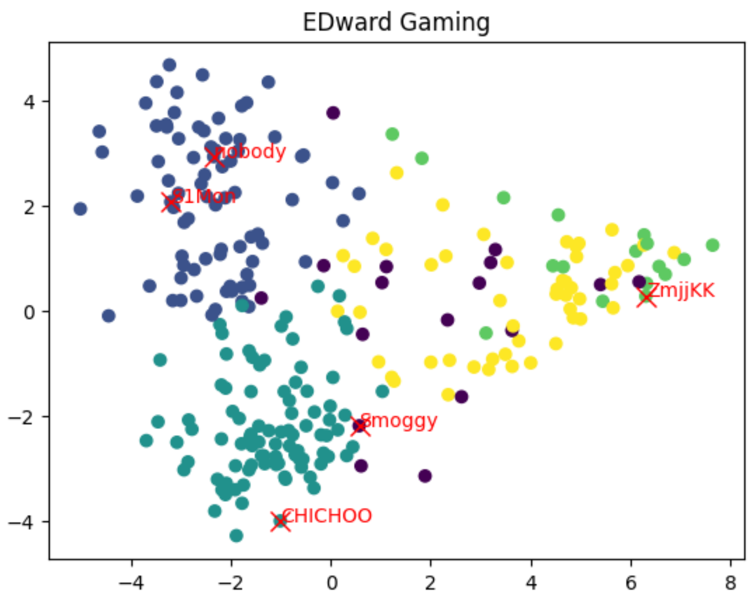
We start our data mining by looking at the VCT International Dataset. This is because it is the sampling of the best players in Valorant as well as the one we have the most familiarity with in order to sanity check results. As mentioned before our data mining follows a very typical classification workflow, with clustering being a natural place to start. Since we have no ideas on what our data's shape is, we decided to do probabilistic clustering rather than distanced based clustering. We decided to use a BGMM model as described by section 2.1.1 in the full methodology.

Before we do clustering we first need to define the components of $\vec{p}$ to best capture a player's profile. Here its important to remove any personal intuition about what makes a good player good so as to minimize personal bias fitting during the unsupervised learning process. While this intuition is valuable, it can be mixed in to the agnostic output of our models later and with more precision. For detailed raw stats used for components, please refer to section 2.1 in the detailed methodology explanation document.

After ingesting every game and constructing round statistics for each player, the individual round statistics can be combined to obtain a player's aggregate $\vec{p}$. Augmenting these vectors together gives us a matrix P that we can data mine.

The components of P can vary wildly with scale and needs to be hit with the normalization stick to stabilize the dataset. We chose a Yeo-Johnson transformation to do the normalization.

Now with clean and normalized data, we can run BGMM and expect decent results. Then by doing a quick PCA and plotting the first two principle components we can visualize the clusters.



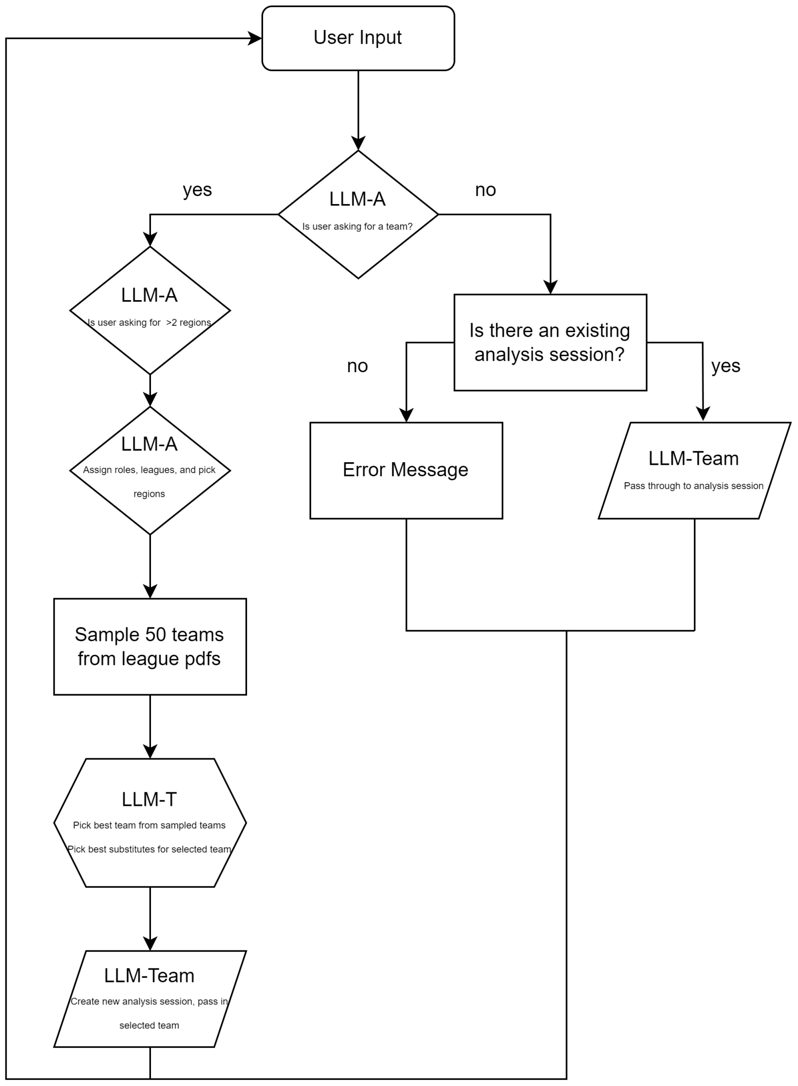
Then by overlaying the players for each role on top of this chart, we can start to look for patterns to filter and rank all players participating in VCT International by role and by performance.

1.1.2 Principle Component Analysis

When looking at known players and the clusters it quickly becomes apparent that the clusters by themselves are not good enough to identify roles and certainly not good enough to rank something as ethereal as performance. This is where the PCA comes in. By studying patterns in player spectral decompositions we can gain additional insight into behavioral patterns of roles while remaining agnostic to personal bias. We can then use the patterns we find and introduce some intuition of what the ranking of VCT Champion players are to define a generalized ranking algorithm that can be applied to the results of the PCA filtering. Once the ranking and player pools for each role and region are decided, all we do is another quick normalization and obtain probability distribution functions (PDFs) for reach role for each region. Looking at the singular values, we determined that 90% of the variance in P is captured by the first 30 principle components (PC_{1-30}) and will be limiting our analysis to those 30. For detailed analysis methodology, please refer to section 2.2 in the detailed methodology explanation document. We use these principal components to generate a pdf for each role and use the pdfs when we sample teams in the next step.

1.2 Application

Now is where the fun stuff happens. When user input comes in requesting a new team, we intercept that message and sample 50 teams, comprising of one player from each role from regions according to the user's request. Then we pass those teams with the respective $\vec{p}$ vectors into a prompt engineered LLM (LLM-T) and have it return the best team as well as a list of good substitutes for each role. Then we pass this team and the subs to another prompt engineered LLM (LLM-Team) to create the final analysis for the end user and a chat session for the user to ask additional questions about the team. This method relies heavily on the LLM base models to have good contextual awareness of what Valorant is, what a Valorant game involves, and what the components of our vectors mean. Thus we pick Claude Sonnet as the basis for LLM-T and LLM-Team to leverage its size and intelligence. In order to make the user experience more seamless, we need to do classification on every user input to see if it is requesting a new team. We can once again turn to the power of an LLM (LLM-A) rather than training up a new classification network. If we detect a new team request we can reset the LLM-Team context and generate a new team. Otherwise we can pass the user input to LLM-Team to give the illusion of a direct chat session. After checking whether the prompt is requesting a new team, we also prompt LLM A to see whether the request is asking for a team with 3 or more different regions. If this is the case, we can ensure that we select a diverse group of players. Otherwise, we try and stick to the VCT import rules of only one imported player. Both of these classifications need to be fast so we decided to use Claude Haiku to save on cost and minimize latency. Finally we need to extract what leagues the user would like us to draw from. Since this is also a simple natural language analysis job, we can re-use LLM-A with a different primer to return an optimal role-league mapping. The final flow of the application ends up looking like this:


User Input
LLM-A
Is user asking for a team?
yes
no
LLM-A
Is user asking for >2 regions
Is there an existing analysis session?
no
yes
LLM-A
Assign roles, leagues, and pick regions
Error Message
LLM-Team
Pass through to analysis session
Sample 50 teams from league pdfs
LLM-T
Pick best team from sampled teams
Pick best substitutes for selected team
LLM-Team
Create new analysis session, pass in selected team

2 Implementation

The implementation of this system starts in Google Collab, where we wrote initial scripts to analyze small sets of the data and develop a proof of concept. We then take the notebook and bring it over to AWS Glue, adapting the scripts to use pyspark so as to scale to the hundreds of gigabytes of json data provided by the competition. We then save the processed $\vec{p}$ data as a parquet on an S3 instance.

*Side note here: This is how we learned the hard way that Jupyter Notebooks + AWS Glue =

ballooning server costs and that running scripts is MUCH more cost efficient. Even with the notebooks already written we were able to rack up $300 of fees by debugging inside of active notebook sessions before we noticed.

Once we have the parquets, we can bring the analysis back into Google Collab to train models, study the principle components, and generate the pdfs. Finally, using Streamlit we implement the user interaction logic tree and polish up a nice UI to improve presentation.

3 Known Issues

Please see the full write up for the limitations of our product.