How to Itemize in League of Legends

A Data Science Project

Laurent Chen **Alexandre Copin**

Master 1 Android

January 2026

I. Introduction

Video games play a significant role in modern culture. Consequently, this report explores one of the world's most prominent titles, *League of Legends* (LoL). As of 2025, the game boasts an estimated 130 million monthly players and achieved a peak viewership of 6.8 million during esports events.

LoL is a Multiplayer Online Battle Arena (MOBA). In its standard competitive mode, two teams of five players compete to defend their respective bases while attempting to destroy the opposing Nexus. Each player controls a "champion" with unique spells, statistics, and a specific role. Champions are assigned positions across the map's three lanes; for instance, mages typically occupy the "midlane," where they face opposing mages or assassins.

There are six primary classes that define a champion's role and strategic requirements:

- **Tanks:** Prioritize defensive stats and provide crowd control (CC).
- **Fighters:** Balance defensive stats, health regeneration, and offensive power.
- **Mages:** Heavily prioritize Ability Power (AP) to deal burst damage via spells, typically from a distance.
- **Marksmen:** Attack from range, prioritizing Attack Speed, Attack Damage (AD), and Critical Strike Chance.
- **Assassins:** Focus on high offensive stats to quickly eliminate high-value targets (typically Marksmen and Mages).
- **Supports:** Utilize specific items to aid allies, focusing on utility and team survival.

Crucially, the game features two primary damage sources: Physical Damage (scaled by Attack Damage/AD) and Magic Damage (scaled by Ability Power/AP). Effective defensive itemization for Tanks and Fighters requires a strategic balance of Health, Armor (to mitigate Physical Damage), and Magic Resistance (to mitigate Magic Damage).

Throughout a match, champions gain experience (XP) to progress from level 1 to 18, unlocking increased statistics and potent abilities. Simultaneously, players earn gold to construct their "build"—a strategic set of items. Itemization is a core component of the game's complexity. Players may purchase up to six items from a pool of over 100, each offering unique stats and passive effects that fundamentally alter gameplay.

Navigating these choices can be overwhelming. Selecting the correct items depends heavily on the match context, and suboptimal purchases can severely hinder performance. While purchased items can be sold, the return on investment is lower, meaning poor decisions have economic consequences within the match.

Current resources, including websites and in-game applications, typically recommend items based on aggregate popularity and win rates. However, these tools often fail to account for specific team compositions (both allied and opposing). This limitation can mislead players into adhering to static "optimal builds" rather than adapting to the dynamic needs of the specific match.

II. Data Acquisition

Our primary data source is the Riot Games API, which provides detailed match- and player-level information for League of Legends. This API grants access to structured JSON files for each game, typically containing significant amounts of granular data per match. The available information includes:

- Player identifiers and ranked tiers
- Game metadata (match duration, patch version, queue type)

- In-game statistics (kills, deaths, assists, items purchased, gold earned, etc.)
- Event timelines (objectives, kills, item purchases, experience, and gold progression)

Using this official API ensures that the collected data is authentic, reliable, and regularly updated. Our use of the API adheres to the official Riot Developer Terms of Service, which permits academic and analytical use provided private data is not redistributed.

In addition to match data, we also rely on Riot's official static data source, *Data Dragon*. This repository hosts JSON files containing detailed information about champions, items, and various game assets, allowing us to accurately map in-game identifiers to meaningful game attributes.

Data Collection Protocol

Our current data collection pipeline consists of the following steps:

Game Retrieval

We query the Riot API to retrieve match data for specific regions and ranks. The system iterates through player Summoner IDs to request their match history. Each request returns a detailed JSON file corresponding to a single match (approximately 5,000 lines), allowing us to track exactly when and which items a champion purchased throughout the game. Riot's rate limits are strictly observed to prevent API bans. Consequently, the query volume is constrained, and mechanisms are in place to ensure we avoid collecting duplicate matches.

Parsing and Filtering

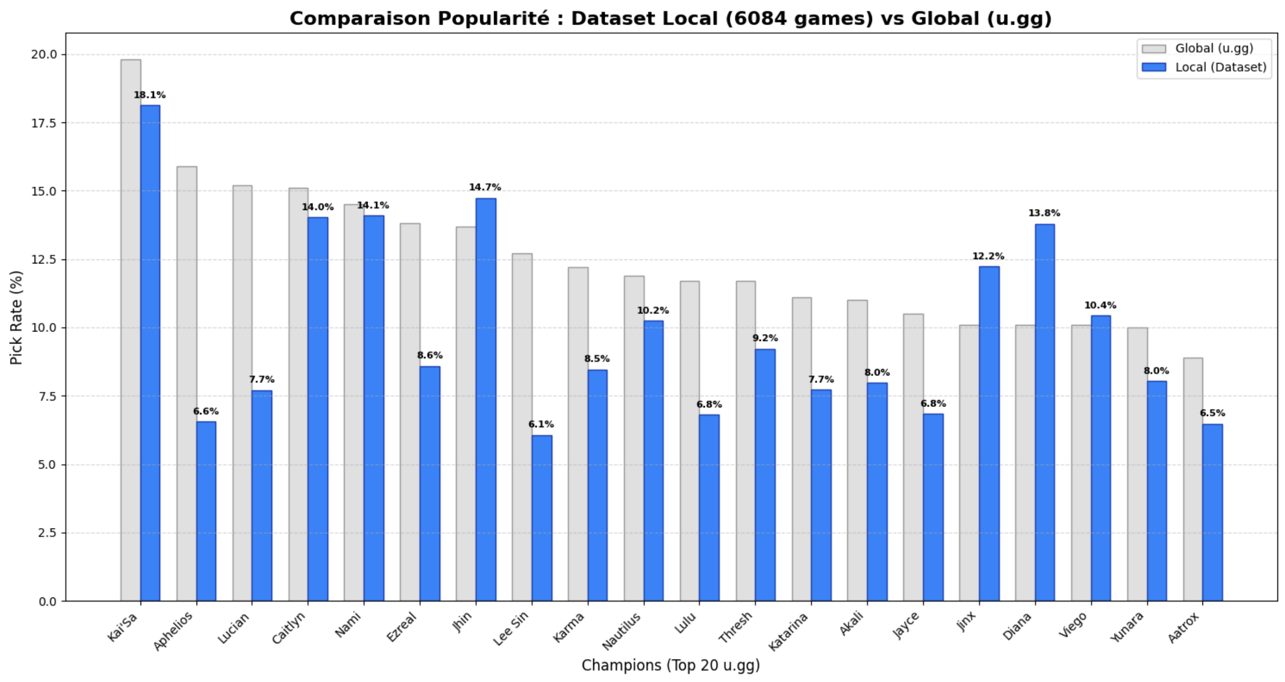
We developed a custom parsing pipeline to transform raw JSON files into structured CSV datasets, retaining only the most relevant features for our analysis (reducing the data to approximately 350 lines per match). The primary variables extracted include timestamped gold values, item purchase events, kill/death events, and team objectives. This data reduction significantly improves efficiency, readability, and usability for downstream analysis.

Data Quality and Biases

Due to the automated nature of the Riot Games API, our dataset possesses a high degree of technical completeness. Unlike survey data or manual entry, there are virtually no missing values or null entries regarding game stats; every item purchase and gold transaction is programmatically recorded.

However, the data is subject to inherent behavioral biases that must be considered:

- **Champion Popularity Bias:** The dataset exhibits a significant class imbalance. Popular champions (e.g., Kai'Sa, Caitlyn) appear with high frequency, providing robust statistical confidence for their itemization patterns. In contrast, "niche" champions are played rarely, resulting in sparse data that may be insufficient for generalizing optimal build paths.
- **Novelty and Beginner Bias:** Newly released champions often suffer from experimental itemization (players guessing what is good) and unstable win rates. Similarly, champions labeled as "beginner-friendly" are often played by less experienced users who may rigidly follow static recommended builds rather than adapting to the specific game context, potentially skewing the "popular" build data away from the "optimal" one.
- **Class Classification Bias:** In the static data provided by riot, they regroup champions by classes except that it is done poorly. For instance, they don't separate attack damage assasins to ability power assasins. The same could be said for some fighters champions. Tanks are composed of the ones able to play for themselves (in role TOP), to the ones that are more of a supports class (in role SUPPORT) and will buy "supports" items. Some champions can also choose to play more AD or AP. To mitigate this, we make sure to include in our dataset the role and the type of damage they deal.



Champion Popularity Distribution in Ranked Matches and in our datasets

Finally, we used different datasets from three ranks (Iron 4, Platinum 4-3 and Diamond 1), representing different levels of skill, to analyze distinctions in player habits.

Feature engineering

Our key numerical data are gold and time, using that we created new variables, namely the gold acceleration, the average gold on purchase (and also the winrate).

- **Gold Acceleration:** We wanted to represent the "snowball" effect, getting an item, could have, so the gold acceleration indicates spikes of gold in the 5 following minutes, indicating if you manage to get gold, easily farm and destroy objectives.
- **AVG gold on purchase:** The average gold on purchase is also interesting because it helps visualize if certain items are bought early on or later in the game.

For further distinction, we also created a separate folder in which we store, for each champion, a file showing the item they bought depending on the opponents faced, the previously purchased item, the gold acceleration it provided, and the win rate percentage. This information will greatly assist us in constructing our dataframe and training a model.

III. Exploratory Data Analysis (EDA)

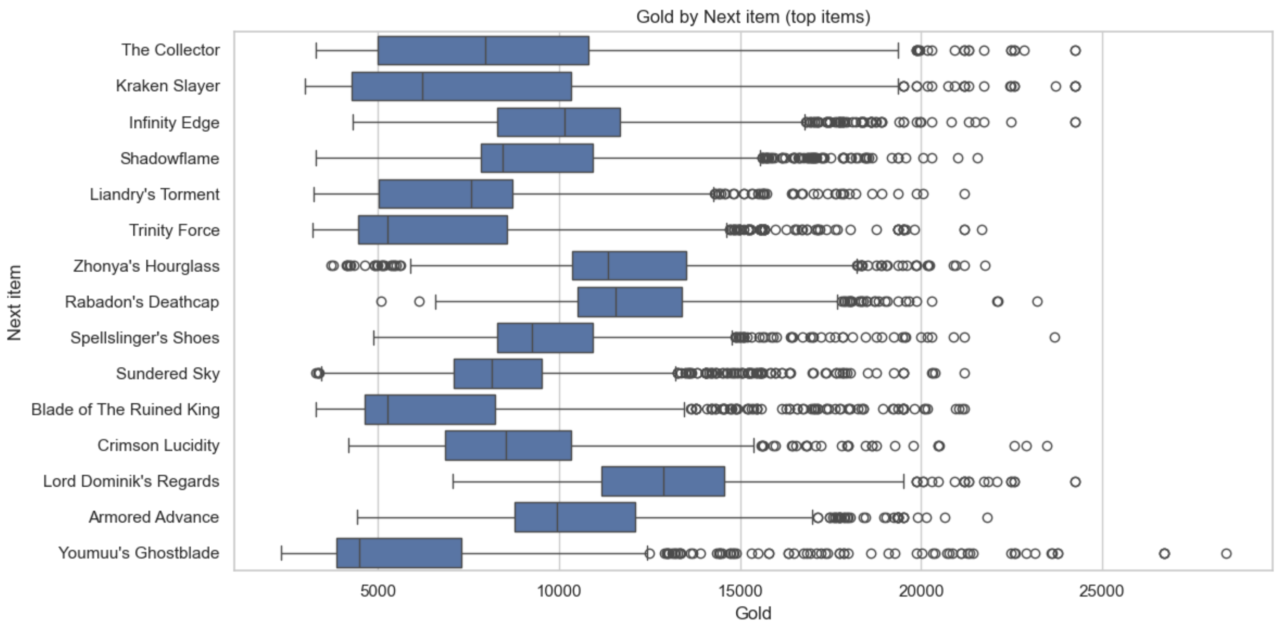
Our exploration around our data is mainly to understand the distribution of the items, and of the champions. The graphic were made using matplotlib and seaborn.

Here's the tail of our dataset to show how it's presented (omitting the opponent's role and damage variables for space) :

ID	Champion	Classe du champion	Dégat du champion	Role	Adversaire	Gold	Item actuel	Next item
46927	Twitch	Marksman	Mixed	BOTTOM	Vayne	7626	Yun Tal Wildarrows	Runaan's Hurricane
46928	Twitch	Marksman	Mixed	BOTTOM	Vayne	11989	Runaan's Hurricane	Infinity Edge
46929	Lulu	Support	AP	UTILITY	Yuumi	3991	None	Ardent Censer
46930	Lulu	Support	AP	UTILITY	Yuumi	5540	Ardent Censer	Crimson Lucidity
46931	Lulu	Support	AP	UTILITY	Yuumi	7390	Crimson Lucidity	Shurelya's Battlesong

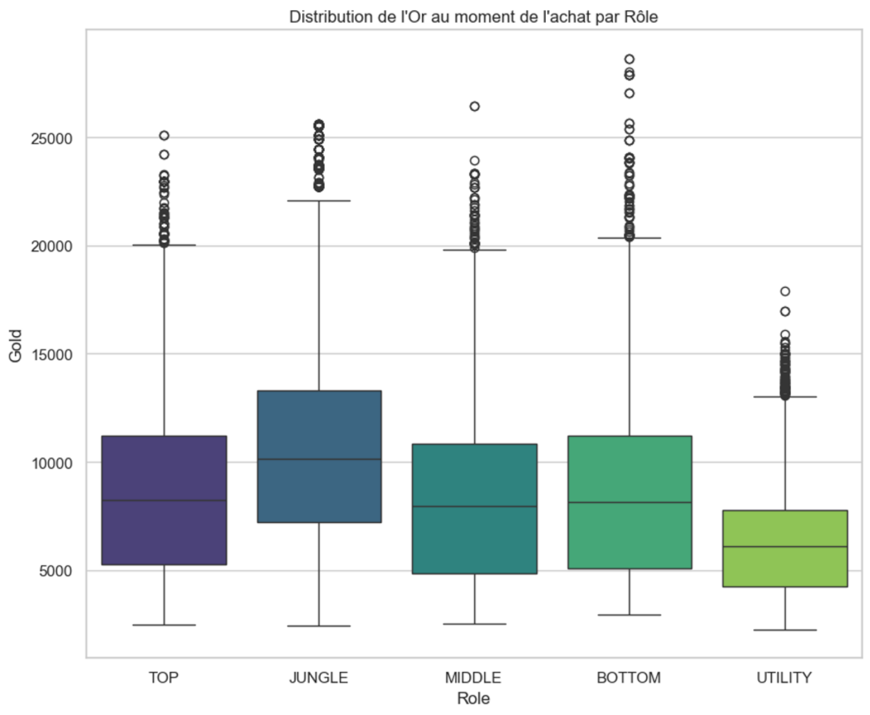
As we can see in the last 3 rows, Lulu bought Ardent Censor -> Crimson Lucidity -> Shurelya's Battlesong

Temporality of items



Boxplot of gold when purchasing items (for top 25 items)

Using gold as a temporal metric,this graph shows what type of items are considered "early", meaning that you get them in first or second, or the more "late" items. The box shows us the usual position, for instance, infinity edge or Rabadon's Deathcap are not first items, while Kraken slayer and Yuumuu's Ghostblade are first buy. It also shows us outliers, typically someone taking an "early items" in the end of the game, or someone taking a "late" item as a first buy.

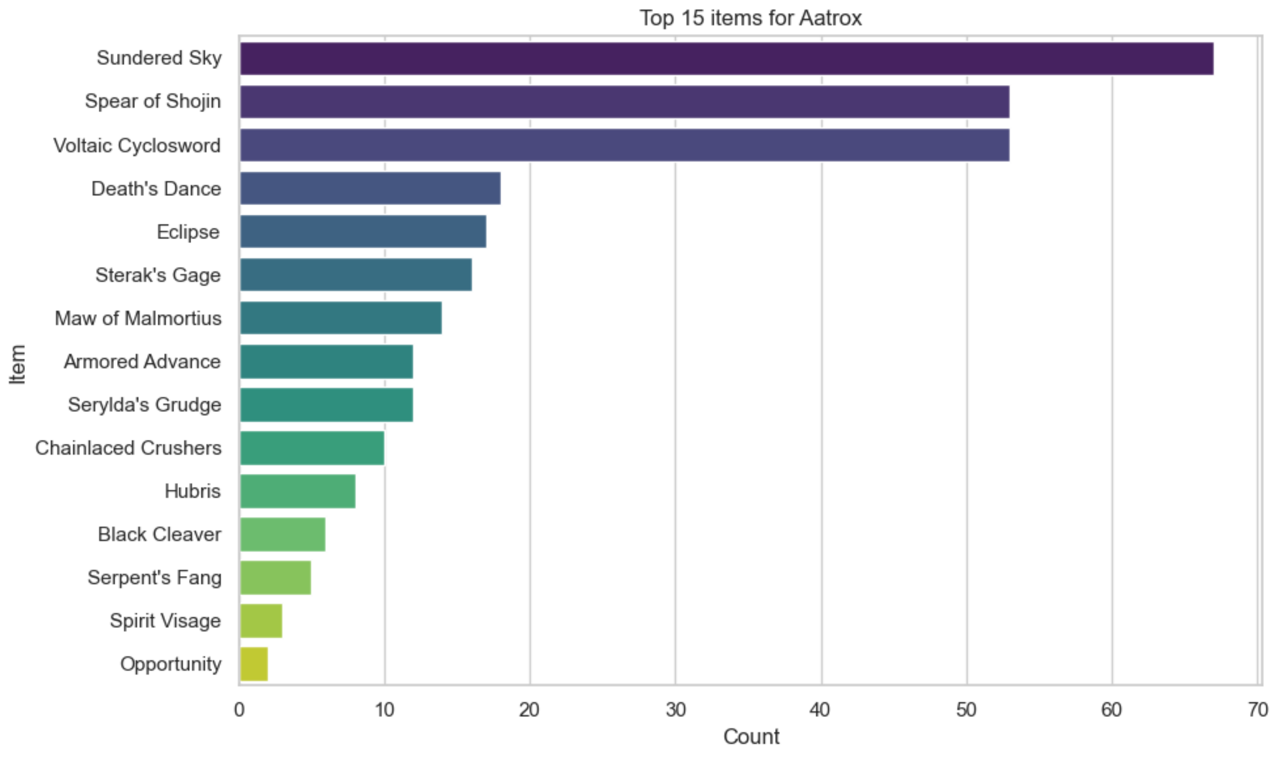


Distribution of Total Gold at Purchase Time by Role

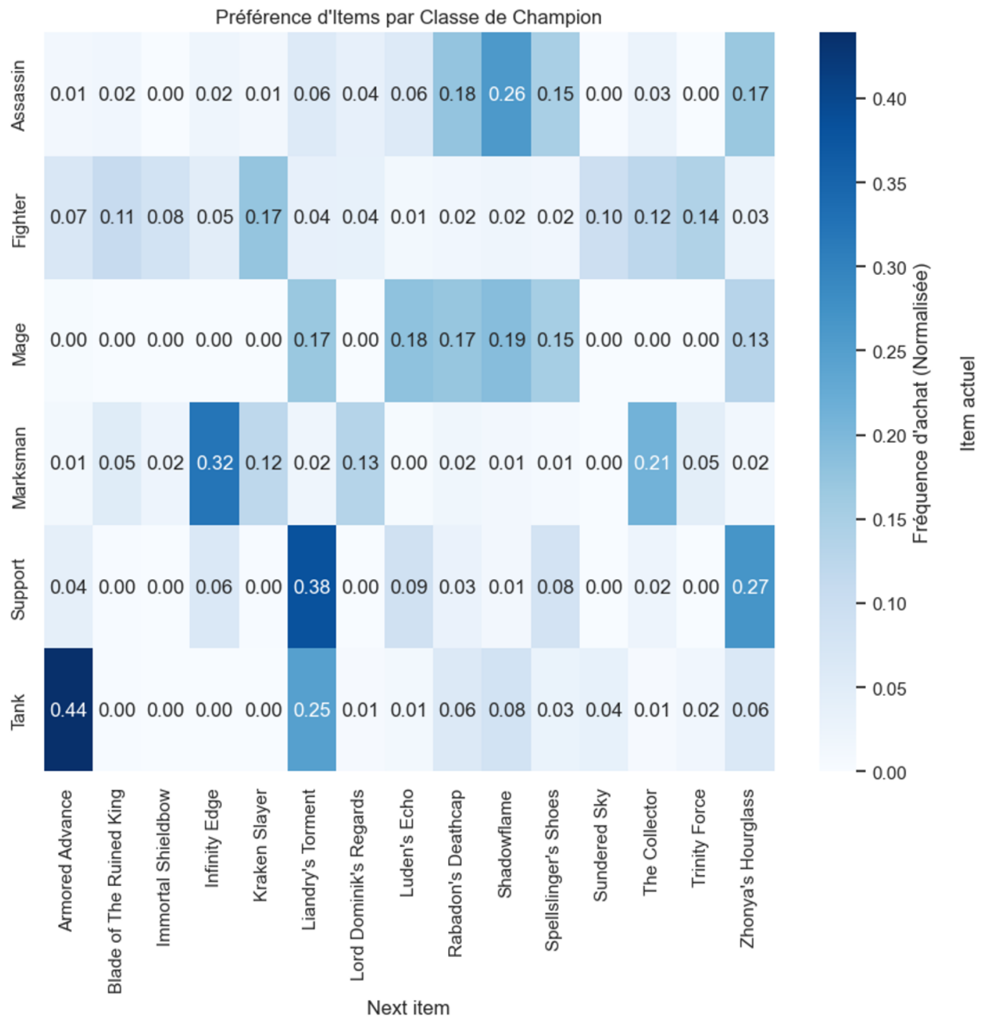
To complement the item analysis, this boxplot (above) illustrates the economic disparity between roles. Supports (UTILITY) operate with significantly lower gold budgets compared to Carries (BOT-

TOM, MID). This confirms that our model must treat the `Gold` variable contextually based on the `Role`. A purchase made at 6000 total gold represents a mid-game decision for a Support but an early-game decision for a Toplaner.

Core items



In this graph, we use the champion Aatrox as a reference to illustrate the pattern of core items. A core item is one that fits a specific champion so well that it becomes a must-buy. The reasoning is that it provides a significant "power spike," helping the character get ahead by killing opponents or taking objectives. For Aatrox, out of nearly 80 games, he almost always purchases Sundered Sky; therefore, we consider this one of his core items. This pattern is typical of champion classes—other fighters also tend to favor Sundered Sky. Similarly, in the previous graph, we saw that Youmuu's Ghostblade is an early item consistently bought by assassins. This highlights an interesting correlation between the champion's class and their itemization.

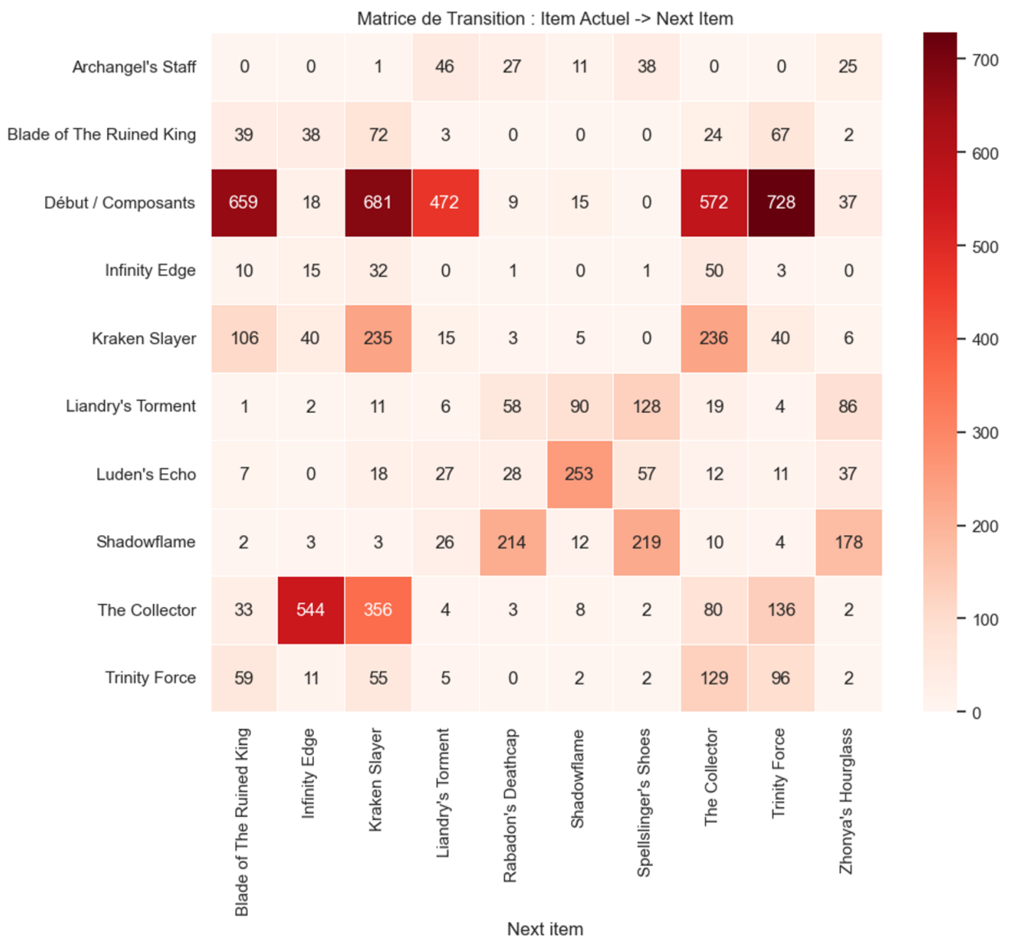


Normalized Heatmap of Item Purchases by Champion Class

Extending this analysis to all champions, the heatmap above reveals strong clusters: Mages almost exclusively buy Ability Power items, while Tanks stick to defensive options (not shown in this smaller heatmap). This strong correlation confirms that `Champion Class` is a powerful predictor for filtering out irrelevant items. Specific patterns align closely with these roles; for instance, Marksmen almost invariably purchase *Infinity Edge*, while Supports frequently opt for *Redemption*. However, broad classifications have limitations. The "Assassin" class, for example, is nuanced because it aggregates both AD and AP champions, who do not share the same item pool. This suggests that while broad classes identify "core" items, specific purchases are often dependent on the sub-class.

Sequential Decision Making

Itemization is inherently a sequence: the choice of the second item is heavily conditioned by the first.

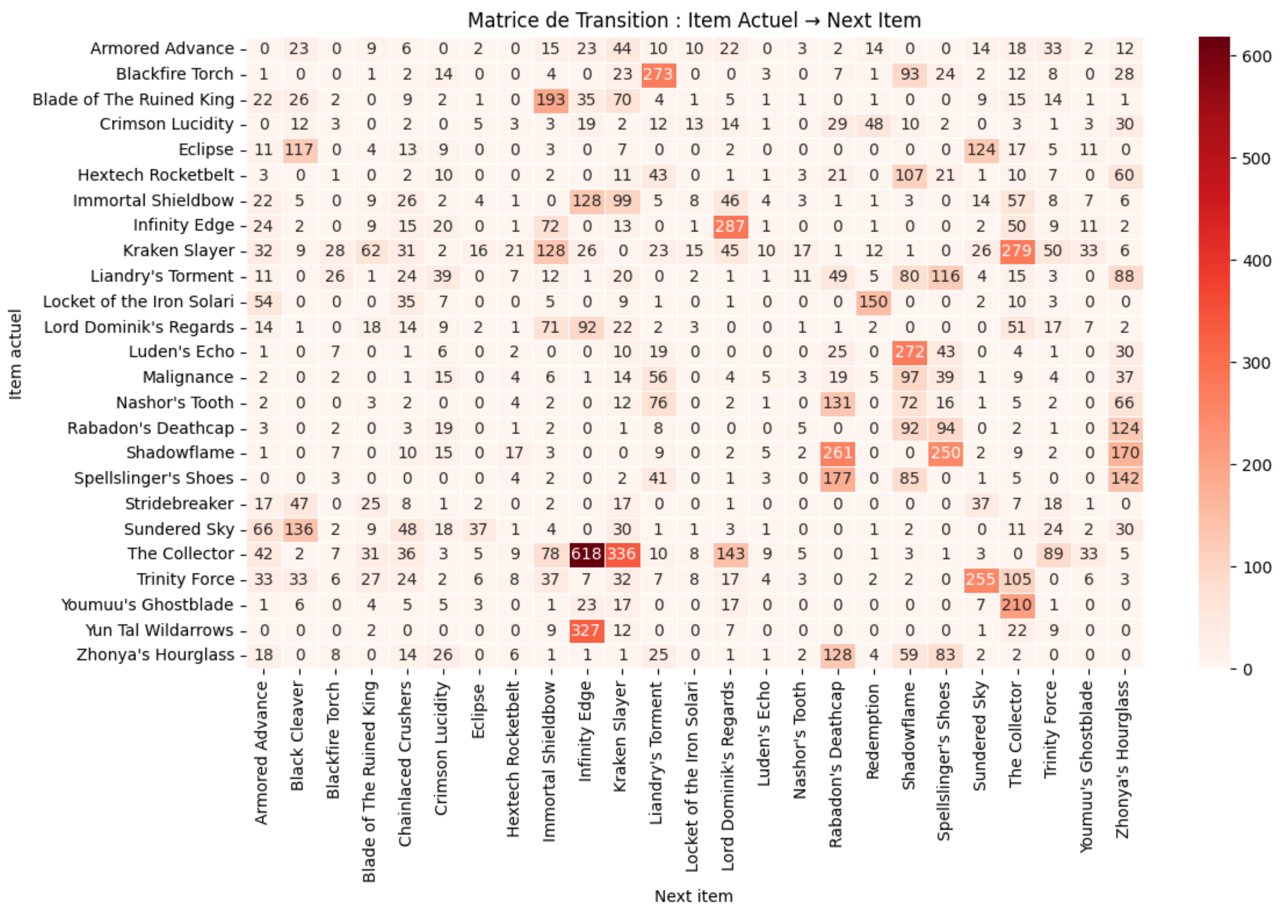


	Blade of The Ruined King	Infinity Edge	Kraken Slayer	Liandry's Torment	Rabadon's Deathcap	Shadowflame	Spellslinger's Shoes	The Collector	Trinity Force	Zhonya's Hourglass
Archangel's Staff	0	0	1	46	27	11	38	0	0	25
Blade of The Ruined King	39	38	72	3	0	0	0	24	67	2
Début / Composants	659	18	681	472	9	15	0	572	728	37
Infinity Edge	10	15	32	0	1	0	1	50	3	0
Kraken Slayer	106	40	235	15	3	5	0	236	40	6
Liandry's Torment	1	2	11	6	58	90	128	19	4	86
Luden's Echo	7	0	18	27	28	253	57	12	11	37
Shadowflame	2	3	3	26	214	12	219	10	4	178
The Collector	33	544	356	4	3	8	2	80	136	2
Trinity Force	59	11	55	5	0	2	2	129	96	2

Item Transition Matrix (Current Item → Next Item)

The heatmap above highlights this dependency. The red "hotspots" indicate popular build paths, while the **"Début / Composants" row** specifically represents the **First Item** decision. This strong sequential correlation justifies using a model that treats the `Current Item` (history) as a key input to predict the `Next Item`.

To analyze this further, we generated a transition matrix for the top 25 most popular items (below).



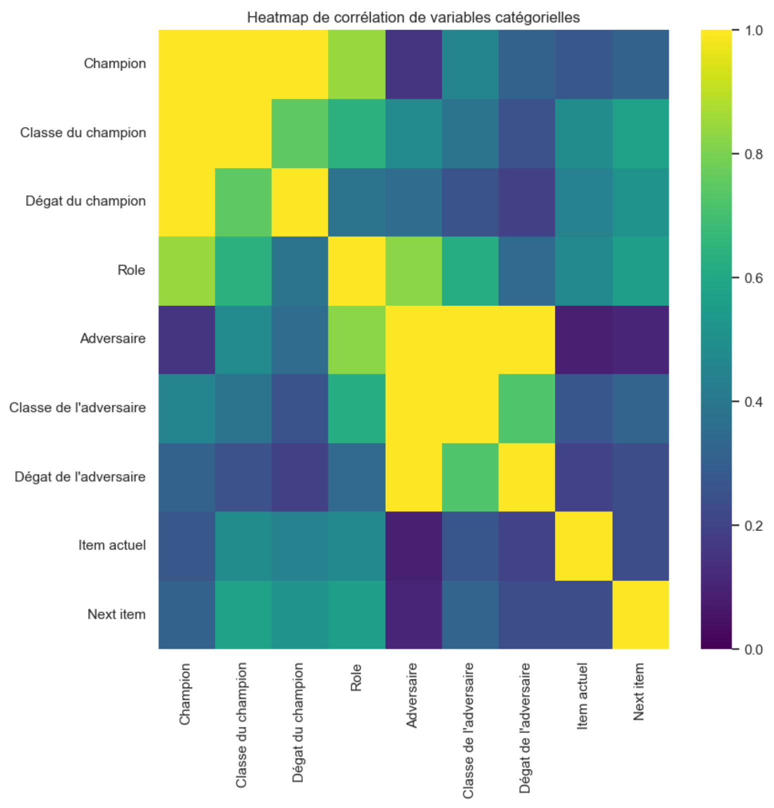
	Armored Advance	Black Cleaver	Blackfire Torch	Blade of The Ruined King	Chainlaced Crushers	Crimson Lucidity	Eclipse	Hextech Rocketbelt	Immortal Shieldbow	Infinity Edge	Kraken Slayer	Liandry's Torment	Locket of the Iron Solari	Lord Dominik's Regards	Luden's Echo	Nashor's Tooth	Rabadon's Deathcap	Redemption	Shadowflame	Spellslinger's Shoes	Sundered Sky	The Collector	Trinity Force	Youmuu's Ghostblade	Zhonya's Hourglass
Armored Advance	0	23	0	9	6	0	2	0	15	23	44	10	10	22	0	3	2	14	0	0	14	18	33	2	12
Blackfire Torch	1	0	0	1	2	14	0	0	4	0	23	273	0	0	3	0	7	1	93	24	2	12	8	0	28
Blade of The Ruined King	22	26	2	0	9	2	1	0	193	35	70	4	1	5	1	1	0	1	0	0	9	15	14	1	1
Crimson Lucidity	0	12	3	0	2	0	5	3	3	19	2	12	13	14	1	0	29	48	10	2	0	3	1	3	30
Eclipse	11	117	0	4	13	9	0	0	3	0	7	0	0	2	0	0	0	0	0	0	124	17	5	11	0
Hextech Rocketbelt	3	0	1	0	2	10	0	0	2	0	11	43	0	1	1	3	21	0	107	21	1	10	7	0	60
Immortal Shieldbow	22	5	0	9	26	2	4	1	0	128	99	5	8	46	4	3	1	1	3	0	14	57	8	7	6
Infinity Edge	24	2	0	9	15	20	0	1	72	0	13	0	1	287	1	0	0	1	0	0	2	50	9	11	2
Kraken Slayer	32	9	28	62	31	2	16	21	128	26	0	23	15	45	10	17	1	12	1	0	26	279	50	33	6
Liandry's Torment	11	0	26	1	24	39	0	7	12	1	20	0	2	1	1	11	49	5	80	116	4	15	3	0	88
Locket of the Iron Solari	54	0	0	0	35	7	0	0	5	0	9	1	0	1	0	0	0	150	0	0	2	10	3	0	0
Lord Dominik's Regards	14	1	0	18	14	9	2	1	71	92	22	2	3	0	0	1	1	2	0	0	0	51	17	7	2
Luden's Echo	1	0	7	0	1	6	0	2	0	0	10	19	0	0	0	0	25	0	272	43	0	4	1	0	30
Malignance	2	0	2	0	1	15	0	4	6	1	14	56	0	4	5	3	19	5	97	39	1	9	4	0	37
Nashor's Tooth	2	0	0	3	2	0	0	4	2	0	12	76	0	2	1	0	131	0	72	16	1	5	2	0	66
Rabadon's Deathcap	3	0	2	0	3	19	0	1	2	0	1	8	0	0	0	5	0	0	92	94	0	2	1	0	124
Shadowflame	1	0	7	0	10	15	0	17	3	0	0	9	0	2	5	2	261	0	0	250	2	9	2	0	170
Spellslinger's Shoes	0	0	3	0	0	0	0	4	2	0	2	41	0	1	3	0	177	0	85	0	1	5	0	0	142
Stridebreaker	17	47	0	25	8	1	2	0	2	0	17	0	0	1	0	0	0	0	0	0	37	7	18	1	0
Sundered Sky	66	136	2	9	48	18	37	1	4	0	30	1	1	3	1	0	0	1	2	0	0	11	24	2	30
The Collector	42	2	7	31	36	3	5	9	78	618	336	10	8	143	9	5	0	1	3	1	3	0	89	33	5
Trinity Force	33	33	6	27	24	2	6	8	37	7	32	7	8	17	4	3	0	2	2	0	255	105	0	6	3
Youmuu's Ghostblade	1	6	0	4	5	5	3	0	1	23	17	0	0	17	0	0	0	0	0	0	7	210	1	0	0
Yun Tal Wildarrows	0	0	0	2	0	0	0	0	9	327	12	0	0	7	0	0	0	0	0	0	1	22	9	0	0
Zhonya's Hourglass	18	0	8	0	14	26	0	6	1	1	1	25	0	1	1	2	128	4	59	83	2	2	0	0	0

Transition Matrix of Top 25 Items

This matrix structures player behavior by capturing how frequently a specific item (rows) leads to another purchase (columns). It reveals notable **asymmetries**: some items appear frequently as a starting point but rarely as a follow-up. Conversely, items that receive high transition counts from many different sources tend to function as universal power spikes. Furthermore, the clustering of these transitions allows us to implicitly identify champion sub-classes based purely on their distinct build paths.

Global Correlations : Heatmap of categorical variables

Our dataset contains mostly categorial variables so it was harder to show the correlation between variables but still manageable with scipy using cramer's V :



Naturally, the correlation between a champion and their role/damage type is 1 (based on static data); however, the analysis reveals several interesting findings:

- **Role Flexibility:** The role/champion correlation indicates that certain champions are played across multiple roles.
- **Itemization:** The next item bought depends heavily on the champion's class, making it a crucial feature for our machine learning model. The opponent's class also shows some correlation, reflecting reactive itemization later in the game (e.g., building magic resistance against a mage). However, there is less correlation with the opponent's damage type; since this is a binary variable, it often provides limited differentiation.
- **Counter-Picking:** The Champion/Opponent correlation is low, suggesting that "counter-picking" (selecting a champion specifically to counter the opponent) is not a prevalent strategy in this dataset.

The analysis shows that the Next Item is dependent on the champion's class, which is an important factor for our machine learning model. The opponent's class shows some correlation too, while the Champion/Opponent correlation is low, meaning that "counter-picking" strategy is not heavily present in the dataset.

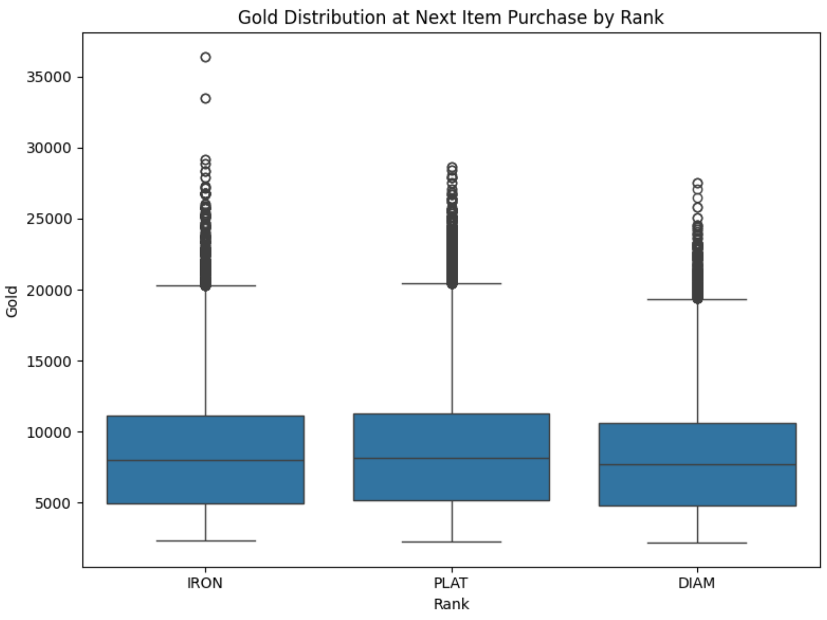
Conclusion of EDA: The exploratory analysis demonstrates that item choice is driven by distinct factors: the **opponent** (counter-building), the **economy** (role/gold), the **class** (identity), and the **sequence** (build path). These dimensions will serve as the input features for our Random Forest classifier.

Comparison at different ranks

In this section, we will use graph to compare our datasets from different ranks. We have a dataset in iron constituted of 2 thousands games,the same size as our diamond dataset, and a larger one from platinum games, with 6 thousands games (The graphs shown earlier were from this dataset), The goal is to show the differences in habits and optimization.

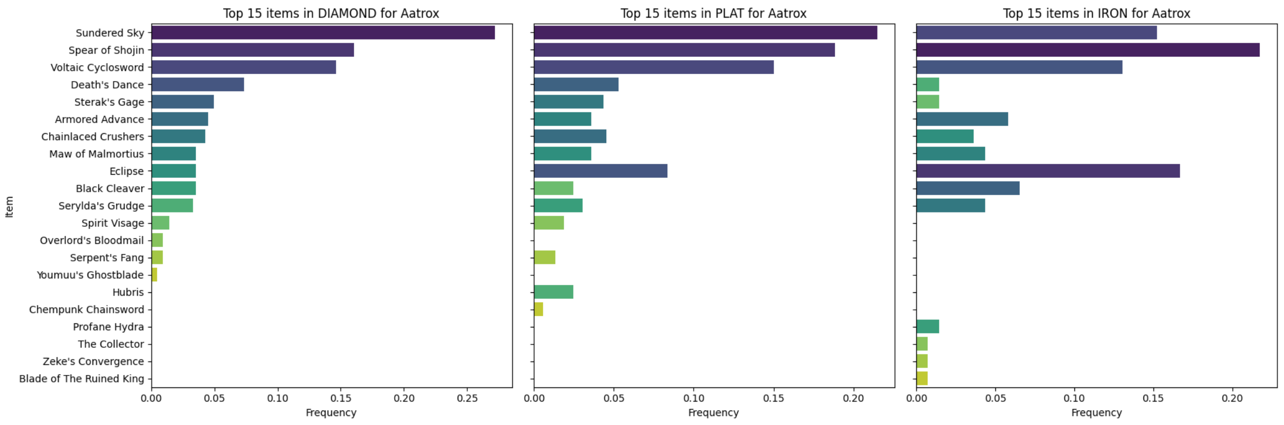
Game speed

An important aspect to note in higher ranks, is that players tend to know a lot better how to use their advantage to finish faster the games. It means that they won't usually be full build as the game ends before it. It also means that they know much better which item gives the better power spike.

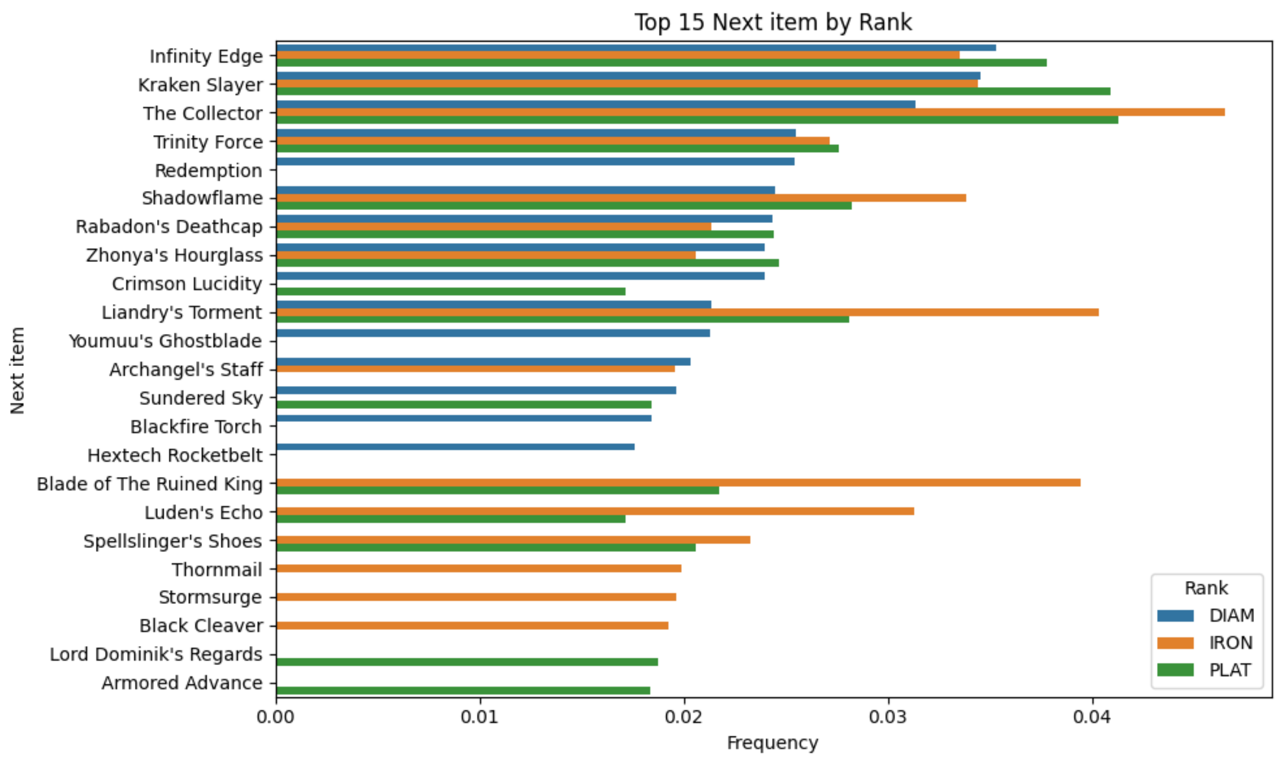


Building habits

Now we can look back at the top items 15 for Aatrox for comparison :



As we can see, in diamond, they take primarily what is seen as the core items, Sundered Sky, and the frequency of other items is lower, yet still present as optional on the context. If we look in iron, their most frequent item is the Spear of Shojin, that while still popular in diamond and plat, not more than Sundered Sky, we can also see a high frequency for Eclipse, but below 10% in plat, and below 5% in diamond. So depending on which dataset we train our model, we could see one that would be overfit on core items, but another one that could be more flexible while not necessary right.



If we look at the top 15 most popular items across ranks, we can see some that appear in all 3, but some are popular only on certain ranks, like hextech protobelt and redemption being built a lot in diamond while black cleaver is more popular in iron so we can tell that the value players give to a certain item is not the same across ranks.

IV. Predictive Model for Item Recommendation

Choosing the optimal sequence of item purchases in League of Legends depends on many factors, including the champion, its role, current gold, and the enemy team composition. The model will learn from historical match data that we transformed into a dataframe, to try and capture common build sequences as well as context-dependent variations.

KNN Model

Following the insights from the EDA, we developed a recommendation system based on a **K-Nearest Neighbors (KNN)** approach. We chose KNN because it mimics how human players learn: by observing what successful players build in similar situations. We also noticed how items patterns tend to be in the same categories.

Data Preparation & Features

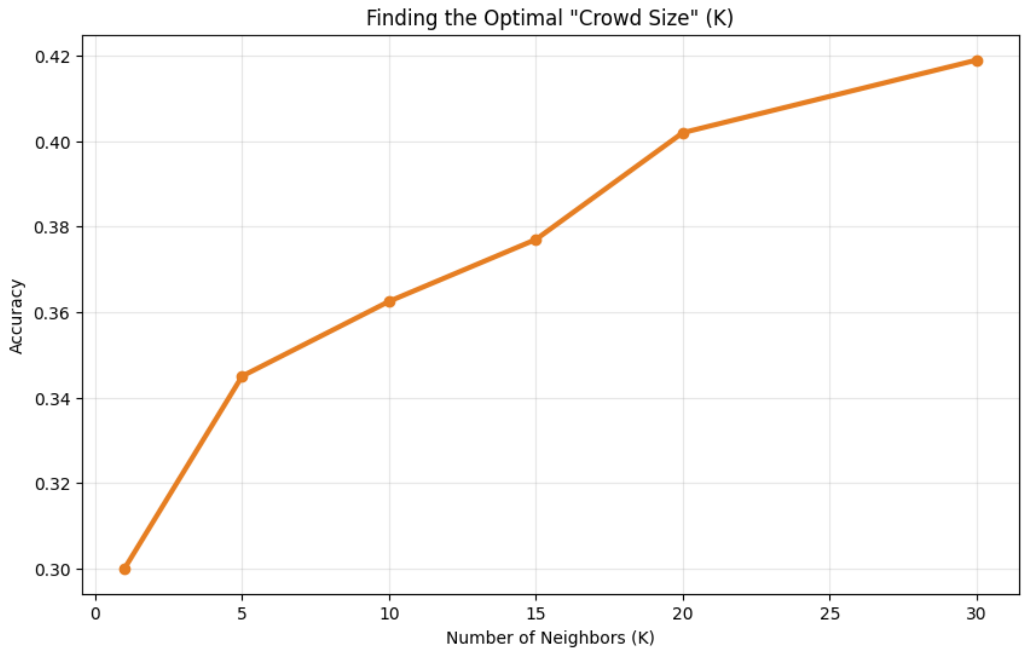
To convert a League of Legends match into a mathematical vector compatible with distance metrics, we performed specific feature engineering:

- **Dataset Scale:** We processed a total of **6,084 matches**, extracting individual decision points (every time a player buys an item). This resulted in a massive dataset of **179,677 samples**.
- **Item Filtering:** To reduce noise, we filtered the item list to focus on **107 Legendary Items**, removing small components (like Long Swords or Ruby Crystals) that don't represent strategic decisions.

- **Feature Engineering (Multi-Hot):** The most complex variable is the player's inventory (a variable list of items). We used a **Multi-Label Binarizer** to encode this into a dense vector. Combined with One-Hot encoding for Champion, Role, and Opponent Class, our final feature vector has a size of **449 dimensions**.

Hyperparameter Tuning: Finding the "Crowd Size"

The most critical hyperparameter in KNN is K (the number of neighbors). If K is too small, the model is unstable (copying a single specific game). If K is too large, recommendations become generic. We used the Elbow Method to find the optimal balance.



Finding the Optimal "Crowd Size" (K): Balancing Bias and Variance

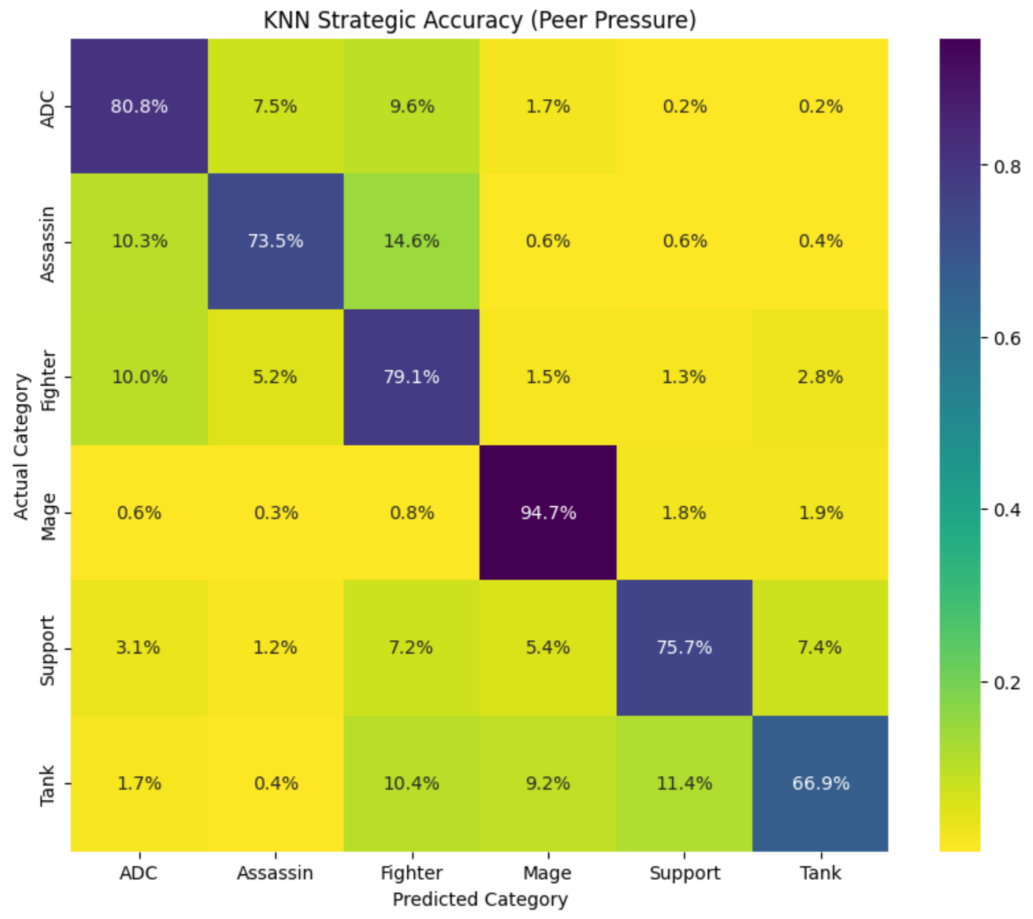
As shown in the graph above, we observed that accuracy stabilizes as the neighborhood grows. We selected $K = 30$ for our final model. This ensures that every recommendation is backed by a "consensus" of at least 30 similar high-level matches.

Results & Evaluation

We evaluated the model's performance using two distinct approaches: raw accuracy and utility.

Strategic Accuracy (Peer Pressure)

The Top-1 Accuracy of our model is **42.2%**. This means that in 42.2% of cases, the model successfully predicts the *exact* item the human player chose. To visualize the errors, we plotted a confusion matrix of item classes.

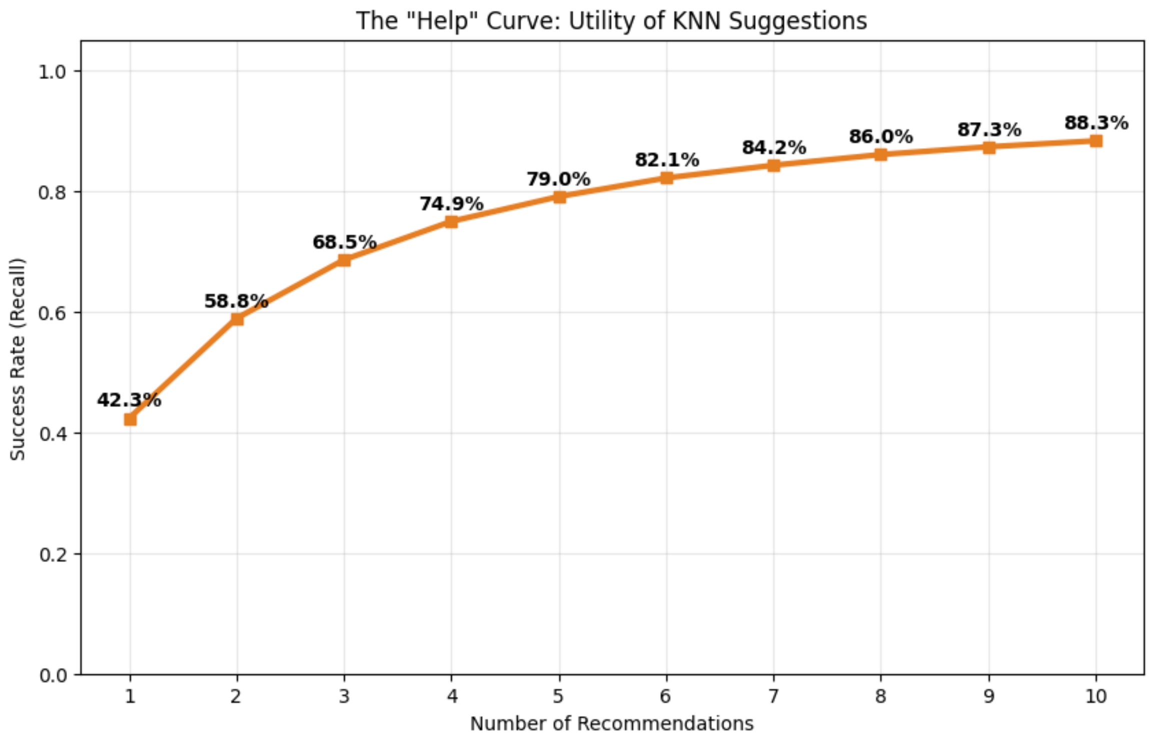


KNN Strategic Accuracy (Peer Pressure): Confusion Matrix of Item Classes

This matrix reveals the model's "strategic reasoning": even when it is "wrong" (predicting Item A instead of Item B), it rarely crosses class boundaries. For example, it almost never recommends a Tank item to an ADC.

The "Help Curve"

In a recommendation system, Top-1 accuracy is not the only metric that matters. It is more important that the "correct" item is *somewhere* in the suggestions. We created the **"Help Curve"** to visualize this.



The "Help" Curve: Utility of KNN Suggestions

The curve demonstrates the practical power of the tool:

- **Top-1 Accuracy:** 42.2%
- **Top-5 Accuracy:** $>$ **79%**

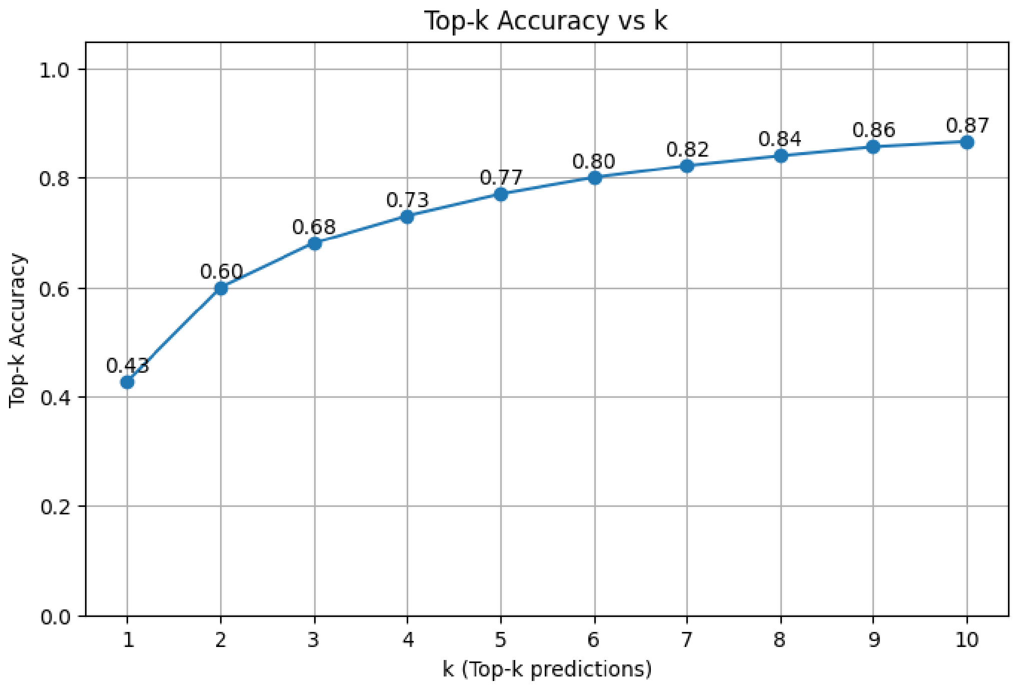
This confirms that our KNN model is an effective filter. It successfully reduces the shop's 100+ options to a relevant shortlist of 5 items, containing the optimal choice 79% of the time, effectively solving the "choice paralysis" problem for players.

Random Forrest Classifier Model

We also tried to implement a well known and solid model, the Random Forrest classifier. Because there is a sequence, depending on the context, the idea to use a decision tree on a large scale makes sense, depending on who you play, you want to get to your core item, and then depending on the opponent you adapt the rest of your build. For the features, this model did not use a full player's inventory, but only the last item (None if it is the first being bought). The model was configured with the following key hyperparameters:

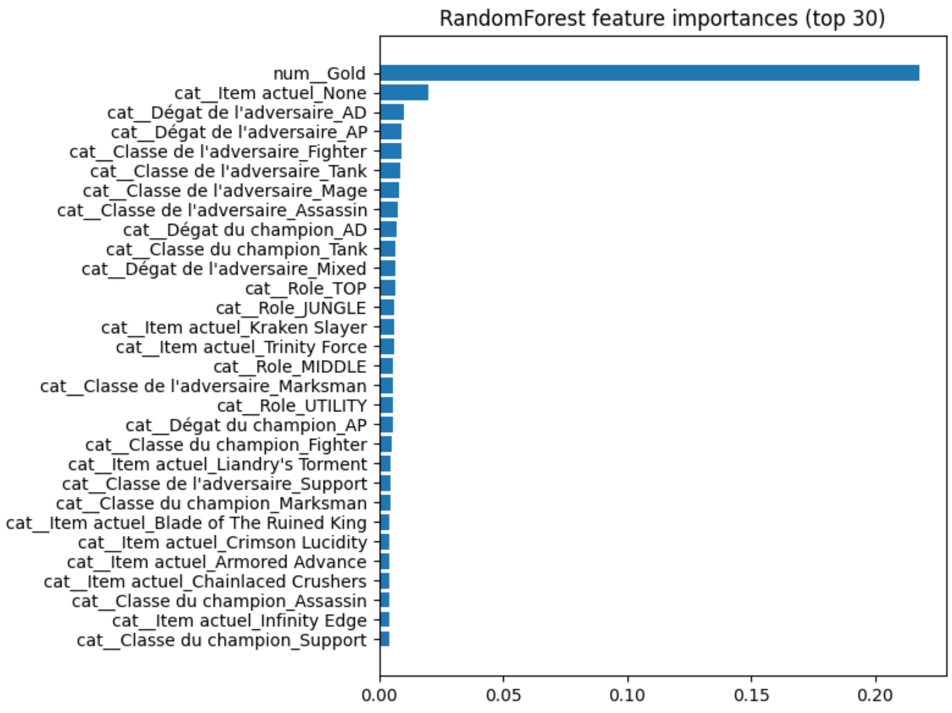
- `n_estimators`: 200
- `min_samples_split`: 10
- `oob_score`: True (for out-of-bag error estimation)
- `random_state`: 42 (for reproducibility)
- `n_jobs`: -1 (to utilize all available processor cores)

We did not specify the min samples leaf parameters or the max depth. Using a column Transformer and OneHotEncoder, we ended up with 480 transformed Features.



Results & Evaluation

We can see that we have fairly similar accuracy as the KNN, while not using a Multi-label binarized but a simple categorical variable. This model has the benefits of giving us the same results, while also being less expensive on spatial memory.



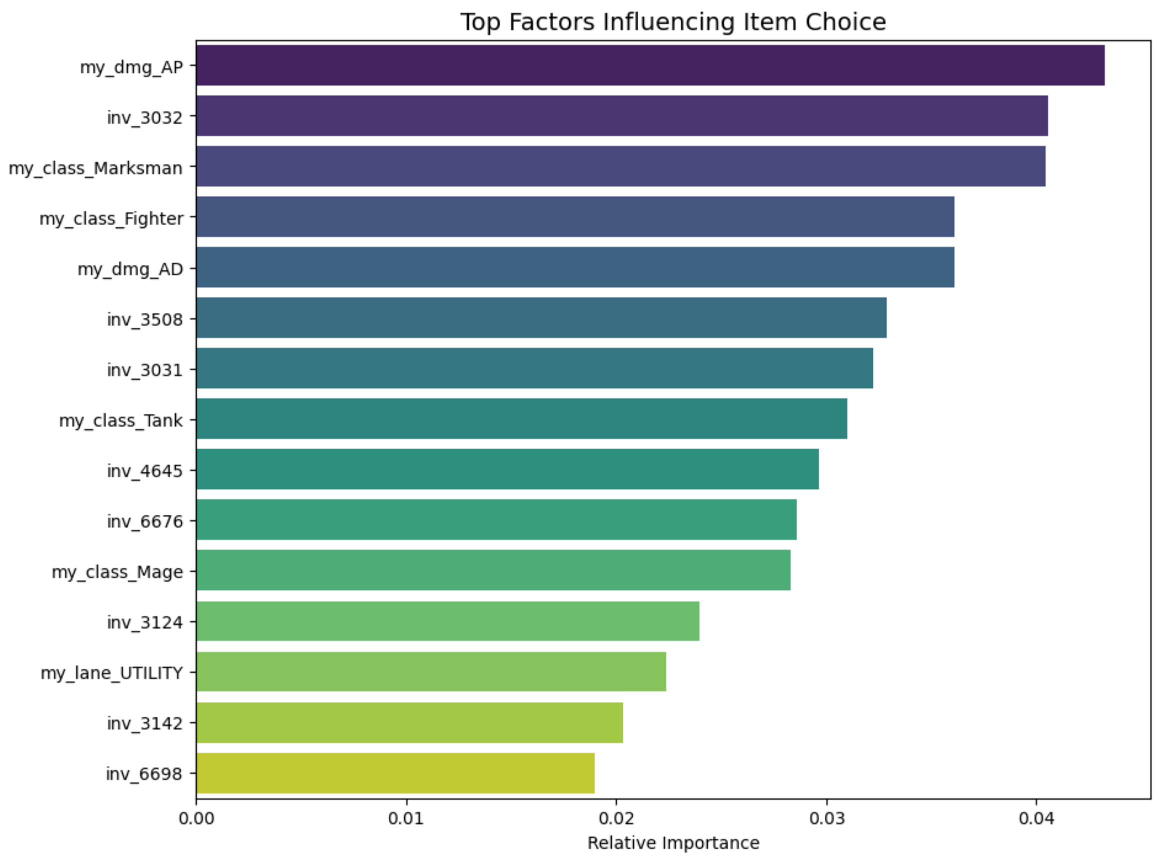
On the most important features of the randomForest, the first one is the number of gold (when can you buy an early or late item), then if you don't have an item (more probably going to rush your "core item"). Interestingly, then comes information on your opponent,namely his damage type and his class, showing that there is indeed some contextualization happening behind the scenes.

V. Advanced Modeling: XGBoost Gradient Boosting

While the KNN algorithm provided a strong baseline by mimicking "peer pressure" (what others do), we sought to improve performance using a gradient boosting framework. XGBoost (Extreme Gradient Boosting) allows us to capture non-linear relationships and, crucially, understand *why* a recommendation is made by analyzing feature importance.

Feature Importance and Decision Drivers

Unlike the "black box" nature of some models, XGBoost allows us to visualize which features drive the decision-making process.

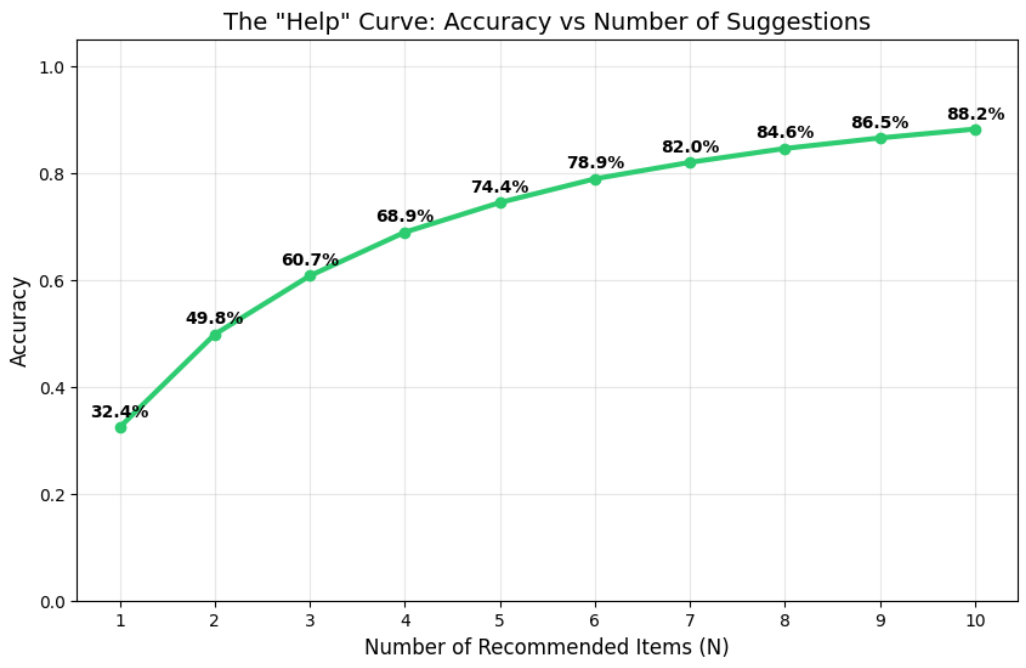


Feature Importance: What drives the purchase?

As seen in the influence plot above, the model does not treat all data equally. It has learned that the **Role** and the **Champion Identity** are the foundational constraints of a build. However, interesting dynamic features like **Gold Acceleration** and specific item history (e.g., *DoransBlade*) play a significant role in fine-tuning the prediction. This confirms that the model is not just memorizing static builds but adapting to the game state.

Model Performance and The "Help Curve"

To evaluate the XGBoost model, we used the same "Help Curve" metric as the KNN model to measure utility.



XGBoost Help Curve: Top-N Accuracy

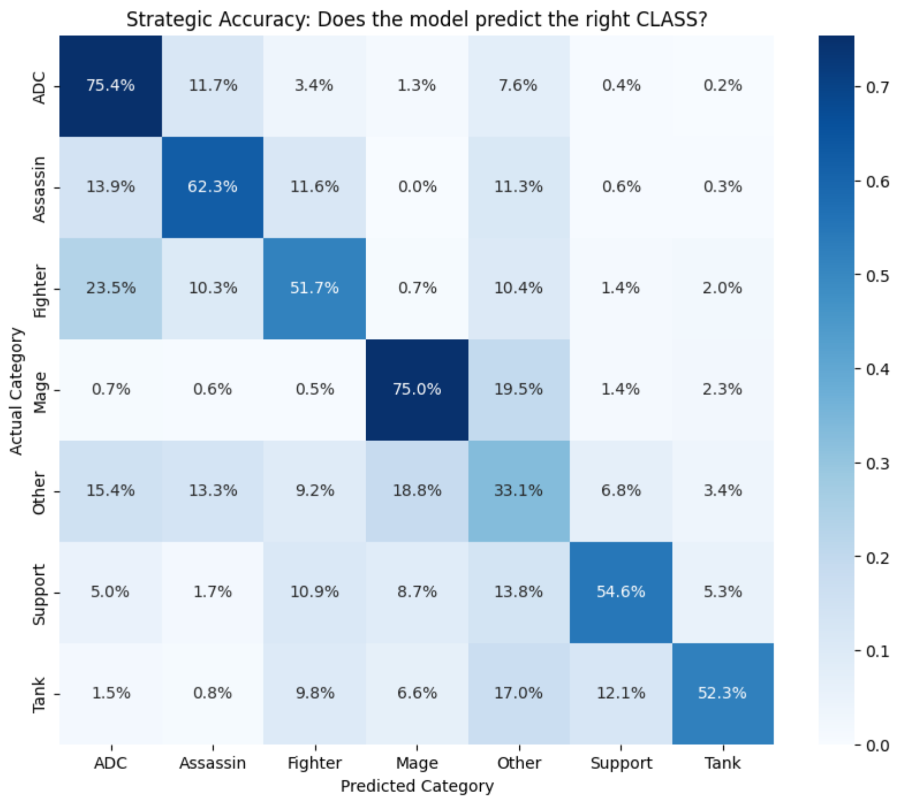
The XGBoost model shows a robust performance:

- **Top-1 Accuracy:** It correctly predicts the exact next item in roughly **43%** of cases.
- **Top-5 Utility:** The correct item appears in the top 5 suggestions nearly **75%** of the time.

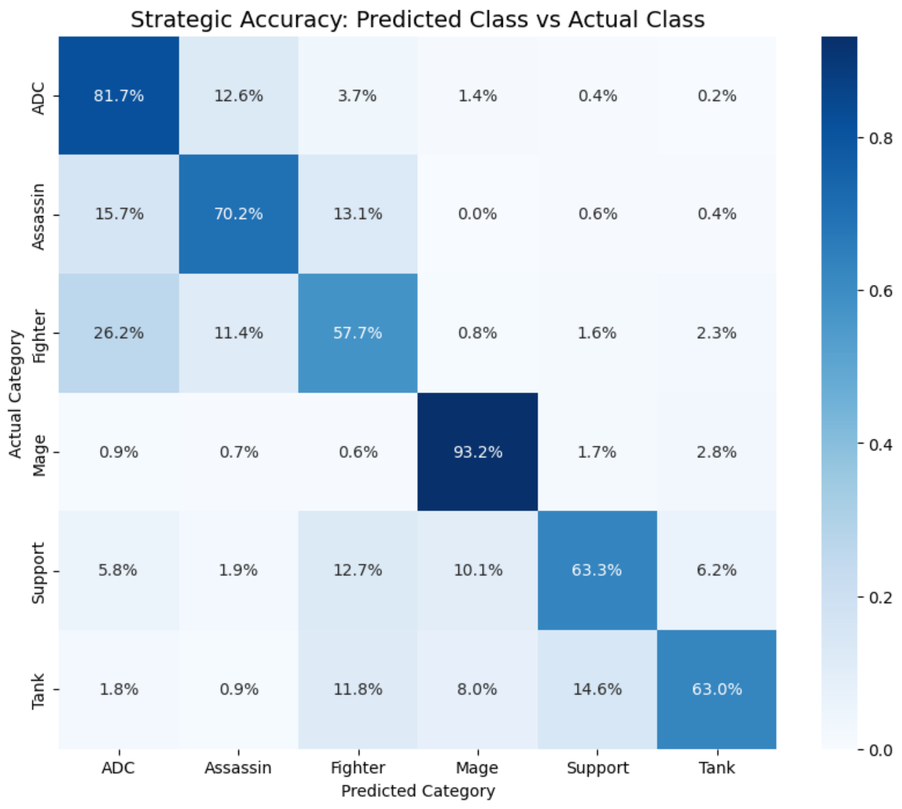
This curve demonstrates that the model is a viable recommender system, effectively narrowing down the shop's vast catalog to a handful of relevant options.

Strategic Accuracy and Class Confusion

Beyond raw accuracy, we analyzed whether the model understands the "strategy" of the game. Even if the model predicts the wrong specific item, does it at least predict the right *type* of item (e.g., predicting a Tank item for a Tank)?



Strategic Accuracy: Prediction counts by Item Class



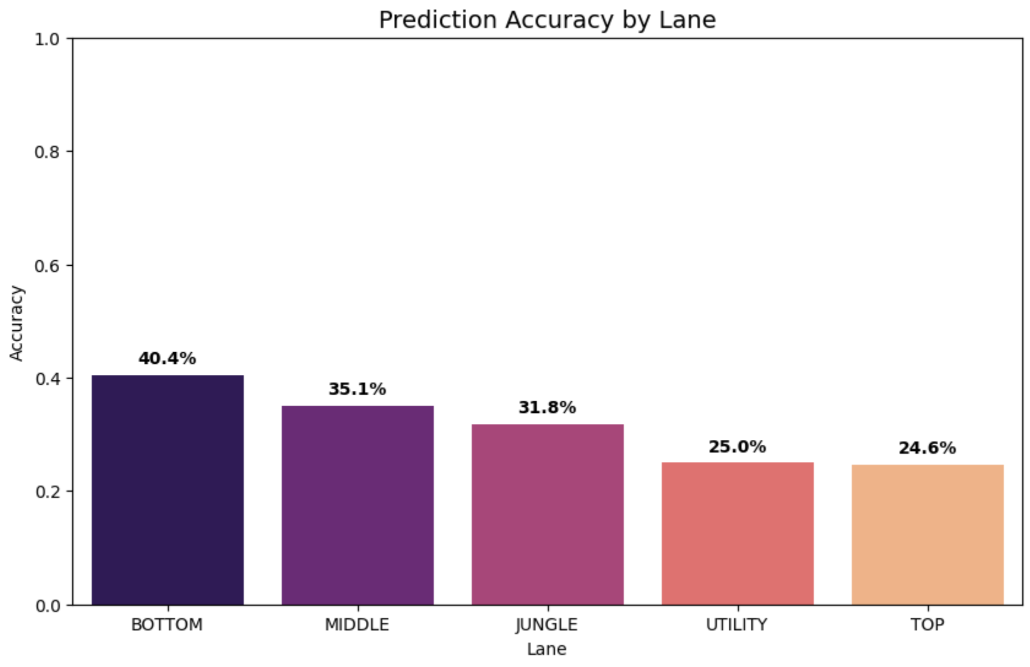
Predicted vs Actual Heatmap: Where does the model get confused?

The confusion matrices above reveal that the model is highly strategically sound. The diagonal density indicates high accuracy. More importantly, the errors are logical: the model might confuse two different *Marksman* items, but it almost never suggests a *Mage* item to a *Fighter*. This implies the model has successfully learned the distinct archetypes of League of Legends champions.

Contextual Analysis: When does the model fail?

To understand the limits of our system, we broke down the accuracy by various game contexts.

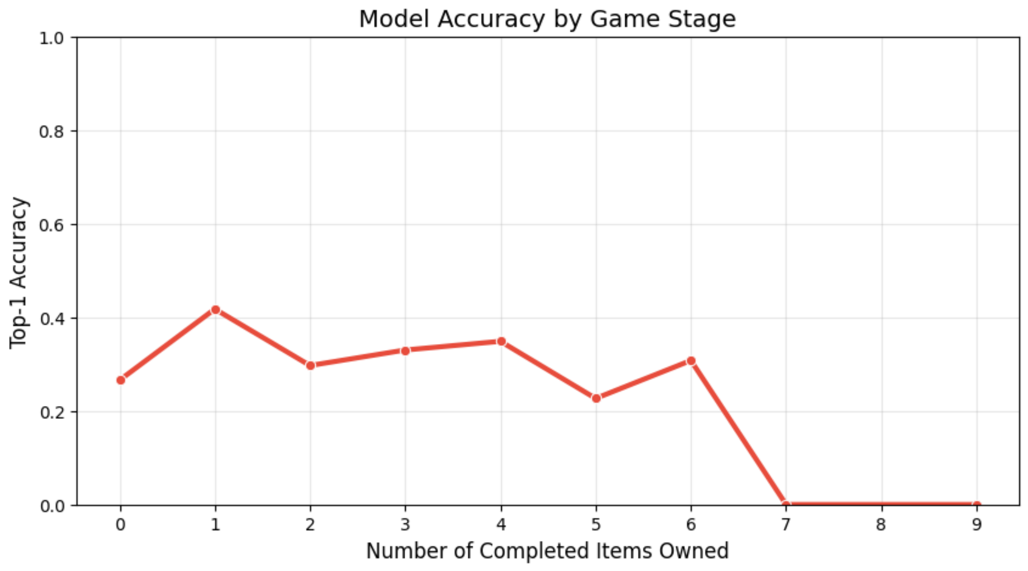
1. Accuracy by Role



Model Accuracy per Role (Lane)

As shown above, **ADC (Bottom)** and **Support (Utility)** are the most predictable roles. This aligns with game theory: Marksmen have rigid mathematical builds (optimizing DPS), whereas Top laners and Junglers must adapt more frequently to be tanky or aggressive, leading to higher variance and slightly lower model accuracy.

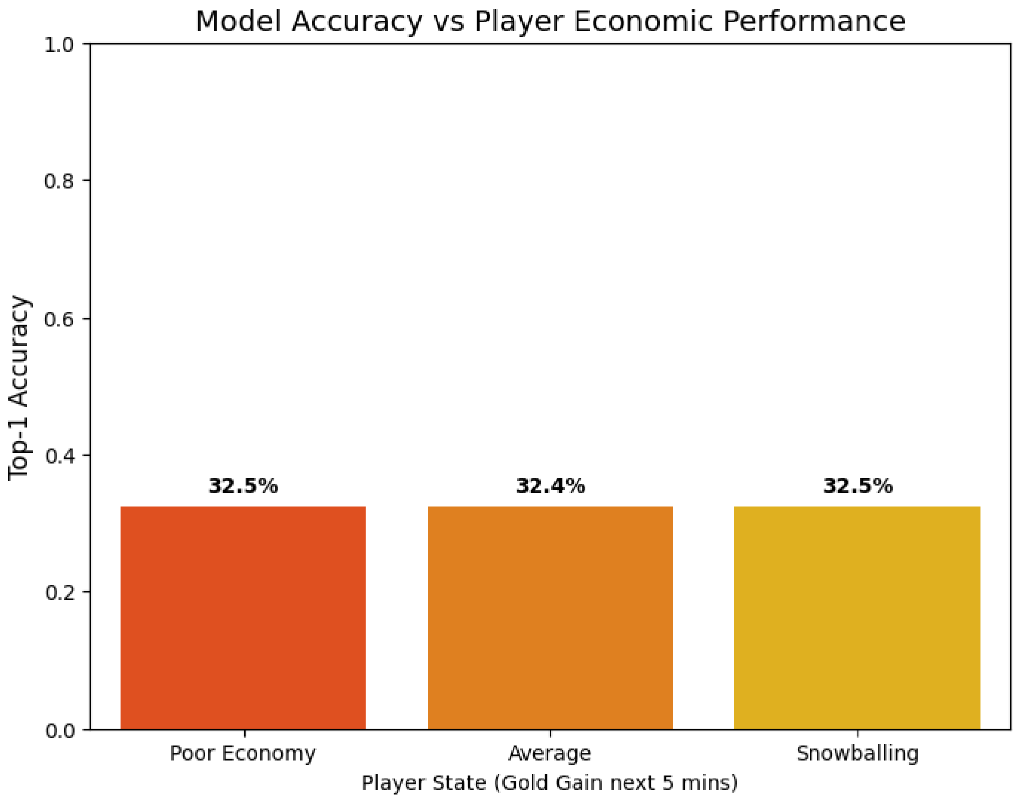
2. Accuracy over Time



Prediction Accuracy Evolution by Game Stage

The model is most accurate in the **Early Game** (First 1-2 items). As the match duration extends, the number of viable strategic options explodes (situational defensive items, unique counters), making prediction significantly harder.

3. Accuracy vs. Player Economy

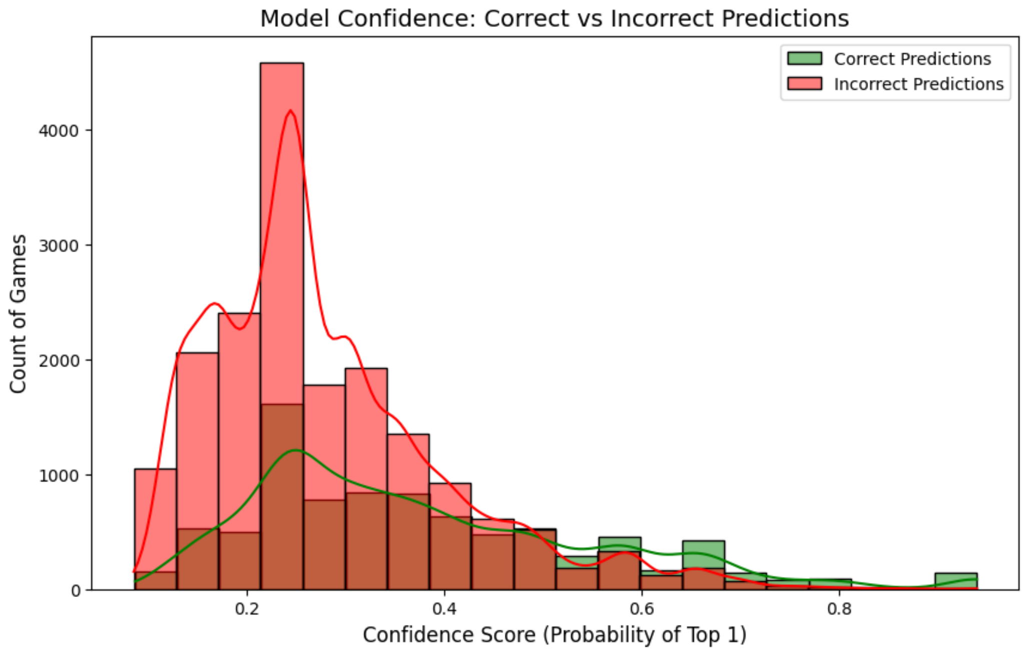


Accuracy vs. Player State (Gold Gain)

This graph demonstrates the robustness of our model across different game states. Whether a player is "snowballing" with high gold income or falling behind, the prediction accuracy remains remarkably consistent. This suggests that our model captures the fundamental itemization logic of champions effectively, regardless of whether the player is winning or losing, and is not biased toward specific economic scenarios.

Model Confidence

Finally, we analyzed the probability distribution of the model's predictions.



Distribution of Prediction Probabilities

The confidence plot shows a healthy distribution. The model is rarely "uncertain" (hovering around 0.1-0.2). When it makes a prediction, it usually assigns a high probability score, indicating that distinct patterns in the data (Gold + Role + History) create strong signals for specific items.

Summary of Classical Models

Through this project, we successfully moved from raw API data to a functional item recommendation engine.

- **Data extraction** from the Riot API allowed us to build a rich dataset of sequential decisions.
- **Exploratory Analysis** confirmed that itemization is not random but driven by Role, Economy, and Momentum.
- **The XGBoost Model** proved that we can predict the next purchase with high strategic accuracy (approx. 80% in Top-5), specifically excelling at identifying standard builds for Carries (ADC/Mid).

VI. Deep Learning Approach: Multi-Layer Perceptron (MLP)

To further push the predictive capabilities of our system, we implemented a Deep Learning model using a **Multi-Layer Perceptron (MLP)**. While XGBoost excels at structured tabular data, Neural Networks are capable of capturing highly complex, non-linear interactions between features and learning its own representation of the game.

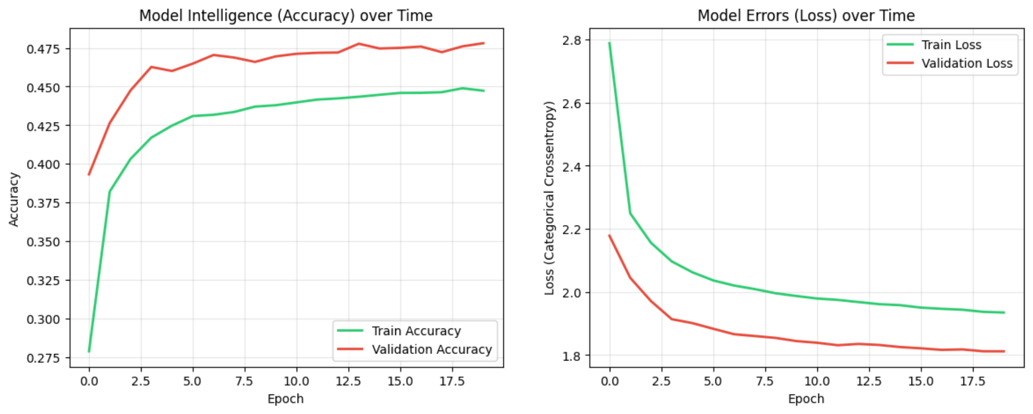
Model Architecture

We designed a Feed-Forward Neural Network using the Keras/TensorFlow framework. The architecture consists of:

- **Input Layer:** 449 dense features (after encoding).
- **Embeddings:** A learned vector representation for Champion IDs.
- **Hidden Layers:** Three dense layers (256, 128, and 64 neurons) with ReLU activation functions to capture non-linearities.
- **Regularization:** We applied **Dropout (30%)** and **Batch Normalization** between layers to prevent overfitting—a common challenge when predicting high-dimensional game data.
- **Output Layer:** A Softmax layer with 107 units, representing the probability distribution across all legendary items.

1. Training Dynamics

The model was trained over 20 epochs using the *Adam* optimizer and *Categorical Cross-Entropy* loss.

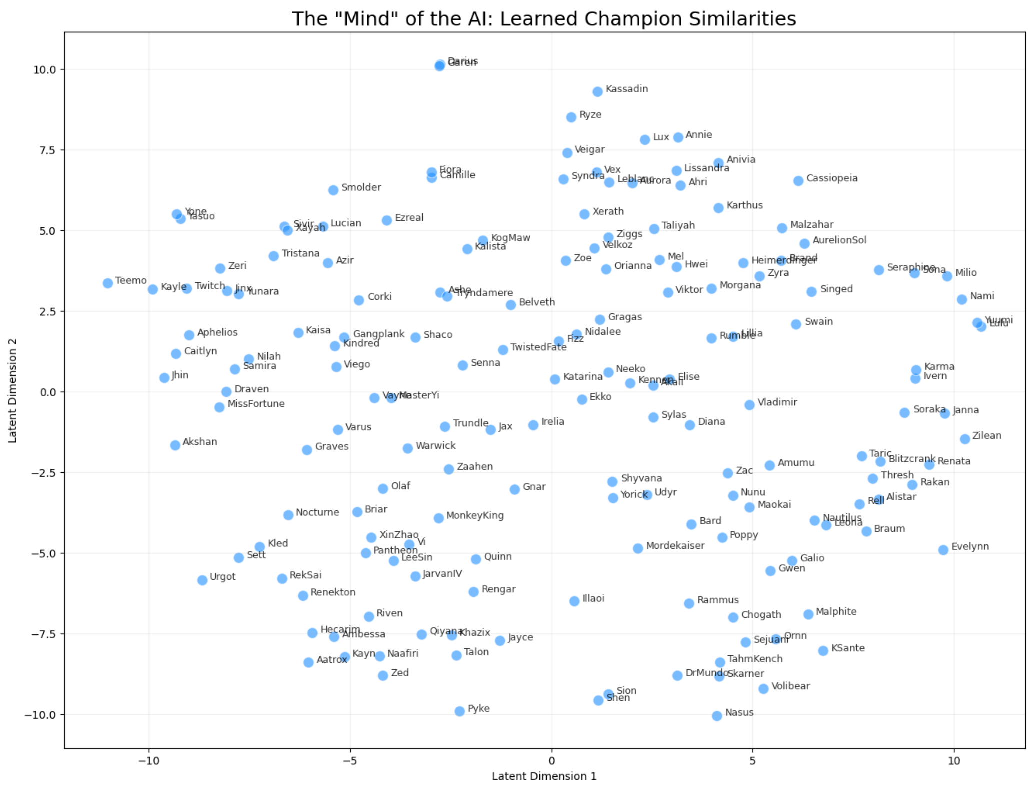


Training History: Loss and Accuracy over Epochs

The training curves (above) show a classic convergence pattern. The validation loss stabilizes around the 10th epoch, indicating that the model has learned generalizable patterns without memorizing the noise of the training set.

2. Learned Representations (Embeddings)

One of the most powerful features of Deep Learning is **Representation Learning**. Unlike the XGBoost model which uses "Classes" (Mage, Tank) defined by humans, the Neural Network learned its own internal map of champions.

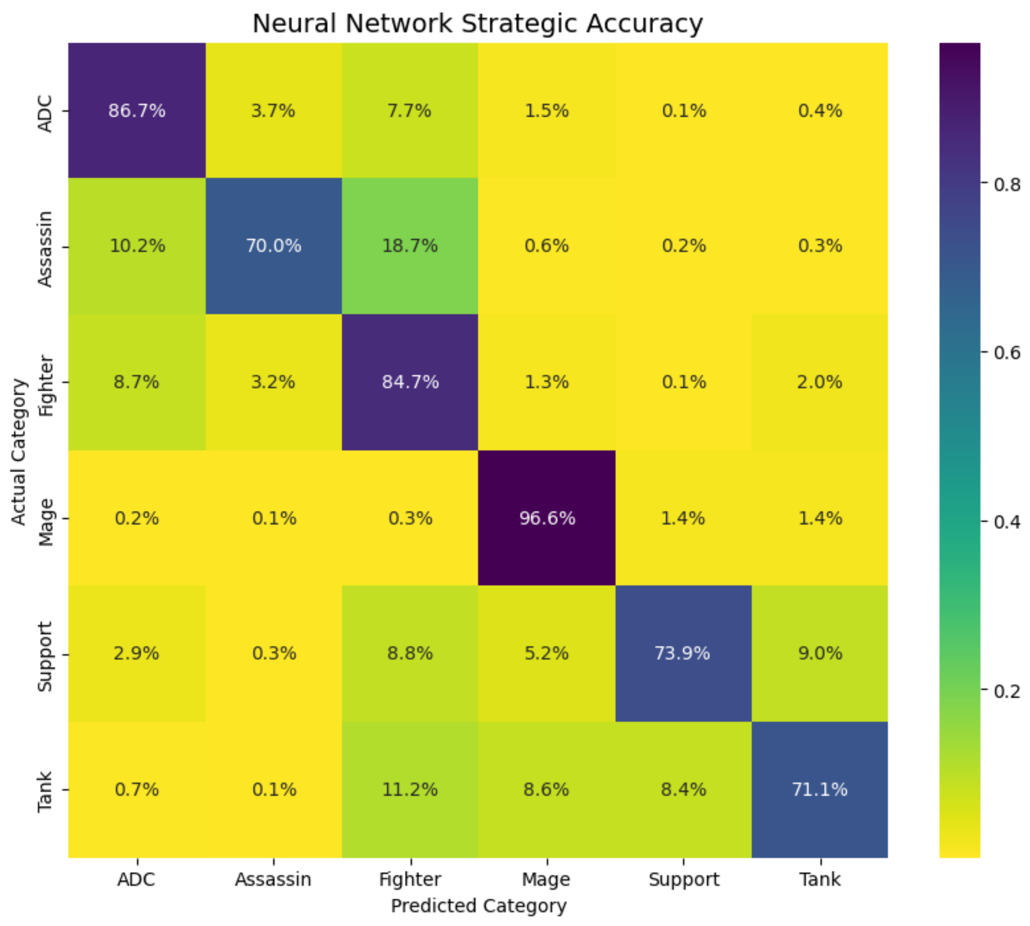


t-SNE Visualization of Learned Champion Embeddings

By extracting the weights from the model's 'Embedding' layer and projecting them into 2D space using t-SNE, we can see that the model **automatically clustered champions** based on their itemization needs. Marksmen are grouped together, distinct from Mages and Tanks, proving that the model "understands" the gameplay identity of each character purely from the data.

3. Strategic Understanding

To verify the model's reasoning, we analyzed its errors using a confusion matrix of item classes.

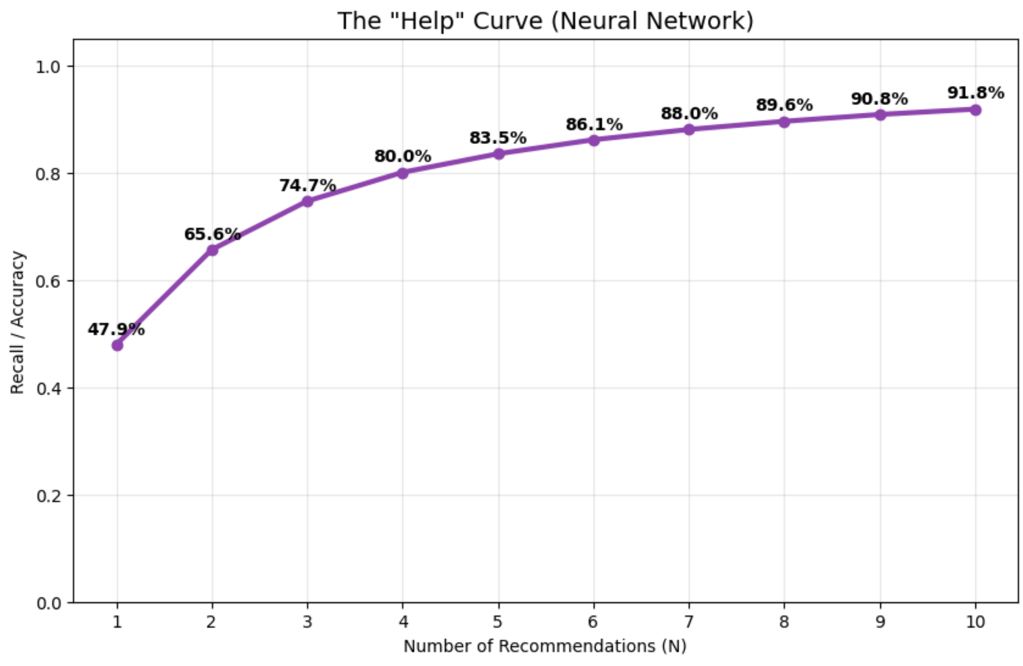


Strategic Accuracy: Confusion Matrix of Item Classes

The diagonal dominance in the matrix above confirms that the model is strategically sound. Even when it predicts the "wrong" item, it almost always predicts an item of the correct *class* (e.g., predicting a different Tank item for a Tank).

4. The Utility And The "Help Curve

We evaluated the model using the "Help Curve" metric to measure its practical utility as a recommender system.



MLP Help Curve: Utility of Deep Learning Suggestions

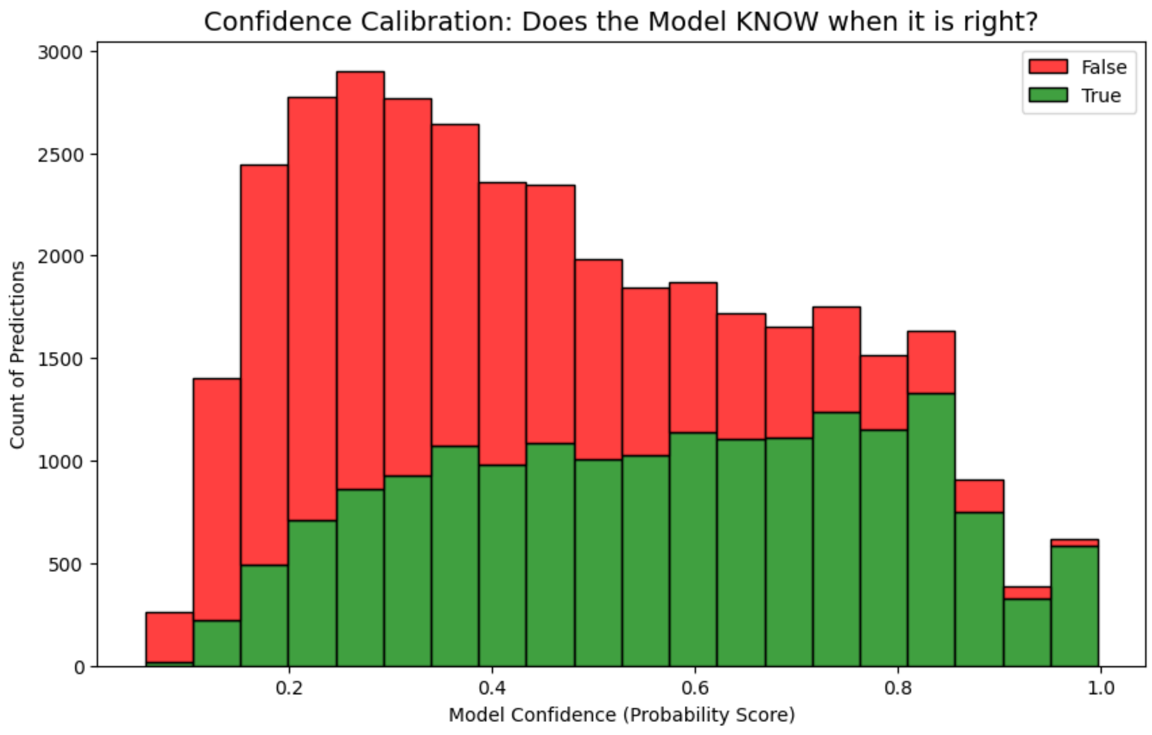
The Neural Network achieves performance highly competitive with XGBoost:

- **Top-1 Accuracy:** $\approx 43.5\%$
- **Top-5 Accuracy:** $\approx 81\%$

This confirms that the MLP is an effective filter, successfully narrowing down the shop's 100+ options to a relevant shortlist of 5 items that contains the optimal choice 81% of the time.

5. Confidence Calibration

Finally, we analyzed whether the model "knows when it is right."



Confidence Calibration: Correct (Green) vs Incorrect (Red) Predictions

The histogram above categorizes predictions by the model's confidence score.

- **High Confidence (Right Side):** When the model is >80% confident, it is overwhelmingly correct (Green).
- **Low Confidence (Left Side):** When the model is uncertain (<40%), the error rate (Red) increases significantly.

This strong calibration allows us to trust the model's "Best Guesses" while knowing when to offer the user more alternatives.

VII. Final Conclusion

In this project, we successfully engineered a data-driven item recommendation system for League of Legends, moving from simple heuristics to advanced Machine Learning.

Model	**Top-1 Accuracy**	**Top-5 Utility**	**Key Strength**
KNN	42.2%	79.0%	Captures "Peer Pressure" / Standard Builds
Random Forrest	42.8%	77.0%	Captures context-dependent decisions
XGBoost	43.1%	80.2%	Explainable Features (Role/Gold)
MLP (Neural Net)	**43.5%**	**81.0%**	**Learned Embeddings & Calibration**

Final Model Comparison

While all models achieved 80% utility, the MLP's ability to learn champion embeddings and calibrate its confidence makes it the most robust candidate for a real-world application. Future improvements could include **Deep Reinforcement Learning** to optimize for win probability rather than imitation, or **Graph Neural Networks** to model team synergy explicitly.