Video Game Discovery - Recommendation System to Connect Users to New and Relevant Games

Key Points

- I designed a content based recommendation system for users to find games most similar to their own, based on cosine similarity from feature vectors of textual and numerical data.
- I used a baseline cosine similarity function enhanced with feature weights from a multilayer perceptron (MLP).

Motivation

All major product centric platforms use some kind of product rec system, whether it be spotify, Netflix, Amazon, Instagram, etc.

One example of a product centric platform is Steam, a video game distribution service owned by Valve. Steam is one of the world's largest distributors and collections of digital video games in the world. Users can buy games, leave reviews, and have games recommended to them. Steam certainly uses recommendation systems to increase sales to users.

A passion project I came across is Backloggd, a free video game virtual library that lets users track their games, share reviews, and see what their friends are playing. While similar to Steam in the sense of a community platform for video games, Backloggd only serves as a virtual library, looking to connect users to new gaming experiences.

After making a free account on the site, I noticed the "Games" feature which is made to find new games only uses a static database query system, with no clear recommendation system in place.

For rec systems, there are 2 general types - content or collaborative. Content focuses on product to product relationships, while collaborative focuses on more user specific actions. I make my decision on which to use after looking at the data.

Data

Data comes from source, which is a subset of 1,099 unique games which comes from a larger set of 12,000 games. The 1,099 games featured in this subset are the most popular from the original set.

- Title - title of the game
- Team - studio(s) behind the game
- Rating - average rating out of 5
- Times Listed - Number of times users included the game in their personal library
- Genres - genres pertaining to the game
- Summary - written summary of the game
- Reviews - user written reviews, collected at time of scraping
- Plays - total number of plays for the game based on the site
- Playing - how many people are actively playing this at the time of scraping

- Backlogs - how many people bought the game but haven't opened it yet
- Wishlist - how many people want the game but haven't bought it yet

Note - I drop the column "Number of Reviews" as it is the same exact data as "Times Listed", there was an HTML error during data scraping for this.

The data is split into 2 groups. The first is text values, being

- Title
- Team
- Genres
- Summary
- Reviews

The second is numerical values, being

- Rating
- Times Listed
- Plays
- Playing
- Backlogs
- Wishlist

Basic cleaning involved dropping 15 missing values (13 ratings, 1 team, 1 summary). < 1 % of data removed, so no integrity loss. Release date was converted to datetime. The other numerical values were converted to ints. To ensure only unique games are included in the set, I drop duplicates based on the Title column. This reduces the number of rows from 1,512 to 1,085.

Problem Space

Form feature vectors from textual data t and numerical data n for each item i in the set. The user will then provide their chosen item, and will be provided the top 3 most similar scores based on cosine similarity rankings.

I will use different iterations of this system.

- First, with just cosine similarity of feature vectors as a baseline.
- Then, apply weights found through multilayer perceptron (MLP) to the baseline

One major limitation to this problem is the lack of reference labels in the set. Without a clear way to quantify the success of the similarity rankings, this work will be focused on optimizing the approach to achieve these rankings, rather than the outcome of the rankings themselves.

EDA

The purpose of this EDA is to find any feature trends or biases in the dataset before constructing the feature vectors.

Title

- K-means clustering

As a baseline for the title, I used full string tokenization with TF-IDF for weights. I expected unique titles after removing duplicates and whitespaces before vectorizing with TF-IDF. However, I found weights [11.7, 6.2] instead of a single weight.

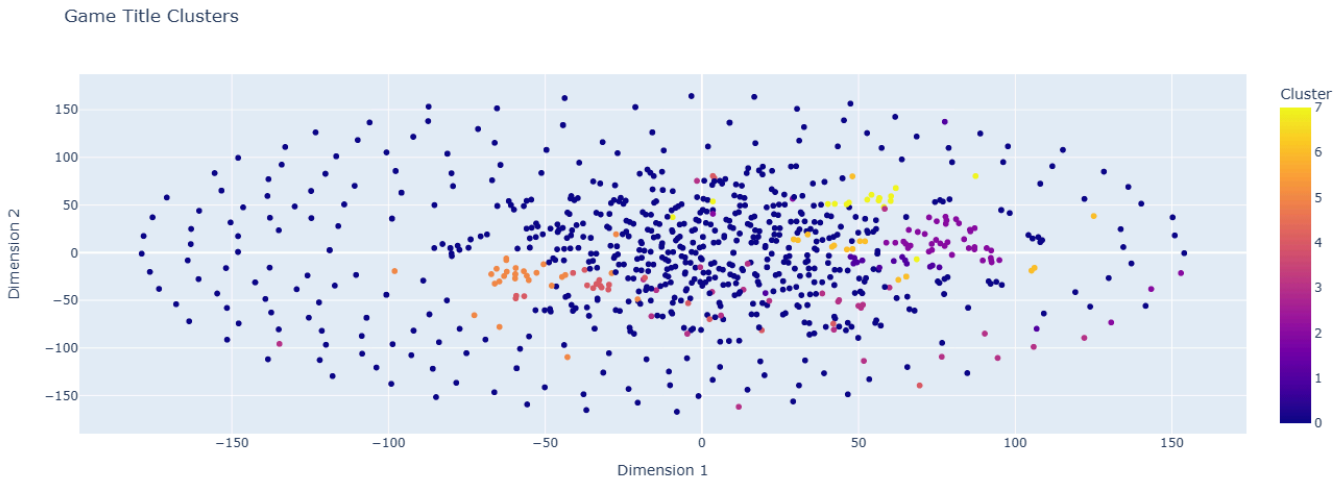
The 11.7 weight corresponded to "Star Wars Battlefront 2", which appeared twice:

1. "Star Wars: Battlefront 2" (2005)
2. "Star Wars Battlefront 2" (2017)

Colons were removed during tokenization, making these titles identical.

To resolve this, I renamed the 2017 release to "EA Star Wars Battlefront 2". After this change and re-running tokenization, all titles had an equal weight of 6.2, confirming uniqueness in the set.

With titles now fully unique, I use text clustering with k-means to group similar titles together. I use the elbow method to decide on 8 as an optimal number of clusters. After plotting the clusters, there are no clear distinctions to be made. Franchises with multiple games of the same brand are grouped together.



The franchises with enough weight in the set to be made their own clusters are

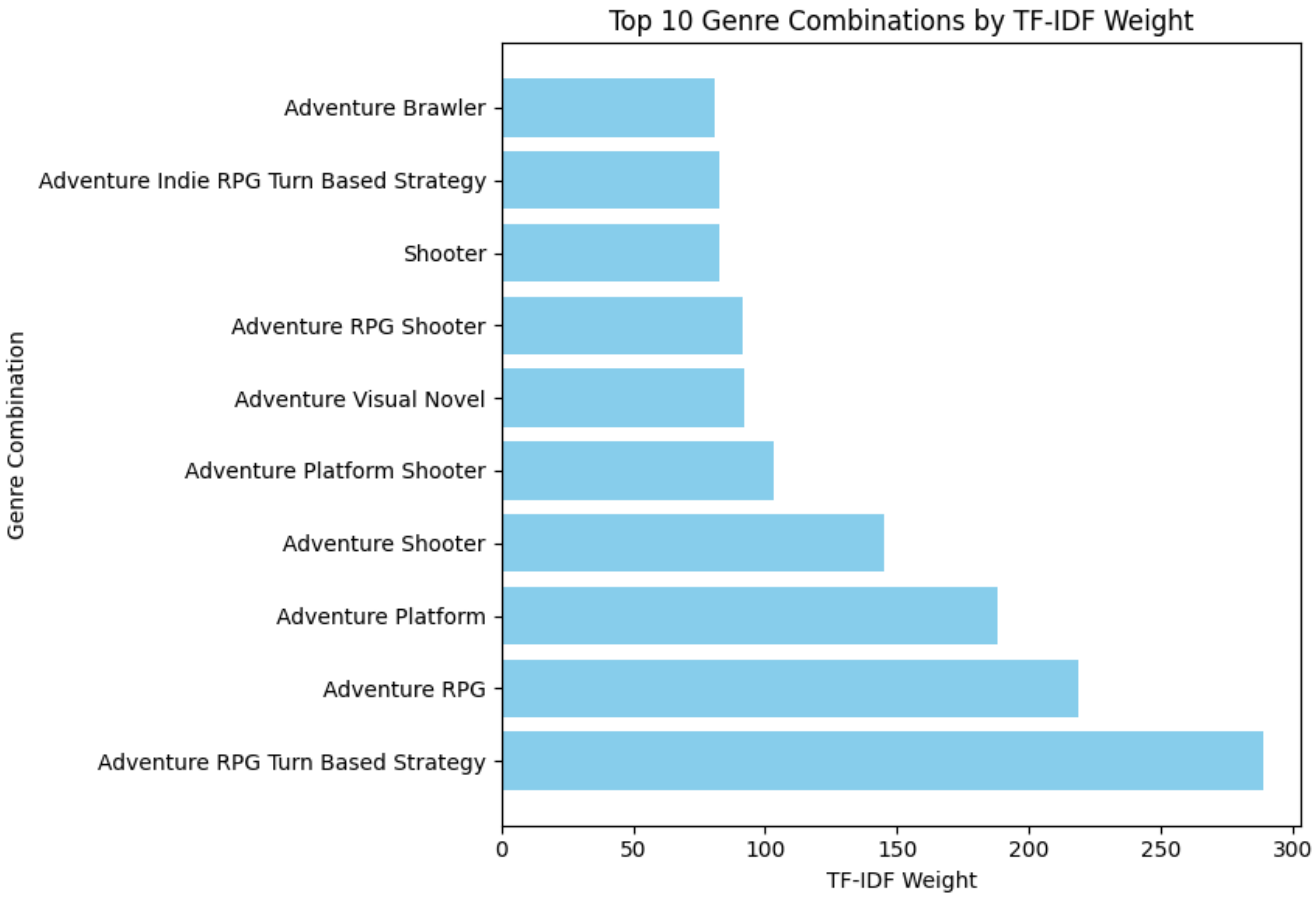
- Pokemon
- Assassin's Creed
- Mario
- Sonic

These are all popular franchises with multiple entries to their name. I will only use the title as an identifier, and will not use it as a predictive feature.

Genres

- Keyword extraction with tf-idf

There are 23 total unique genres within the set, creating a total of 254 genre combinations. Each game can have more than one genre listed, such as [‘Adventure’, ‘RPG’, ‘Shooter’]. To find the most prevalent genre combinations, I use keyword extraction with Tf - Idf. I avoid clustering here as genres are relatively simplistic in comparison to the other textual data, and want to provide more granularity.



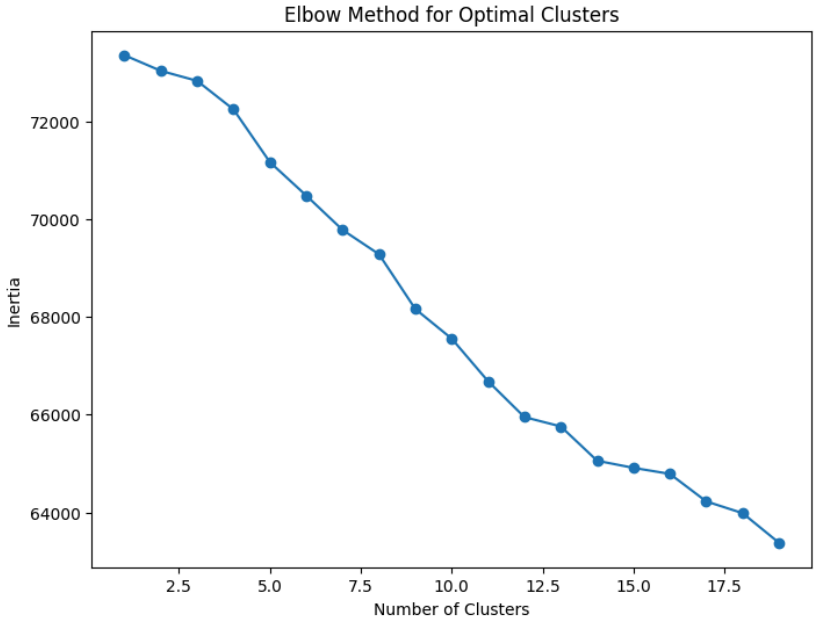
It is clear being an adventure game is favored in this dataset, as 9 of the top 10 genres have adventure within their combination. I assign each game a new feature, called Genre_Weight, which is the relevant genre combination weighted score.

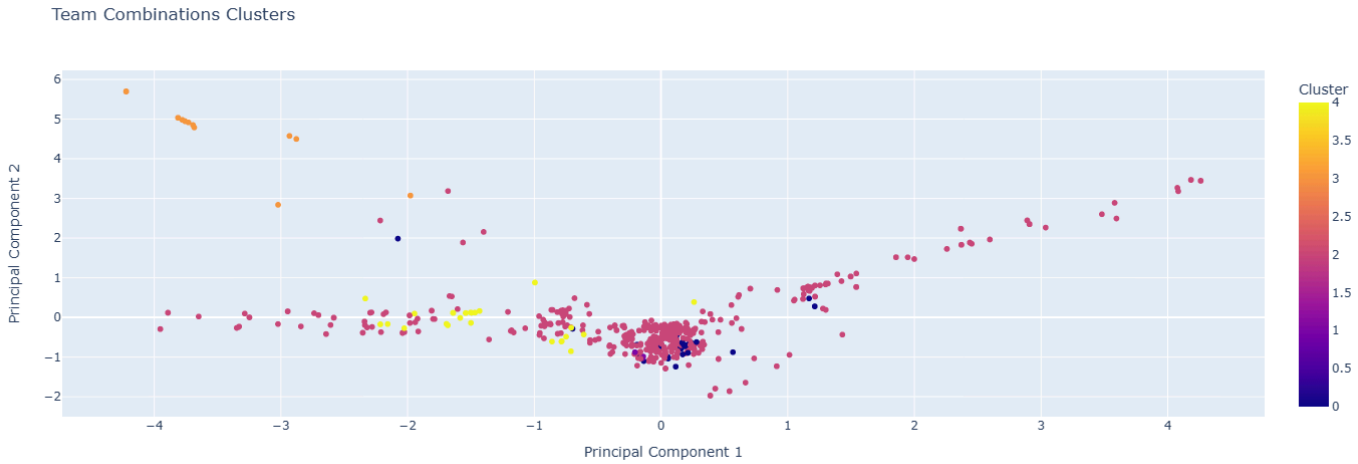
Teams

- K-means clustering

There are 655 unique teams in the set. Same as genres, they appear as combinations for each respective game, such as [‘EA’, 2k Games’].

I tokenize each set of teams into a single string for each game [‘EA’, ‘2k Games’] → EA 2k Games and use k-means clustering again to visualize any patterns. I use 20 clusters as a baseline, and can change this value later if I want to.



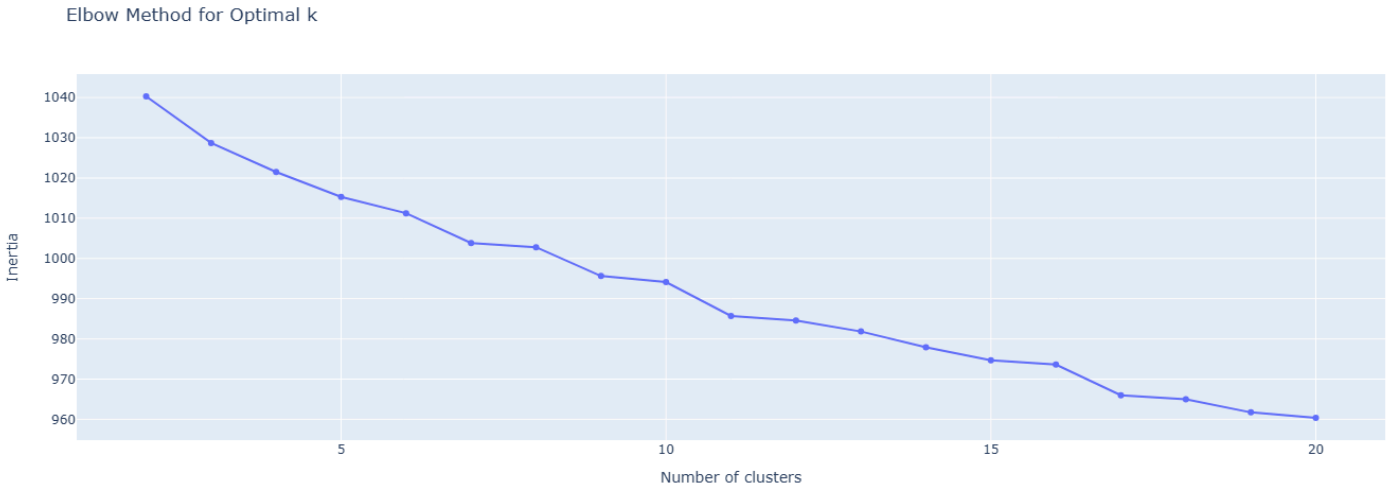


Nintendo (right), Capcom (bottom right), Ubisoft (top left), and Sony (bottom left), and all have their own respective branches. In the center of the graph is where we find our more independent and smaller game studios. I assign each game its respective team cluster under Team_Cluster.

Summary

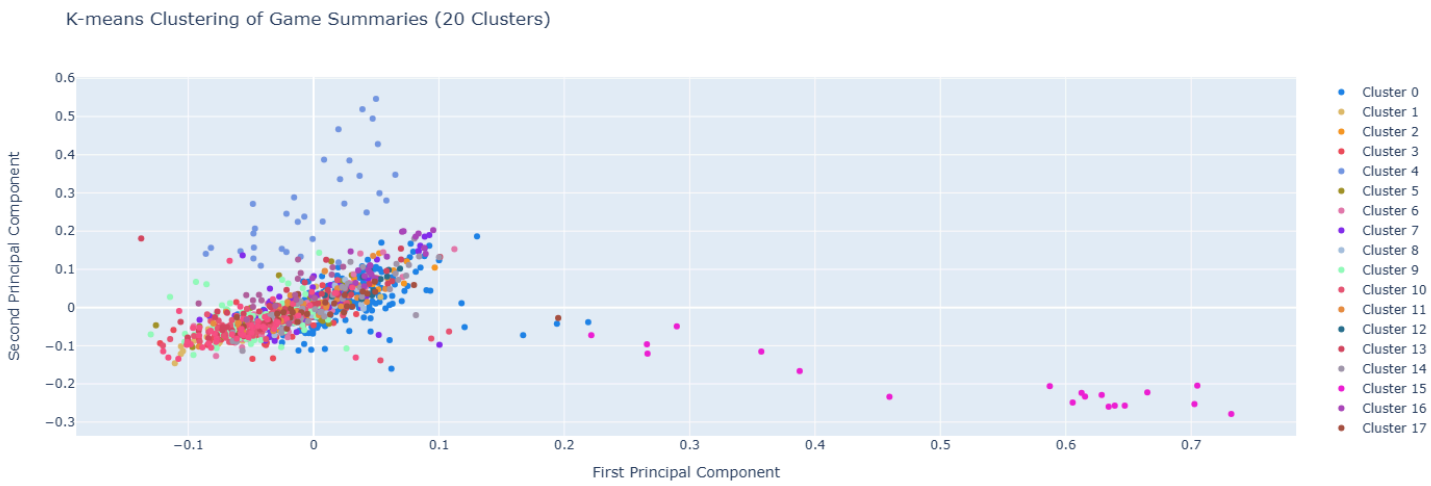
- k means text clustering

The game summaries provide insights into a game in a short text format. I start by finding an ideal number of clusters to use based on inertia value.



I decide on 20 clusters to capture as many patterns in the summaries as possible. From looking at the titles, I know some clusters will be entirely dedicated to the more popular franchise. To work around this, I identify the most frequent words within the summaries and add them as custom stopwords.

Many of these terms are generic and can apply to a wide grouping of games.



It's a good sign these clusters don't imitate the title clustering, meaning we are finding new patterns that didn't exist before.

Top words for each cluster:

- Cluster 0: man, person, 3d, third, mega, main, fighting, shooter, takes, known
- Cluster 1: black, half, assassin, white, creed, journey, co, person, ops, begins
- Cluster 2: cry, devil, may, dante, style, far, novel, second, dynamic, information
- Cluster 3: planet, kirby, metroid, people, samus, future, fate, space, alien, choose
- Cluster 4: fantasy, open, epic, rpg, remake, dark, japan, magic, remaster, graphics
- Cluster 5: puzzles, puzzle, solve, solving, challenges, ds, professor, escape, based, zero
- Cluster 6: island, battles, fast, paced, fun, kombat, king, wii, mortal, team
- Cluster 7: emblem, fire, based, rpg, tactical, war, turn, god, kratos, squad
- Cluster 8: star, duty, wars, warfare, call, force, modern, battlefield, ii, ops
- Cluster 9: dark, monster, monsters, deadly, another, journey, twisted, mind, human, master
- Cluster 10: family, kong, girl, hand, together, ancient, little, young, become, discover
- Cluster 11: dragon, nier, content, ball, apocalyptic, powerful, post, full, edition, award
- Cluster 12: bros, smash, luigi, wii, entry, princess, items, scrolling, platformer, side
- Cluster 13: horror, survival, resident, person, survive, silent, hill, third, role, dark
- Cluster 14: racing, kart, high, join, big, every, speed, favorite, including, go
- Cluster 15: gear, metal, solid, snake, stealth, kojima, enemy, hd, bonus, included
- Cluster 16: zelda, legend, link, resolution, oracle, dungeons, hyrule, fps, bioshock, models
- Cluster 17: music, 100, dlc, chapter, lego, prepare, rhythm, man, tower, marvel
- Cluster 18: strange, city, drake, unravel, tokyo, uncover, mysterious, face, nathan, deadly
- Cluster 19: persona, megami, tensei, shin, role, enhanced, powers, japanese, given, follows

I really like some of these clusters, there are some clear takeaways here.

- 0 points to fighting.
- 4 points to Japanese open world.
- 8 points to war based.

- 9 points to monsters.
- 12 points to platformers.
- 13 points to survival horror.
- 14 points to racing.

I take a look at some titles to see where their summaries are grouped.

Elden Ring: 4 ← open world fantasy, Japanese development, makes sense.
Hades: 0 ← Hack and slash fighting games, makes sense.
Legend of Zelda: Breath of the Wild: 16 ← the Zelda cluster.
Undertale: 5 ← RPG puzzle solving game, makes sense.
Hollow Knight: 0 ← 2d dungeon crawler.

I assign each game its cluster value as Summary_Cluster.

Reviews

- Review Similarity with SBert

Reviews are the most unstructured feature within the textual data. I start with tokenization and cleaning.

Reviews per game are separated through outer square brackets []. Each game can have multiple reviews, these are separated by slashes \. Use of \\n to denote newlines, needs to be removed.

Additionally, I use langdetect to find a total of 7 unique languages, with the majority being in English (1024 reviews), followed by Portuguese (50 reviews), and Spanish (9 reviews).

With a multilingual presence, I use *distiluse-base-multilingual-cased-v* from SBert to create embeddings for each review. These embeddings are then stored as 'Embeddings' for each game.

Preprocessing + Modeling

I start by standardizing Rating, Times Listed, Plays, Playing, Backlogs, and Wishlist with StdScalar().

Now, it is time to form the feature vector for each game.

From our sentence transformer embeddings, I need to convert each review into a numpy array. I can then concatenate the 1 dimensional features with this high dimensional array to form a feature vector for each game, based on all textual and numerical data present in the set.

Now that we have our feature vectors, we can try different approaches to best rank them based on similarity score.

As a baseline, I can just use cosine similarity to take a given game and output the *x* most similar titles. This is the most straightforward and simple approach. However, there are a few things we don't account for here with this approach:

- no feature importance or weighting taking place, each feature contributes equally in the similarity score.

- curse of dimensionality, our feature vectors are very high dimensional.
- any biases present in the set.

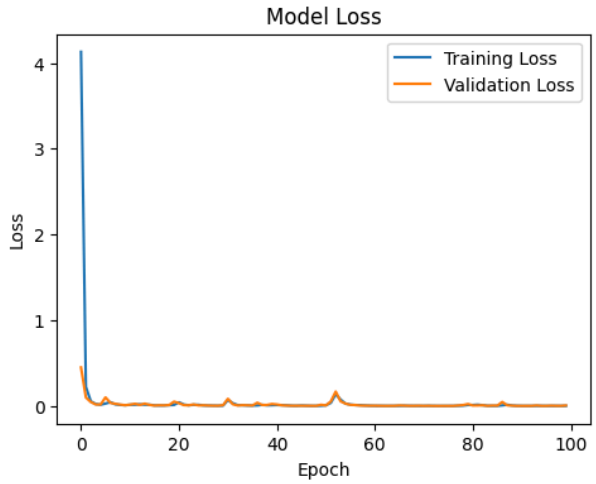
I use Elden Ring (because it's the first game in the set, and a very popular one) as a baseline. It is a dark medieval fantasy world, with an emphasis on difficulty. The top three most similar titles returned are

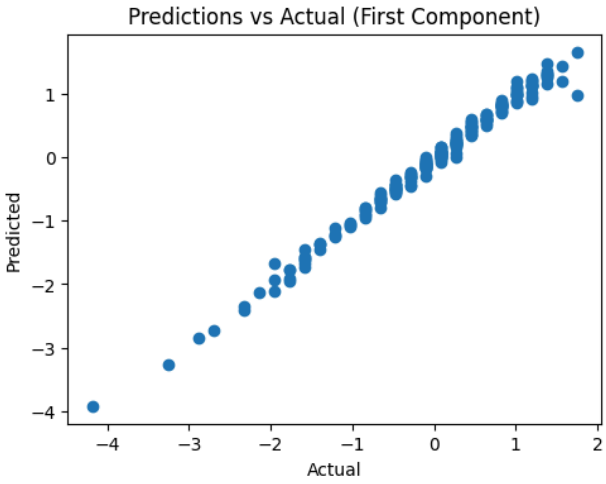
- Metroid Dead 0.99
- Kirby and The Forgotten Land 0.99
- Super Mario Odyssey 0.98

Not the most compelling list. For example, Metroid Dead is a 2-d platformer, while Mario and Kirby are much more lighthearted and intended for young kids. Additionally, the similarity scores are very high and very similar across all three titles, suggesting proper differences are not accounted for.

To improve this performance, we can use a multilayer perceptron (MLP). There are several advantages to this form of neural network for our situation.

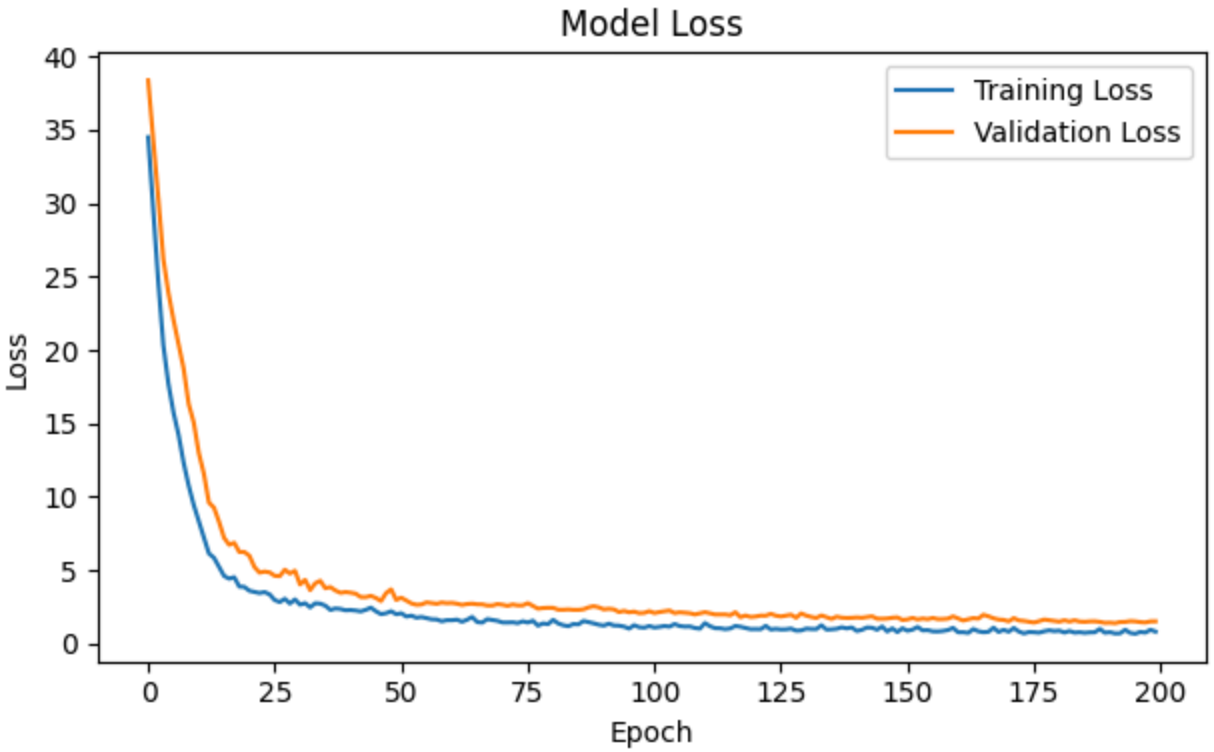
- Flexible Architecture: MLPs can be easily adapted to handle high-dimensional outputs by adjusting the output layer.
- Non-linear Mapping: They can learn complex, non-linear relationships between inputs and high-dimensional outputs.
- End-to-end Learning: MLPs learn the entire mapping from input to output in one go, without needing separate dimensionality reduction steps.





While the predictions look good, validation loss plateaus, and training loss decreases sharply, suggesting overfitting. To address this, I incorporate the following:

- L2 Regularization
- Batch Normalization
- Dropout Layers
- ReduceLROnPlateau
- Early Stoppage
- Increased epochs to 200



Much better now. I then pull the weights from the first layer and implement them back into my cosine similarity calculations.

With our feature weights from our MLP model, let's go back and apply these to the baseline cosine similarity function we used above.

I query on Elden Ring again, and this time I get much better results.

- Witcher 3, 0.35
- Bloodborne, 0.31
- Dark Souls, 0.29

Much better in my opinion. All three are dark medieval fantasy settings, just like Elden Ring. We see the Teams_Cluster weight come into play here, as Bloodborne and Dark Souls are from the same dev team behind Elden Ring.

Next Steps

The next steps in this project would be to polish the system for more personalized recs, as well as providing pre labeled data to help with similarity rankings.

1. Hybrid Content-Collaborative Approach
    - Implement a prelabeled training set that includes user statistics and preferences.
    - Incorporate collaborative filtering techniques to leverage user-item interactions.

- Combine content-based features with collaborative features to create a more robust hybrid model.

2. User Feedback Integration
    - Implement a mechanism to collect and incorporate user feedback on recommendations.
    - Use this feedback to continuously improve the model's performance over time.

3. Personalization
    - Develop user profiles based on their game preferences and playing history.
    - Tailor recommendations to individual users based on their profiles.

Conclusions

While the current system provides a solid foundation, there's significant potential for improvement, particularly in incorporating user-specific data and collaborative filtering techniques.