Reverse Engineering the GiantBomb API: Using Word Embeddings to Predict Video Game Recommendations

Yash Karandikar
ykarandikar@oxy.edu
Occidental College

1 Introduction and Problem Context

Working on this project is interesting to me because it provides an opportunity to combine machine learning (ML), natural language processing (NLP), and real-world data analysis into one problem. In terms of ML, I want to gain ML experience so that I can apply it in my career going forward (possibly in a healthcare-oriented context). Focusing on video games as the subject for the supervised machine learning problem has also allowed me to tap into an immense amount of text-based video game data (descriptions, reviews, and other qualitative aspects of online game discourse). In this way, I can use NLP techniques as ML features and apply them to a subject of interest. The real-world nature of this problem is also stimulating and challenging, because the data comes from an API that requires authentication and preprocessing. This is very different from the cleanly defined, open-access datasets used in my other courses. Combining ML, NLP, and real-world data modeling into one project has helped motivate my work in comps, and it has challenged me and helped me grow as a developer and machine learning researcher.

In terms of societal applications, recommender systems are critical to today's highly technological, choice-laden society. These recommender systems are found in many different aspects of life. Commercial websites (like Amazon [8]), video content platforms (YouTube [20] and Netflix [12]), and social media websites (LinkedIn [26]) are just a few examples. To help process data and provide recommendations, supervised learning (especially self-supervised variants of supervised learning algorithms) is often explored in the implementation of these recommender systems. Addressing data sparsity, alleviating the "cold start" problem of not having an initial suggestion to base future recommendations on, and including temporal aspects of recommendation as user sentiment changes are all good reasons to apply supervised or self-supervised learning. [25, 45, 37]

The difficulty of this project also stems from the machine learning side of the problem. Because common approaches for recommendation systems require immense amounts of user data and interconnected user profiles, they are not practical for the scope of this project. Instead, transforming the recommendation problem into a supervised learning problem creates an opportunity to practice real-world machine learning. Deciding which dataset I can use to provide ground-truth labels for recommendation is good practice, because real-world applications of ML often require extracting and preprocessing data from different sources in order to solve the question at hand. After obtaining the data, practicing iterative ML algorithm improvement - specifically identifying breakpoints and making changes to the algorithm to test out new hypotheses - has helped me gain a better perspective of how machine learning is conducted.

2 Technical Background

Current approaches to recommendation systems use two main strategies, collaborative filtering and content-based recommendations. Content-based approaches examine how different entities connect together in qualitative ways. These recommendations typically cluster semantically similar products (especially when text-based descriptions are reasonable indicators of product appeal). One approach involves converting string queries (e.g., game titles like "Breakout" or "Tetris") or string content (game descriptions, reviews, and other corpora) into numerical vectors and comparing their contents to vectors of related games. The TF-IDF algorithm uses sparse numerical vectors (meaning that most vector elements are 0) to identify common keywords and penalize non-informatic keywords. Approaches that use it aim to find patterns of highly informatic, low-density word usage. [31, 4, 16] Similarly, word embedding approaches such as Word2Vec [44] or the Google Universal Sentence Encoder [15] use dense vectors (with mostly non-zero vector elements) to compare similarity.

On the other hand, approaches that use collaborative filtering consider user similarity when linking together products. The key idea of collaborative filtering is to learn what product a customer likes, find related users in the database who like the same product, and use related users' profiles to generate suggestions for the customer. As an example, consider a user who shares that they like Super Mario Bros. Collaborative filtering examines other users in the database

who also enjoy Super Mario Bros and recommends their enjoyed entries (perhaps The Legend of Zelda or Donkey Kong) to the initial user. The database thus needs to be data-dense enough to connect related users to the input provided by the initial user. Within industry, this data density requirement means that collaborative filtering is especially powerful in recommendation systems like Spotify [27] which have a large, communicative user base. Collaborative filtering-based approaches are also standard in the literature. [5, 19, 23] Because of its ability to leverage the power of social networks for recommendations, collaborative filtering is a highly influential technique. Collaborative filtering is such a powerful idea that it can form a benchmark model for future improvements. [18, 10]

In both content-based approaches and collaborative filtering applications, recommender systems require large datasets (of products, users, or both) to effectively find connections between entities. Machine learning poses a way to process this immense data and identify patterns to form clusters, similarities, and eventually recommendations. Many types of machine learning techniques are applied to the recommendation problem. Supervised learning (where labels are known ahead of time), unsupervised learning (where the algorithm learns to cluster features by itself), and other approaches are also used. [3, 24] Supervised learning algorithms are especially helpful in recommendation for a project of this scope, because answers are known ahead of time, so the focus can be improving the recommendation algorithm with respect to standard evaluation metrics rather than considering recommendation-specific evaluation metrics. Depending on the recommendation problem, different types of algorithms can be used.

In my comps project, the problem of recommendation is transformed from a collaborative filtering problem into a supervised learning problem. Because collaborative filtering requires millions of user profiles, any algorithm that uses collaborative filtering needs access to a robust user dataset. Instead, this project finds ground truth labels for recommendation from an online application programming interface (API) and builds a machine learning algorithm to perform well on the video game dataset. Thus, the problem turns from recommendation to *reconstruction* of a recommendation algorithm using supervised learning. The problem is formulated as follows: Given an input game g, decide whether a potential recommendation game r should be recommended.

To solve this supervised learning problem, the support vector machine (SVM) algorithm is used. SVM is a binary classifier, so it separates data points into two classes. SVM can create a linear or nonlinear decision boundary to differentiate the two classes. In this case, the kernel (the core that determines how SVM creates the decision boundary) is the radial basis function ("RBF") kernel. RBF is the standard for any nonlinear SVM classification. So, RBF-kernel SVM creates a nonlinear decision boundary between points of two classes (1s and 0s) such that distances from the margin are reduced as much as possible. [13] Cases with high class imbalances (where there are far more 0s than 1s, or vice versa) can be addressed by artificially creating similar points and adding those to the smaller class. [33] The way that SVM is constructed combines optimal margin classification (reducing margin distances) and an efficient "kernel trick" which teaches the classifer how to map to nonlinear decision boundaries. By definition, the kernel trick helps SVM to work with high-dimensional data, which makes SVM particularly effective for problems that require high-dimensional inputs. [43, 30]

The input data to the SVM uses techniques from natural language processing (NLP) to transform textual data into numerical data that the machine learning model can use for training and testing. One possible NLP approach is to identify informatic words by topic modeling, where key ideas are extracted from a series of text documents and used as ML features. [21, 32] A related, but distinct, approach is to identify informatic words by context/through their syntactic structure [31, 28]. Another technique is to use word embedding vectors or matrix/tensor representations of word embeddings, where words are mathematically transformed from strings to floats and represented in N-dimensional space. [39] Many NLP approaches rely on powerful ideas from linear algebra to turn informatic text content into numerical machine learning features.

3 Prior Work

Prior work uses key ideas outlined in the technical background section as a foundation and strives to pivot or improve upon them. For instance, some approaches take collaborative filtering as a baseline and provide a more efficient, performant model. Others address limitations of collaborative filtering (the issue of *only* considering user profiles) and limitations of content-based recommendation (the difficulty of gathering enough data to start the recommendation process) by combining them with other techniques. Many approaches display an interdisciplinary perspective by combining collaborative filtering with ideas from related fields (computational linguistics/NLP, ML, and so on).

Because of collaborative filtering's efficacy in the recommendation space, many approaches combine content-based methodology and non-social, numerical metrics with collaborative filtering models. Within industry, the Steam Interactive Recommender and Quantic Foundry [42, 38] generate recommendations for a user U by combining data-dense collaborative filtering techniques with non-social metrics that corroborate recommendation choices. For example, Steam looks at users who own similar games to U

in *their* libraries, but it also looks at U's user play time as an objective metric to see if U actually enjoys their owned games. Similarly, Quantic Foundry combines collaborative filtering with profiles of gamer psychology - what motivates and interests players is a qualitative metric that further informs the collaborative filtering approach. Within the research world, collaborative filtering is often a benchmark model or baseline approach. Many approaches combine collaborative filtering with other analyses and hybrid methodologies. [9, 18, 23]

Interdisciplinary approaches using ML and NLP are also common because of their ability to apply various performant algorithms to the recommendation space. Machine learning algorithms like SVM can be tuned to delineate and output recommendation, which inspired part of this project's motivation. [24] Because these ML algorithms are used to simulate recommendation, conventional ML metrics are used to evaluate the approaches [3, 41, 35]. Similarly, NLP approaches appear in recommender systems as a means to categorize, extract, and recommend different entities. Content-based approaches that use categories like genre, theme, or sentiment as features can help classify the propriety of a potential recommendation. [30, 17]

A specific subset of ML and NLP approaches can help address the limitations of typical recommendation approaches. These limitations are primarily the difficulty of gathering enough information to start finding related items (the "cold start" problem). Any representation or data structure that can provide a qualitative feature (removed from quantitative metrics like similarity score, number of connections in the social network, etc.) is a potential improvement in performance. Some researchers turn to word embeddings as an answer. Combining word embedding content-based analysis with benchmark collaborative filtering obtains the best of both worlds from related users and products. [29] Similarly, context-aware methods of word embeddings enhance typical content-based approaches (by considering context sensitivity, syntactic structure/relationships, etc.) and provide relationships between key words in recommendation corpora that can be informatic features in their own right, independent of the density of the database. [11, 10]. Such word embedding approaches are similar to the word embedding vector features used in this project.

4 Methods

4.1 Problem Formulation

I am reconstructing the GiantBomb API [2] (and specifically the GiantBomb similarity algorithm) by using a support vector machine algorithm trained on word embeddings of game descriptions. This reconstruction is what transforms the recommendation problem (should a game be recommended with respect to some initial input?) into a machine learning problem (does the algorithm output the correct label?) to solve.

The GiantBomb API acts as follows: given a query game g, it provides a set of similar games S. For this project, for every $s \in S$, s is considered a valid recommendation for g. Conversely, for every $n \notin S$, n is not a valid recommendation for g. Using these definitions, the SVM algorithm can reconstruct the API output.

Let g be the query game, and let r be a potential recommendation for g. Let S be a set of similar games for g, obtained by $GB(g) = S$. Then the GiantBomb API's answer to the recommendation problem is to check if $r \in S$. If it is, the GiantBomb API is considered to return 1; else, 0. Similarly, the SVM's answer to the recommendation problem is to run $SVM(g, r)$, which gives a label 0 or 1. The performance of the SVM can then be measured by how well the SVM's labels correspond to the GiantBomb API's labels.

By using the GiantBomb API's labels as the ground truth data, training this supervised learning algorithm is the same process as training any other supervised learning algorithm. Thus, the problem statement is as follows. Given a query game g and a potential recommendation r as input, give output of whether the recommendation makes sense with respect to the query. Equations 1-3 display example runs of the SVM algorithm to illustrate sample inputs and outputs.

$$recommender(g, r) = 1 \tag{1}$$

Eq. 1: r makes sense as a recommendation with respect to g.

$$recommender(\text{"}Tetris\text{"}, \text{"}Breakout\text{"}) = 1 \tag{2}$$

Eq. 2: Breakout makes sense to recommend if Tetris is the query game.

$$recommender(\text{"}Tetris\text{"}, \text{"}Baldur's Gate\text{"}) = 0 \tag{3}$$

Eq. 3: Baldur's Gate does not make sense to recommend if Tetris is the query game.

4.2 Dataset

Game titles were obtained from a combination of calling the GameSpot API [1] and reading in the Metacritic CSV [14]. The GameSpot API returned game titles, a collection of demographics (genres, themes, franchises), and text (sentence-long taglines called "deck" and longer-form paragraph called "description") as the response. The Metacritic titles were fed into the GiantBomb API to receive the same information. The query set Q mapping titles to their demographics and text was then formed. When possible, titles were also mapped to their respective reviews (from the

Metacritic CSV and GameSpot API) where they were available. Figure 1 provides a sample entry in the query set.

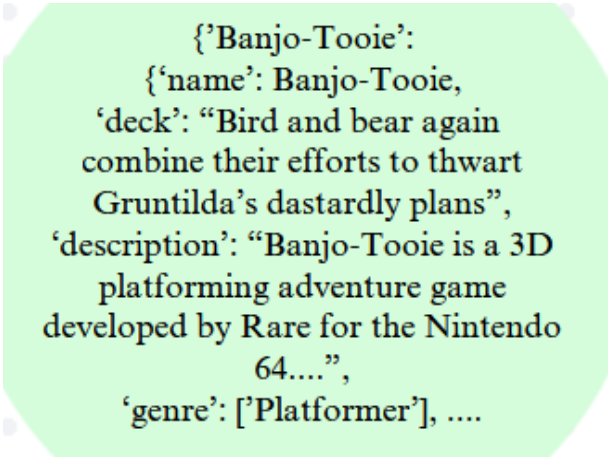

Figure 1: Sample entry in query set Q, mapping a game title to its text data and demographic categories

After forming the initial query set Q, the dataset for machine learning was built. For $g \in Q$, a set of similar games S was obtained from the GiantBomb API (such that $GB(g) = S$). Every similar game $s \in S$ should be recommended (label 1). Similarly, every game $n \in Q$ where $n \notin S$ should not be recommended (label 0).

The tokenized words in the decks and descriptions of g and either s or n were combined into one string. This string was converted into a word embedding vector, which formed the feature vector for the SVM. These vectors were mapped to the label (1 for s and 0 for n). When reviews were available, adjective-context pairs were identified in the review text using dependency parsing. [31, 28] The words in the pairs were added to the combined string. So, the word embedding vectors of deck and description (and adjective-context pairs, when available) formed the features of the dataset, and the label was either 0 or 1 depending on the game similarity.

4.3 Data Processing

Data processing was conducted to smooth out the results. There were far more non-similar games than similar games, so synthetic minority oversampling was used to offset class imbalances. [33] This technique generates points in N-dimensional space which are similar to the points already existing in the dataset. The goal is to balance the frequencies of the labels 0 and 1. For the SVM algorithm, dimensionality reduction was applied by principal component analysis. [43] Before fitting the SVM to the data, the dataset was projected from an N-dimensional space to a 2-dimensional space using linear algebra. This technique ensures that the decision boundary formed between the two classes is more precise.

4.4 Algorithm

For this experiment, SVM was used because of its ability to create a nonlinear decision boundary between the two classes. [13] It is a better choice for the task than other supervised learning algorithms like logistic regression, K-nearest neighbors, and tree-based models like random forest.

By design, logistic regression deals with continuous data that has real-world, inherently meaningful implications. For example, logistic regression can be used to model housing prices given features of distance to the beach, highway, etc. This data is a real-world value (physical distance), so each data point has a meaningful physical representation. In this case, reducing distance to the beach may have a continuous, smooth increase in price. By contrast, the word embeddings used as feature vectors are not inherently meaningful, because they are numerical representations of words. So, a nudge in the word embedding vector values gives no intuition about how the word being represented might change, and it does not help in guessing similarity of different products the same way that a change in distance to the beach might help guess a change in housing price.

For K-nearest neighbors, the goal is to cluster together related entities by proximity. Using this model, given a game g, potential recommendations could be represented as points that are close to g geometrically, and they could be evaluated by Euclidean distance. However, the problem at hand is not a problem of game similarity as much as a *reconstruction* of the GiantBomb API. Because the API provides a list of similar games, the labels are less interesting than the process of reconstructing the process that provides the labels. As a result, an algorithm that determines similarity via inter-cluster distance is less necessary.

Finally, tree-based models like random forest often require enumerably many features that are determined before model training. The idea of random forest is to create partitions that isolate one part of the dataset at a time. For example, one partition in a random forest could be "distance from the beach", where a distance greater than a certain amount leads to a 0 label ("will not buy the house"). Every single partition in such a model is isolating certain cases. However, in this project, natural language is the key feature. The variability and density of natural language makes it hard to isolate numerical, meaningful features ahead of time, so random forest cannot find meaningful partitions to use.

Consequently, the SVM algorithm was the best choice for this specific use case. The algorithm was trained on the dimensionality-reduced dataset of word embedding vectors of deck/description. Figure 2 displays a sample X_train and

y_train element. The list of word embedding vectors in the X array corresponds to the first label in the Y array.

```
(Pdb) print(X_train_lowdim)
[[-0.22052667 -0.02281859]
 [-0.16389866 -0.14357036]
 [-0.25638531 -0.12966227]
 ...
 [-0.30287167 -0.2272258 ]
 [ 0.60433637 -0.12612876]
 [ 0.62722653 -0.20777729]]
(Pdb) print(y_train)
[0 0 0 ... 0 1 0]
```

Figure 2: Training dataset mapping word embedding vectors to labels

This dataset includes embeddings from both the query game g and the potential recommendation $s \in S$ or $n \notin S$. Word embeddings were chosen instead of TF-IDF because TF-IDF uses sparse word vectors (where a majority of vector elements are 0s). If sparse vectors are used, there are artificial positive matches between query game vectors and potential recommendation game vectors. This is because all vectors contain mostly 0s - there is overlap almost by definition. Such overlap leads to a non-discriminative classifier that is prepared to recommend everything. By contrast, word embeddings use dense word vectors, so game similarity can be constructed with respect to non-zero vector elements.

4.5 Hyperparameter Tuning

Hyperparameters are parameters used in model training which can be adjusted to tweak performance. They are not fundamentally related to the dataset, but they do change some aspect of the model parameters which can slightly boost performance.

The hyperparameter of interest for SVM is the C-value, which is inversely proportional to the tightness of fit around the decision boundary. How well does the model trade off between effectively fitting the training data and remaining general enough to fit the test data? A lower C-value means that the model has a better fit on the data, but it becomes more likely to overfit on test data. By contrast, a higher C-value means that the model has a worse fit on the data, but it becomes less likely to overfit on test data. The idea of tuning the C-value is to find the optimal point between a good fit and generality.

Accordingly, the C-value is a hyperparameter of interest. 5-fold cross validation is used to split the train/test data in 5 different ways so that C-values can be compared across models trained on different sections of the dataset. C-values of 1, 10, 100, and 1000 were compared.

4.6 Outcome

The desired outcome is a provided label (0 or 1) when a query g and a potential recommendation r are fed into the SVM algorithm. This outcome meets the goal of the project, because the idea is to use a supervised learning approach to reconstruct what the GiantBomb API would say.

As stated in the problem formulation, the GiantBomb API takes a query game g and returns a set of similar games S in an operation $GB(g) = S$. A valid recommendation for g is defined as $s \in S$. So, the GiantBomb API answer (0 or 1) to the recommendation problem can be determined by checking if the potential recommendation $r \in S$.

As a result, running $SVM(g, r)$ simulates running $GB(g) = S$ and checking if r is in the resulting set S. Since the SVM algorithm gives a label, that label is equivalent to running and checking whether or not r should actually be recommended ($r \in S$). In this way, the SVM is able to reconstruct what the GiantBomb API says about game similarity.

5 Evaluation Metrics

The problem of interest has been turned from a recommendation problem (with standard approaches of collaborative filtering and content-based analysis) into a supervised learning problem. As a result, an immense amount of user data, preferences, and connections is no longer needed. To evaluate the methods, standard machine learning metrics can be used.

Even though the features for the machine learning model use word vectors of deck and description, cosine similarity is not applicable as a metric for the end result of the model. Cosine similarity is used frequently in the literature [19, 22, 40], but it is primarily used in cases where natural language context is not considered. (For instance, a database search only requires string similarity between a query and an article - it does not necessarily need *semantic* similarity between the two tokens.) In other words, raw word counts and frequencies are the features rather than the semantics of the words themselves. Since the features for the SVM are primarily natural language, context (word order, syntactic structure, and parts of speech) still matter for effectively training the classifier. Cosine similarity would be applicable for a search engine-like tool (matching query strings to database strings), but it is less applicable for matching game descriptions to game descriptions.

Other standard metrics of recommendation systems are inapplicable because they measure good recommendation rather than good reconstruction. Some recommender systems evaluate results with respect to the user rather than with respect to an external set of ground truth data. Novelty, serendipity (a measure of unexpectedness and relevance for the user), and other such metrics are valuable when the evaluation considers whether the user likes the product. [36] For instance, scoring well on serendipity is a good sign for a system like Amazon [8] which wants to maximize profit by presenting new yet useful products to customers. However, the project focuses on reconstruction of the GiantBomb API rather than measuring how well users would like a recommendation r given that they like a query g. This focus makes conventional ML metrics more relevant for evaluation.

Precision, recall, and F-1 score are all essential evaluation metrics for any supervised learning algorithm. [34, 41, 35] Precision calculates $\frac{TP}{TP+FP}$. Given that the model labels the instance as 1, how often is the model correct - how often is the recommendation $r \in S$? Recall calculates $\frac{TP}{TP+FN}$. Given that the label is 1 in actuality, how many 1s did the model accurately capture? More specifically, recall considers how many items $r_1, r_2, \ldots, r_n$ were recommended by the model, given that $\{r_1, r_2, \ldots, r_n\} \in S$. Finally, F-1 score is a metric that combines precision and recall. These metrics (which can be visualized with a confusion matrix) are valuable tools to understand how well the algorithm performs a trade-off between recommending all games it should and staying away from all games it should.

For a machine learning model in the recommender system domain, it makes sense to prioritize precision over recall. Precision involves the correctness of the overall recommendations. Given that the model labeled an instance as "recommend," how likely was that label to be true? Recall involves the correctness of the model in getting all instances that *should* have been recommended - how many true recommendations did it miss? People have a finite amount of time for entertainment. This means that a) they cannot possibly consume everything, so missing out on a theoretically good recommendation is less important and b) their time means more in general, so the displeasure of a bad recommendation is higher.

In addition to standard metrics of precision and recall, manual test cases are constructed and applied. These are not fundamental indicators of performance like precision, recall, and F-1 score, but they are informal, intuitive metrics which can indicate whether the model is leaning more towards precision or recall. Given an expectation of $SVM(g, r)$, does the test case conform to the expectation? Running these experiments allows for an informal assessment of whether the model errs on the side of high false positives (over-eager to recommend) or high false negatives (missing opportunities to recommend).

The minimum viable product for this project is a machine learning algorithm that outperforms random guessing. Assuming that the classes are balanced, an algorithm that always returns 1 will be correct 50% of the time. Outperforming random guessing with an F-1 score greater than 0.5 is a benchmark that indicates basic satisfiability for the model. With more time and resources, the goal for the project would be a machine learning algorithm that is more performant (around 0.8 F-1 score). The best case scenario would be an algorithm that is highly performant (around 0.9 F-1 score, which is typically seen in the literature) and explainable (where test cases conform to intuition from several human volunteers interested in video game recommendation).

6 Results and Discussion

Table 1 displays the standard ML metrics for this project.

Precision	0.825
Recall	0.697
F-1 Score	0.756

Figure 3 displays a confusion matrix of true positives, false positives, false negatives, and true negatives. The idea of the confusion matrix is to show whether the model tends to focus on accurately recommending everything that it should (and also being too eager to recommend items it should have stayed away from) or whether the model tends to be correct (at the expense of missing items it should have recommended). In recommendation, it makes sense to emphasize precision over recall because the displeasure of acting on a bad recommendation outweighs the displeasure of missing good recommendations.

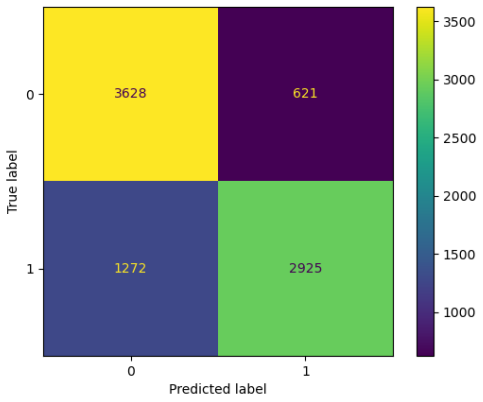

Figure 3: Matrix illustrating trade-off between precision and recall (more TP implies more FP)

Figure 4 displays the test cases for recommendation. These are intuitive indicators of performance that provide insight into whether the model tends to be over- or under-eager to label specific games as 1 (”should be recommended”). Because most of the games are labeled as 1, the model captures most of the games it should recommend, but it also recommends games it should stay away from. This corresponds to the eagerness to label a product as 1 which makes sense for a recommender system.

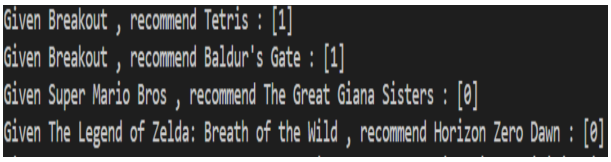

Figure 4: Qualitative test cases to assess if model tends towards either precision or recall (keeping trade-off in mind)

The results of hyperparameter tuning were also applied to further improve performance. Since the RBF kernel was used to create a nonlinear decision boundary, hyperparameter tuning was applied to different C-values (1, 10, 100, and 1000) to compare relative performances. A new SVM was created and fitted using 5-fold cross validation to compare the C-values across 5 different iterations. In every iteration, C=1000 performed the best. Table 2 displays the F-1 score of each C-value for these different iterations.

C=1	0.739	0.733	0.743	0.732	0.741
C=10	0.743	0.739	0.747	0.737	0.748
C=100	0.746	0.744	0.751	0.740	0.751
C=1000	0.750	0.747	0.756	0.743	0.755

Overall, the results make sense given the methodology used in the experiment. Both the table and the test cases suggest that precision is greater than recall - false positive count is less than false negative count. This implies that the model tends to be accurate when it labels a game as ”recommend,” but it misses good recommendations that it could have provided. In addition to conforming to the expectations of a recommender system, the trend towards precision over recall makes sense given the use of natural language. The word embedding vectors of deck and description were key features in training the SVM. Because the algorithm is comparing possible recommendations through natural language features, it is more likely to recommend matches where the deck/description vectors between games have strong matches (higher precision). It is also likely to be conservative in what it recommends if the natural language input does not clearly indicate a match between query g and recommendation r (lower recall).

Even with lower recall than precision, the natural language features used to train the model capture much of the variability of the dataset. Just by feeding in word embedding vectors of deck and description, the model is able to accurately reconstruct the GiantBomb API to an F-1 score of 0.7. Being able to beat the baseline model just with text features implies that text (whether matter-of-fact descriptions, reviews, or perhaps even more emotive text found on forums online) is a valuable feature for predicting game similarity.

On the other hand, one of the limitations of the approaches used in this experiment is that natural language features may not only *display* reduced recall, they may fundamentally *imply* reduced recall. Even if two games have the same demographics (genre, theme, and franchise), the way in which they are described in their deck and description supersedes any demographic connection two games may have. This hesitance to connect two games purely on demographics means that the model will have less positive labels (1s) in favor of negative labels (0s), thus increasing precision at the cost of recall. For example, consider recommending the first Super Mario Bros game given the original Donkey Kong arcade game. The genre (platformer) and franchise (Mario) are the same, but the descriptions may very significantly (”famous, influential arcade game” vs. ”blockbuster home console video game”). In this way, an algorithm trained purely on word embeddings will miss what is otherwise an accurate recommendation.

To address this limitation, other NLP approaches like dependency parsing or sentiment analysis can be applied. [22] Addressing cases where demographics are similar but descriptions are different with sentiment analysis can help improve performance. For example, suppose Gran Turismo is posed as a recommendation for Mario Kart. If the descriptions for both discuss similar driving physics, the social nature of the gameplay, and so on, the algorithm can apply NLP techniques (e.g., connecting adjectives to nouns) to see if the embeddings for *those* natural language strings are more similar. [31] In this way, NLP techniques can reveal similarities which unedited text did not show.

In general, the results meet the goals of the project. Because $SVM(g, r)$ simulates running $GB(g) = S$ and checking if $r \in S$, it provides the desired output. The SVM captures much of the variability of the GiantBomb API labels using this approach, so it is an appropriate reconstruction algorithm for the API.

7 Ethical Considerations

Any machine learning project is subject to ethical considerations which need to be addressed. Because this project uses machine learning to reconstruct the GiantBomb API, mitigating data bias is essential. Furthermore, any project that uses natural language, free-text data needs to consider how to moderate and mitigate abusive content if it is found.

Like any machine learning project, the model is only as good (or as biased) as the data on which it is trained. The SVM algorithm is trained on data from Metacritic, GameSpot, and GiantBomb. For the deck and description, the text data comes from verified reviewers who have write access to those websites. This content is less likely to be subject to bias, because it is a matter-of-fact text corpus that summarizes the content of a video game without opinion. However, variability among reviewers who write decks and descriptions may be an issue. This is addressed in future work.

For the review data, average video game players can submit review content. This content has more of a likelihood to skew the model because of "edge cases" like irony, misinformation, and speaker intention. [6] To address this possibility, multiple different sources were used instead of only using the GiantBomb API.

Abusive content in the free text reviews is also a possibility. Any application which uses natural language as a feature or output possibly has to contend with abusive content, which includes (but is not limited to) hate speech, casual racism/sexism/other discrimination, etc. Care has to be taken to make sure that this sort of speech is not learned and casually amplified by the model. With more time, one way to check for this would be to use common methods of removing hate speech from the dataset. One way to do so is to perform a search to ensure that any provided input string is not found within a centralized, curated hate speech dataset. [7] This is not currently implemented, but is an important direction of future work. As important as moderation is, however, it is less necessary in a project like this where data is obtained from websites that are about game *content* rather than game *discussion*. For example, GameSpot has lots of news about game news, development, updates, and so on, but it has less activity in forums where users congregate. By contrast, forums such as Reddit which have more game *discussion* are more likely to be vectors of this abusive content. This means that abusive content is less likely to be obtainable in this project, but it is still worth considering and implementing if more time was available.

8 Future Work, and Conclusion

Future work can improve the model performance by ensuring that the model does not propagate bias further in its output. In terms of ethical considerations, future work includes using standard approaches of abusive content detection [7] to process such content as needed. Even without ethical considerations, the way in which decks and descriptions are written is still subject to variability. Even amongst the same API, different authors may have written the text items in different ways, and this introduces a new layer of complexity. Future work could mitigate this uncertainty by getting deck and description elements for each game from different APIs (rather than just one API per game as in the current architecture).

Model training is another area where future work can continue to improve performance. Currently, word embedding vectors of deck and description are the influential features used to train the SVM algorithm. Future work could include dependency parsing approaches on different sorts of text to get a better sense of syntactic structure. [28] Instead of only considering adjective-noun pairs, further grammatical structure (adverb-verb pairs, other dependency tags, etc.) could be explored. Other features of sentiment analysis could also be used; topic modeling, sentence polarity, and other feature extraction methods could also potentially improve performance. [10, 21] Any NLP method that focuses on extraction or identification provides informatic features for the model to train on to further contextualize its word embedding focus.

A Replication Instructions

A.1 Software Used

Key software used by the project is listed below.

- spacy=3.7.2 (for dependency parsing analysis of adjective/context word pairs)
- tensorflow=2.14.0 and tensorflow-hub=0.15.0 (setting up the TensorFlow Universal Sentence Encoder for word embedding vector generation)
- imbalanced-learn=0.11.0 (performing synthetic minority oversampling to offset class imbalance)
- pandas=1.5.3 (setting up DataFrames)
- numpy=1.25.0 (using numpy arrays for word embedding data types)
- scikit-learn=1.2.2 (for the SVM algorithm, pipeline, evaluation, etc.)

A.2 How to Run

- obtain API keys from GiantBomb and GameSpot
- navigate to a directory on your computer, and run:

```
git clone https://github.com/y0shK/comps-video-game-recommender.git
```

- create .env in the same directory as recommendation.py by using the following command:

```
touch .env
```

- put API keys into .env
- download metacritic_game_info.csv and metacritic_game_user_comments.csv from Kaggle, and put both files in same directory as recommendation.py
- create conda environment in the same directory

- run:

```
conda create --name YOUR_ENV --file
    requirements.txt
```

- run:

```
python recommender.py
```

B Code Architecture

B.1 Dataset

B.1.1 Initial setup

Relevant imports are made (as described in the Replication Instructions section). For the data, a requests_cache session for API calls is created and CSV files for Metacritic data are read in. A web model for Spacy dependency parsing through adjective/context word pairs is also instantiated.

B.1.2 Data collection

From the CSV titles, game titles are mapped to reviews, deck/description, and demographic information. This hash table of reviews is joined with API call data (from GameSpot and GiantBomb) to form the query set of data, and reviews are also mapped to when available.

B.1.3 Dataset collation

The TensorFlow Sentence Encoder is set up for word embedding generation. Iterating through the query set, the query game g is mapped to every possible recommendation r. The API is called to generate S, where S is a set of "similar games" to g. Then checking if $r \in S$ maps the word embedding vectors of deck/description for both g and r to 0 (not similar) or 1 (similar). Adjective-context word pairs are also found using Spacy dependency parsing, and key words of adjectives and their respective nouns are fed into the string that is converted to a word embedding. If reviews are present in the current g or r, they are tokenized and added to the string before word embedding as well.

B.2 Algorithm

B.2.1 Synthetic minority oversampling

To offset the class imbalance, synthetic minority oversampling [33] is used to generate similar points in N-space. Upon resampling, the data is shuffled to prepare for training.

B.2.2 Training

The dataset is split into train and test sets with an 80-20 split. An SVM instance is created by using principal component analysis so that the decision boundary is simpler. The SVM model is trained on the train set.

B.2.3 Evaluation

The SVM model is evaluated on the test set. A confusion matrix is generated to visualize the tradeoff between precision and recall.

B.3 Postprocessing

B.3.1 Data exploration

Hyperparameter tuning is conducted by creating a new SVM classifier and varying C-values. 5-fold cross validation is used and results are compared. Test cases are executed and visuals are created to get a sense of the demographic characteristics of the dataset.

References

[1] . *GameSpot API*. https://www.gamespot.com/api/. Online; accessed 12 December 2023.

[2] . *GiantBomb API*. https://www.giantbomb.com/api/. Online; accessed 12 December 2023.

[3] Agata Nawrocka, Andrzej Kot, and Marcin Nawrocki. *Application of machine learning in recommendation systems*. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8399650&tag=1. Online; accessed 3 December 2023. 2018.

[4] Anqi Zhang. *Sentiment without Sentiment Analysis: Using the Recommendation Outcome of Steam Game Reviews as Sentiment Predictor*. https://openprairie.sdstate.edu/cgi/viewcontent.cgi?article=1487&context=etd2. Online; accessed 2 December 2023. 2022.

[5] Anthony Chow, Min-Hui Nicole, Foo, and Giuseppe Manai. *HybridRank : A Hybrid Content-Based Approach To Mobile Game Recommendations*. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=05c96a297534a5bbf4980c71f58e0cd503a43cc2#page=16. Online; accessed 4 March 2023. 2018.

[6] Bertie Vidgen, Leon Derczynski. *Directions in abusive language training data, a systematic review: Garbage in, garbage out*. https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0243300#sec030. Online; accessed 9 April 2023. 2020.

[7] Binny Mathew, Punyajoy Saha, Seid Muhie Yimam, Chris Biemann, Pawan Goyal, Animesh Mukherjee. *HateXplain: A Benchmark Dataset for Explainable Hate Speech Detection*. https://ojs.aaai.org/index.php/AAAI/article/view/17745. Online; accessed 12 April 2023.

[8] Brent Smith, Greg Linden. *Two Decades of Recommender Systems at Amazon.com*. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7927889&tag=1. Online; accessed 30 November 2023. 2017.

[9] Brian Choi, Misun Ryu, Dharmesh Panchal, Andrew Le. *Building Recommender Systems for Video Games on Steam*. https://library.ucsd.edu/dc/object/bb5021836n/_3_1.pdf. Online; accessed 27 March 2023.

[10] Bushra Ramzan, Imran Sarwar Bajwa, Noreen Jamil, Riaz Ul Amin, Shabana Ramzan, Farhan Mirza, and Nadeem Sarwar. *An Intelligent Data Analysis for Recommendation Systems Using Machine Learning*. https://www.hindawi.com/journals/sp/2019/5941096/. Online; accessed 8 December 2023. 2019.

[11] Camila V. Sundermann, Jo˜ao Antunes, Marcos A. Domingues and Solange O. Rezende. *Exploration of Word Embedding Model to Improve Context-Aware Recommender Systems*. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8609619&tag=1. Online; accessed 3 December 2023. 2018.

[12] Carlos A. Gomez-Uribe, Neil Hunt. *The Netflix Recommender System: Algorithms, Business Value, and Innovation*. https://dl.acm.org/doi/pdf/10.1145/2843948. Online; accessed 30 November 2023. 2015.

[13] Corinna Cortes, Vladimir Vapnik. *Support-Vector Networks*. https://link.springer.com/content/pdf/10.1007/bf00994018.pdf. Online; accessed 3 December 2023. 1995.

[14] Dahlia Ma. *Metacritic Video Game Comments*. https://www.kaggle.com/datasets/dahlia25/metacritic-video-game-comments. Online; accessed 12 December 2023.

[15] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, Ray Kurzweil. *Universal Sentence Encoder*. https://arxiv.org/pdf/1803.11175.pdf. Online; accessed 2 December 2023. 2018.

[16] Donghui Wang, Yanchun Liang, Dong Xu, Xiaoyue Feng, Renchu Guan. *A content-based recommender system for computer science publications*. https://www.sciencedirect.com/science/article/pii/S0950705118302107. Online; accessed 2 December 2023. 2018.

[17] Elham Asani, Hamed Vahdat-Nejad, Javad Sadri. *Restaurant recommender system based on sentiment analysis*. https://www.sciencedirect.com/science/article/pii/S2666827021000578. Online; accessed 3 December 2023. 2021.

[18] Faezeh Sadat Gohari, Hassan Haghighi, Fereidoon Shams Aliee. *A semantic-enhanced trust based recommender system using ant colony optimization*. https://link.springer.com/article/10.1007/s10489-016-0830-y. Online; accessed 5 March 2023. 2016.

[19] Hdioud Ferdaous, Frikh Bouchra, Ouhbi Brahim, Mountasser Imad-eddine and Benghabrit Asmaa. *Recommendation using a clustering algorithm based on a hybrid features selection method*. https://link.springer.com/article/10.1007/s10844-017-0493-0. Online; accessed 4 March 2023. 2018.

[20] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, and Taylor Van Vleet. *The YouTube Video Recommendation System*. https://dl.acm.org/doi/pdf/10.1145/1864708.1864770. Online; accessed 30 November 2023. 2010.

[21] James Owen Ryan, Eric Kaltman, Michael Mateas, and Noah Wardrip-Fruin. *What We Talk About When We Talk About Games: Bottom-Up Game Studies Using Natural Language Processing*. https://users.soe.ucsc.edu/~jor/publications/ryanGamesStudiesUsingNLP.pdf. Online; accessed 4 March 2023.

[22] James Owen Ryan, Eric Kaltman, Timothy Hong, Michael Mateas, and Noah Wardrip-Fruin. *People Tend to Like Related Games*. https://eis.ucsc.edu/papers/ryanEtAl_

PeopleTendToLikeRelatedGames . pdf. Online; accessed 4 March 2023.

[23] Javier Pérez-Marcos, Lucía Martín-Gómez, Diego M. Jiménez-Bravo, Vivian F. López, María N. Moreno-García. *Hybrid system for video game recommendation based on implicit ratings and social networks*. https://link.springer.com/article/10.1007/s12652-020-01681-0. Online; accessed 4 March 2023. 2020.

[24] Jin An Xu and Kenji Araki. *A SVM-based Personal Recommendation System for TV Programs*. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1651358. Online; accessed 3 December 2023. 2018.

[25] Junliang Yu, Hongzhi Yin, Xin Xia, Tong Chen, Jundong Li, and Zi Huang. *Self-Supervised Learning for Recommender Systems: A Survey*. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10144391&tag=1. Online; accessed 1 December 2023. 2022.

[26] Krishnaram Kenthapadi, Benjamin Le, Ganesh Venkataraman. *Personalized Job Recommendation System at LinkedIn: Practical Challenges and Lessons Learned*. https://dl.acm.org/doi/pdf/10.1145/3109859.3109921. Online; accessed 30 November 2023. 2017.

[27] Kurt Jacobson, Vidhya Murali, Edward Newett, Brian Whitman, and Romain Yon. *Music Personalization at Spotify*. https://dl.acm.org/doi/abs/10.1145/2959100.2959120. Online; accessed 8 December 2023. 2016.

[28] Luigi Di Caro, Matteo Grella. *Sentiment analysis via dependency parsing*. https://www.sciencedirect.com/science/article/abs/pii/S0920548912001237. Online; accessed 8 December 2023. 2013.

[29] Luong Vuong Nguyen, Tri-Hai Nguyen, Jason J. Jung, and David Camacho. *Extending collaborative filtering recommendation using word embedding: A hybrid approach*. https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.6232. Online; accessed 2 December 2023. 2021.

[30] Melania Berbatov. *Overview on NLP Techniques for Content-Based Recommender Systems for Books*. https://aclanthology.org/R19-2009.pdf. Online; accessed 3 December 2023. 2019.

[31] Michael Meidl, Steven Lytinen, and Kevin Raison. *Using Game Reviews to Recommend Games*. https://ojs.aaai.org/index.php/AIIDE/article/view/12752/12600. Online; accessed 2 December 2023.

[32] Myrthe Reuver, Antske Fokkens, and Suzan Verberne. *No NLP Task Should be an Island: Multidisciplinarity for Diversity in News Recommender Systems*. https://aclanthology.org/2021.hackashop-1.7.pdf. Online; accessed 3 December 2023. 2021.

[33] N. V. Chawla K. W. Bowyer L. O. Hall W. P. Kegelmeyer. *SMOTE: Synthetic Minority Oversampling Technique*. https://www.jair.org/index.php/jair/article/view/10302. Online; accessed 3 December 2023. 2002.

[34] Nosayba Al-Azzam, Ibrahem Shatnawi. *Comparing supervised and semi-supervised Machine Learning Models on Diagnosing Breast Cancer*. https://www.sciencedirect.com/science/article/pii/S2049080120305604. Online; accessed 8 December 2023. 2021.

[35] P. Sujatha and K. Mahalakshmi. *Performance Evaluation of Supervised Machine Learning Algorithms in Prediction of Heart Disease*. https://ieeexplore.ieee.org/document/9298354. Online; accessed 8 December 2023. 2020.

[36] Paolo Cremonesi, Yehuda Koren, Roberto Turrin. *Performance of recommender algorithms on top-n recommendation tasks*. https://dl.acm.org/doi/10.1145/1864708.1864721. Online; accessed 8 December 2023. 2010.

[37] Pawel Matuszyk, Myra Spiliopoulou. *Stream-based semi-supervised learning for recommender systems*. https://link.springer.com/article/10.1007/s10994-016-5614-4. Online; accessed 1 December 2023. 2017.

[38] *Quantic Foundry Recommender*. https://apps.quanticfoundry.com/recommendations/gamerprofile/videogame/. Online; accessed 3 March 2023.

[39] Rafet Sifa, Raheel Yawar, Rajkumar Ramamurthy, Christian Bauckhage, and Kristian Kersting. *Matrix and Tensor Factorization for Game Content Recommendation*. https://link.springer.com/article/10.1007/s13218-019-00620-2. Online; accessed 4 March 2023. 2019.

[40] Sandeep Tata, Jignesh M. Patel. *Estimating the selectivity of tf-idf based cosine similarity predicates*. https://dl.acm.org/doi/abs/10.1145/1328854.1328855. Online; accessed 8 December 2023. 2007.

[41] Shovan Chowdhury and Marco P. Schoen. *Research Paper Classification using Supervised Machine Learning Techniques*. https://ieeexplore.ieee.org/document/9249211. Online; accessed 8 December 2023. 2020.

[42] *Steam Interactive Recommender*. https://store.steampowered.com/recommender/?snr=1_4_4_. Online; accessed 3 March 2023. 2020.

[43] Thorsten Joachims. *Text Categorization with Support Vector Machines: Learning with Many Relevant Features*. https://wiki.eecs.yorku.ca/course_archive/2013-14/W/6339/_media/joachims_98a.pdf. Online; accessed 3 December 2023. 2005.

[44] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean. *Efficient Estimation of Word Representations in Vector Space*. https://arxiv.org/pdf/1301.3781.pdf. Online; accessed 2 December 2023. 2013.

[45] Xin Xin, Alexandros Karatzoglou, Ioannis Arapakis, and Joemon M. Jose. *Self-Supervised Reinforcement Learning for Recommender Systems*. https://dl.acm.org/doi/pdf/10.1145/3397271.3401147. Online; accessed 1 December 2023. 2020.