Machine Learning Project:
Steam™ Game Reviews
Baizhou Hou, Ruiqi Tao

Introduction. Machine Learning is a big topic that has carved out a sizable niche between Math and Computer Science. Over the duration of the course, we have covered many different Machine Learning models and methods and the theory behind them, yet one aspect that isn't very intuitive despite all this background is the use of non-numerical data. Intuitively, it makes sense that with a sufficiently large dataset, correlation can be found within just about anything. In this project, we attempt to look at a data set that not only is composed primarily of non-numeric data, but also is arguably highly subjective: Video Game Reviews.

The Dataset. Though we intended originally to scrape the data via Steam's own store API endpoint and some scraping for the rest of the data, actually building the crawler proved quite troublesome. Though the steam API works well, there is little motivation to build our own dataset if we cannot congregate the scraped data into it. Ultimately, we used a dataset found on Kaggle: "Steam Games Complete Dataset."

The dataset initially consisted of just over fourty thousand rows, most of which had to be removed in the cleaning process. Between games with few reviews, those without descriptions, and DLC/OST and other content, a big chunk of the data simply wasn't usable for training for this project. The dataset used ended up being just over eleven thousand entries, all games with enough user reviews to have an all-time user positive review percent(hereby referred to as "review score") and descriptions snippets around one paragraph long.

Baselining. For any Machine Learning endeavor, it is important to establish a baseline to which different models and features can be compared against. For this project, we chose two standards to baseline against: one where the predicted output is just the mean of the training input, the other using a simple linear regression with basic features including genres, user tags, achievements, and game prices, with genres and usertags being considered via one-hot encoding. For this project, accuracy is evaluated with mean absolute error.

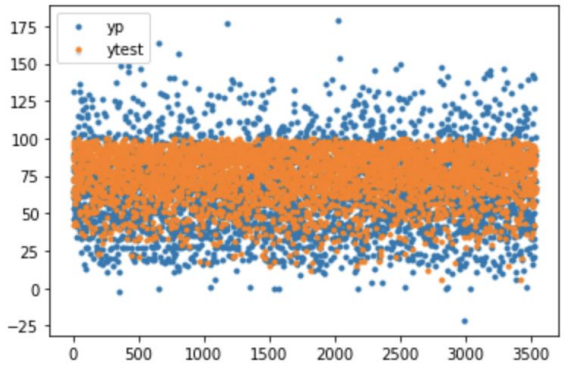

Fig. 1: Linear Baseline

This model yields a mean absolute error of 26.4, or that on average, the estimated review score is 26.4% off from the actual review score. Figure 1 shows both the predicted and actual testing values. Note that the model is not normalized and so many scores actually exceed 100% positive review, which is obviously not possible. That said, although this is a huge contributor to this model's inaccuracy, we leave it as is for fear of changing the nature of the model. After all, it is merely a baseline.

For the next baseline, we simply guess the predicted output as the average of the training input. For this naive approach to baselining, we find that the mean error is just 13.6, far better than the previous model shown in figure 1. This second baseline is shown in figure 2.

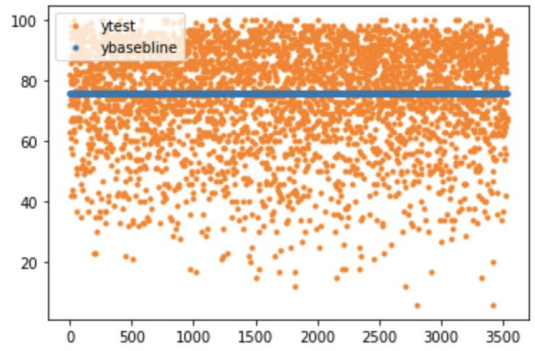

Fig. 2: Simple Baseline

The two baselines clearly illustrate the difficulty in working with a dataset like game reviews: high variance data where games with similar tags, price, etc, can have wildly different review scores. Indeed, the simplest baseline yielded a far better result than trying to predict the review scores of games based on easily categorizable features like price and user-defined tags.

More Features. One of the motivations for trickling in features into the model is to see if adding more features necessarily has a positive effect on the model's performance. In the next model, we tried to use more features for training the model. The dataset leaves much to interpretation when it comes to what features to add and how to represent them in the feature matrix. In this case, we added the game's supported languages to the list of features. Like user-defined tags, this is added in a one-hot encoded format: many games support many languages.

There are a few surprising results when languages were added, the first being that the variance has decreased significantly with this new addition. In part, this resulted from the fact that very few games are now predicted to be above the 100 user review score. This is fascinating, as it is hard to intuitively predict that adding a feature that is not obviously correlated with the games' quality can have such a large effect on the overall performance of the model, dropping the mean absolute error to just under 14.

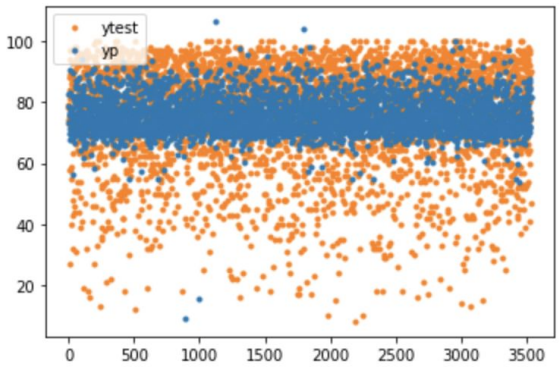

Fig. 3: Linear Model with Languages Added

Note that in figure 3, very few titles score above a 90, and only a countable number of cases exceed 100. Without any sort of normalization, this is really an unexpected outcome from simply adding language to the model. However, we also do note that the variance of the predicted vector to be far lower than that of the training one, which is not the case in the baseline. The model generally predicts very close to the average of just under 80, and fails to predict many scores under 50, even though this occurrence is not so uncommon in reality.

A plausible cause for this "tightening" of the output space, so to speak, is hard to pinpoint. What can be said is that the performance of this model is hard to comment on: just like the "simple baseline" model in figure 2 where the average from the training data is used as the predicted output, even though the mean absolute error is low, the model can hardly be said to actually be "predicting" the game review scores; rather, the outputs all happen to be close to the mean and thus the error presents as low. This also illustrates that simply looking at the error is a very poor way to evaluate the performance of a model, and this point is made sufficiently clear when looking at the outcome graphically.

Word Embeddings. Putting numbers to words can be rather difficult. That is, in this dataset, there is a massive datapoint that has not been exploited: the games' descriptions. On the Steam™ store, each game gets a small blurb to market itself to the potential buyer. This is the first thing that one might see when opening the store page for a game, along with the title and a gallery of some screenshots for the game. As important as this piece of information is, words can be hard to abstract into numbers that are useful in the context of training our machine learning model. Word embeddings is a learned representation for words by mapping their "meanings" onto some n-dimensional vector. In this case, we use Gensim's implementation of Word2Vec for building our vocabulary and training the context.

The Word2Vec Model. Word embeddings is a generally complicated model with implementation details that likely far exceed the scope of this project; but to cover the basics, it is a shallow neural network with an arbitrary number of hidden-layer neurons that attempts to infer meaning by using a "pivot" word and its neighboring "context" words. The big idea is that in a well-trained Word2Vec model, similar words will have n-dimensional vector representations that are near each other in this higher-dimension space.

Gensim's implementation of the Word2Vec model uses the continuous bag-of-words model by default, training by using the pivot word to infer the context words to extrapolate the n features defined in the training parameters. Other training parameters include learning rate and context window, which all have a significant effect on the model itself. Although the accuracy of such a model can be rather subjective, we tweaked the model parameters until we felt that it represented many commonly seen words in game descriptions well.

Source: https://pathmind.com/wiki/word2vec

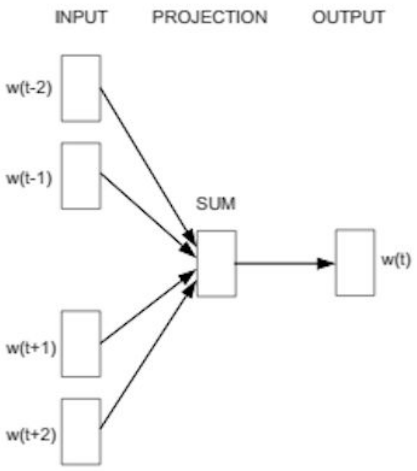

Fig 4: Word2Vec CBOW Model

When choosing whether to use CBOW or Skip-Gram, we note that training results are generally more accurate on larger datasets with the CBOW method. Since our dataset consists of twenty-four thousand paragraphs and a final vocabulary of more than 9000 words after preprocessing, the CBOW method seems like the better model.

We investigate the performance of our Word2Vec model very loosely and base it on a number of commonly seen words. Since the Word2Vec model trains pretty quickly thanks to multi-threading and the power of Cython, it was easy to tweak the training parameters and compare the results. Gensim also conveniently provides the ability to export the final model as a binary file so that it can be accessed easily without needing to be retrained when opened from a different python script. The details of the "failed" models will be omitted, but our perhaps curious findings will be summarized thus: if the minimum learning rate is too small, the results will be generally poor. When the window is too large, the most similar words in our human perception tend to be in the middle range of similarity in the mode. That said, it still has reasonable accuracy overall. This is the case even though the data is preprocessed to not have

so-called “stopwords” like “of” and “for” removed. Regarding the dimensionality of the vector space onto which the words are mapped: we noted that increasing the number of features beyond a sufficiently large number had minimal or negative effect on the model. Though Gensim documentation recommends 200~500 features, we found very little reason to use that many features. We settled on 150 features for a reasonably performing Word2Vec model that will not slow down or overall model much.

The final trained Word2Vec model, despite having some weird outcomes, actually presented with surprisingly accurate results against some of the words most commonly seen in the game descriptions. For example:

most similar to intense:
frantic, fastpaced, explosive
most similar to monster:
demons, harmful, devils
most similar to card:
collectible, tabletop, strategic

Further, word combinations also yield interesting results, even though they are not used in the overall model, a few amusing ones are illustrated here:

war + sad: ussr, templar, feud
dark + magic: supernatural, occult

Word combinations are an interesting way to analyze the model’s accuracy, and although the war+sad example is amusing, we note that certain combinations like dark+magic can be great for examining how well the model really captures the larger “meaning” behind the words. Ultimately, I think this model does a good job.

Integrating Word2Vec. With the Word2Vec model finally trained, it is now time to integrate it into the overall model. For this, we implemented a simple abstraction for our Word2Vec model that takes the first five “meaningful” words of the game’s description and vectorizes them. Then, the vector mean is used, adding 150 features to each entry in our dataset. Now, the model is retrained using the new feature matrix.

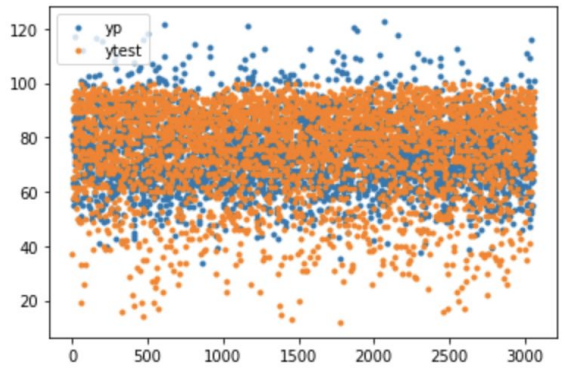

Fig 5: Linear Model with Word Embeddings sans Language Features

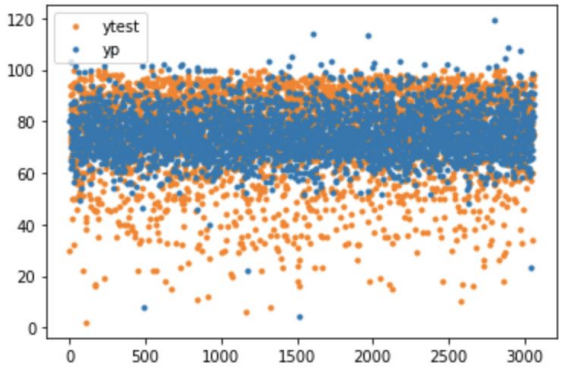

Fig 6: Linear Model with Word Embeddings and Language Features

Adding word embeddings to the simple linear baseline model resulted in a mean error of 14, and again adding language features to that reduced the mean error further to just above 12. We note in this case that adding more features does indeed reduce the error, yet when looking at the graphs we again see a tightening of data variance when the language features are added. Truly fascinating. Considering the

above two models graphically, it is hard to argue that overall, figure 5's model does a better job at filling in the reviews of lower scores. The larger error most likely results from extremely low deviation in the training and testing data, making outliers hard to account for and thus leading to an inability to predict too far from the mean. That said, the model with both word embeddings and languages ultimately performed the best when looking only at the mean absolute error.

Conclusion. In the end, the project is more than anything an exercise to see the ability for basic machine learning approaches to be able to predict an outcome that is, by every definition of the word, *unpredictable*. *Gears 5* is a massive triple A title that is reviewed poorly not because of anything related to the quality of the game but rather because of players unhappy about censorship in certain regions. Similarly, a game like *Metro Exodus* faces massive controversy for its limited-time exclusivity with Epic while another game, *HADES*, with a similar deal has overwhelmingly positive reviews with not a comment in sight about its own exclusivity deal. Even games from the same franchise can have massively differing scores.

The result is not too different from what is expected: regardless of the data extrapolated from the store or the model used, there are always going to be elements that are hard to predict. That said, the exploration of Word2Vec proved to be quite fascinating and is definitely an area that can be expanded upon. Perhaps hardware requirements can be recognized when training a specific Word2Vec model. Or maybe user-defined tags and genres can have more meaning extrapolated from them by using a Word2Vec model. Moreover, even our method of selecting words for vectorization is arbitrary and likely leads to a poor selection of word features on many titles. There are probably many places that the project can still be taken to to improve performance, but what we initially set out to do is done: using non-numeric features to make predictions on nigh unpredictable outcomes. In the end, the mean error was a bit lower than the baseline, and although the actual accuracy of that model is debatable, that is a result as good as we could have hoped for.

References

Overview of Word2Vec
https://pathmind.com/wiki/word2vec

Gensim Documentation
https://radimrehurek.com/gensim/auto_examples/index.html

The Dataset
https://www.kaggle.com/trolukovich/steam-games-complete-dataset

Course Website: Lectures, Labs, Demos
http://www.chrismusco.com/introml/