Categorizing Boardgames based on Descriptions

GitHub Repository

Kristiyan Dimitrov
kristiyandimitrov2020@u.northwestern.edu

Abstract

This paper presents work done on building a model to perform multilabel classification for boardgame categories based on their descriptions. More specifically, it presents performance metrics for multiple different data preprocessing & model combinations. The best performing model – a linear Support Vector Classifier - is productionized as a REST API and made available in this GitHub repository.

1 Introduction

The website BoardGameGeek.com is often referred to as the IMDb of boardgames. It contains data such as ratings, descriptions, and many more details on tens of thousands of boardgames. This paper concerns itself with the multi-label classification task of determining the category labels for boardgames based on their descriptions.

There are at least two benefits to such an application. First, new boardgames that appear on the website can immediately be tagged, without waiting for user generated input. Second, the model developed in this work can be applied to existing boardgames in the database to identify missing labels. This would have the added benefit of making more boardgames easier to discover by users searching for a specific category of boardgame.

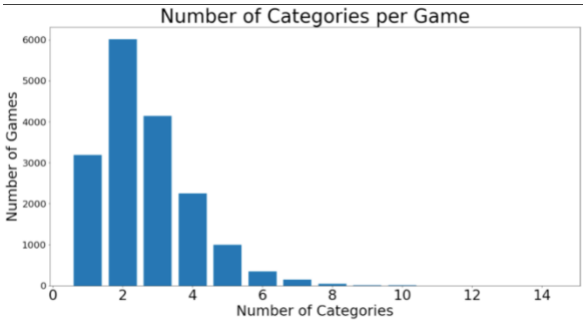
2 Related Work

There is a plethora of related work that can be found on the Internet. Multi-label classification is a popular Machine Learning task and has been performed on many different datasets, including IMDb mentioned earlier. However, I was not able to find any work pertaining to this specific dataset.

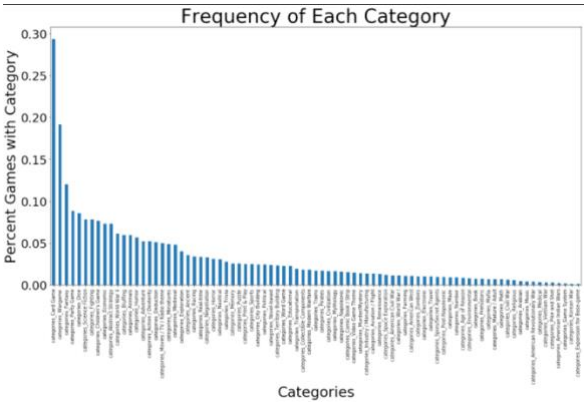
3 Dataset

BoardGameGeek.com provides an XML API for retrieving data from their database. Due to difficulties in working with XML, I have instead used a Python package called boardgamegeek, which provides a JSON based wrapper for calling the API. In other words, it allows the user to call the XML API and converts the response into JSON. The code for retrieving the data is part of another project, which can be found here.

This work starts with the data/raw.json file. It contains data on 17,147 boardgames. More specifically, each boardgame has a description and category information. There are 83 unique categories with 2.64 categories per game on average.



Only 15 out of the 83 categories appear in more than 5% of the boardgames. In other words, most of the categories are fairly rare.



The descriptions were preprocessed using gensim.utils.simple_preprocess. Explicitly, this means they were tokenized, converted to lowercase, and any tokens containing only 1 character or more than 15 characters were removed. Finally, all stopword tokens were dropped with reference to the nltk English stopwords corpus. The resulting descriptions have an average length of 105 tokens, a total number of 1,805,528 tokens, and a dictionary length of 60,429.

3.1 Method – Feature & Target Generation

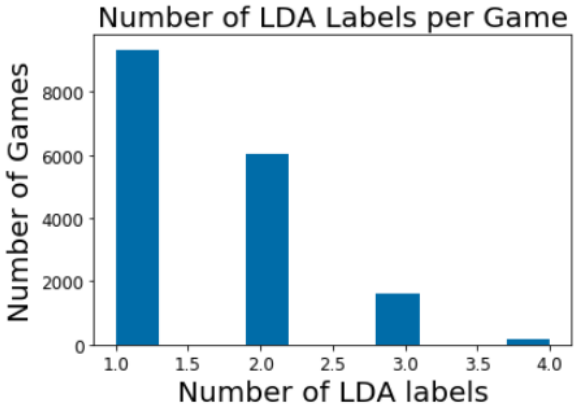
The method of feature & target generation can be just as important as the model chosen for a supervised learning task.

To that end, I have tested four different feature generation approaches:

- Bag of Words with unigrams (BOW-1)
- Bag of Words with uni & bigrams (BOW-2)
- Tf-Idf with unigrams (TFIDF-1)
- Tf-Idf with uni & bigrams (TFIDF-2)

All the above corpora were "pruned" by removing any features that appear in more than 50% of all documents and in fewer than 100 documents. The resulting unigram corpora have 2,261 features and the uni & bigram corpora have 2,773 features.

Moreover, I tested Latent Dirichlet Allocation (LDA) as a means of reducing the number of categories from 83 to 20. The 20 LDA scores for each game are in the range [0, 1]. I converted them to binary values by calculating the average score for each game and labelling all scores above that average as a 1 while all scores below the average I labelled as 0. As a result, most games now had 1 or 2 LDA labels, with some having 3 or 4.



As an interesting aside, I also tried Latent Semantic Analysis (LSA) and Feature Agglomeration (FA). LSA is based on Truncated Singular Value Decomposition of the one-hot encoded matrix representing the categories for each boardgame. FA is similar to Agglomerative Clustering, but it clusters features instead of samples. I specified 20 clusters using cosine affinity and average linkage. The challenge with both LSA & FA, as with LDA, was in finding a meaningful way to convert scores to binary values.

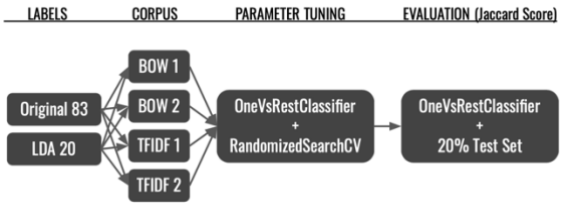
3.2 Method – Non-Deep Learning Models Evaluation & Parameter Tuning

All non-Deep Learning models were tested on all four corpora mentioned in the previous section. For each corpus, I tuned the model hyperparameters by performing 10 iterations of 3-fold cross-validation via RandomizedSearchCV. This accomplishes 2 goals: finding the best hyperparameters for each model and also finding which corpus works best. Models were scored using micro Jaccard Score to address for the imbalance in category representation and emphasize the importance of identifying 1s (this is a characteristic of the Jaccard Score). Finally, once the best corpus & parameters were identified, I compared all models based on their Jaccard Scores on the same 20% holdout test

set. Additional metrics I considered are: AUC, F1, Precision, Recall, Hamming Loss, and Zero-One Loss.

It is important to note that the Random Forests Classifier inherently supports multi-label tasks. However, the same is *not* true for all other models I tested. For them I used the OneVsRestClassifier from the sklearn library.

The entire process can be summarized in the below graph:



The models I tried are: Random Forests, Logistic Regression, Linear Support Vector Classifier, Radial Basis Support Vector Machines.

3.3 Method – Deep Learning Models

My deep learning models were not trained on the previously described corpora. I kept the simple_preprocess and stopword removal steps, but converted to tokens via the Keras Tokenizer class, its texts_to_sequences method and pad_sequences function with 'post' padding.

I tried multiple different combinations of fully connected layers, Convolutional layers, LSTMs, and fine-tuned BERT. I also added Dropout and Embedding layers to attempt to improve performance.

3.4 Results – Non-Deep Learning Models

I tried multiple different combinations of fully connected layers, Convolutional layers, LSTMs, and fine-tuned BERT. I also added Dropout and Embedding layers to attempt to improve performance.

Non-Deep Learning Model Results				
Model	**Best Param.**	**Best Corpus**	**Labels**	**Jaccard Score**
Random Forests	N_trees = 230	TFIDF 2	Original 83	0.1948
Radial SVM	-**	TFIDF 2	Original 83	0.265
Logistic Regression	C* =.5	BOW 2	Original 83	0.3654
Linear SVC	C = 1	TFIDF 2	Original 83	**0.3878**
Logistic Regression	C =.5	BOW 2	LDA 20	0.3024
Linear SVC	C = 1	TFIDF 2	LDA 20	0.3269

*The parameter C is inversely proportional to regularization strength

**Radial SVM took too long to train and performance was poor so I did not tune parameters.

3.5 Results - Deep Learning Models

All models end with a fully connected, 83 node output layer with a sigmoid activation function and binary cross-entropy loss. This is necessary for the multi-label classification. Moreover, each model was trained for 10 epochs; further training did not show performance improvements.

#	Deep Learning Models - Architecture
1	D(200) > D(200)
2	Emb(128) > Conv1D(32,8) > MP1D(2) > FL > D(100)
3	Emb(128) > Conv1d(32,8) > MP1D(2) > FL > D(100) > D(100)
4	Emb(128) > Conv1d(32,8) > MP1D(2) > FL > Dr (.3) > D(100) > Dr (.3) > D(100)
5	Emb(128) > Conv1d(32,8) > MP1D(2) > FL > Dr (.3) > D(100) > Dr (.3) > D(100) > Dr (.3) > D(100)
6	Emb(128) > Conv1d(32,8) > MP1D(2) > FL > Dr (.3) > D(200) > Dr (.3) > D(100) > Dr (.3) > D(100)
7	Emb(128) > Conv1d(32,8) > MP1D(2) > FL > Dr (.3) > D(500) > Dr (.3) > D(200) > Dr (.3) > D(100) > Dr (.3) > D(100)
8	Emb(256) > Conv1d(32,8) > MP1D(2) > FL > Dr (.3) > D(500) > Dr (.3) > D(200) > Dr (.3) > D(100) > Dr (.3) > D(100)
9	Emb(128) > LSTM(128) > Dr(.5) > LSTM(64) > Dr(.5)
10	Fine-tuned small BERT model

- Conv1D(x,y) – Convolutional layer with x filters and y kernel size.

- LSTM(x) – Long Short-Term Memory with x nodes
- D(x) – Dense layer with x nodes
- Emb(x) – Embedding with x size
- MP1D(x) – MaxPool 1D with kernel size x
- FL – Flatten layer
- Dr(x) – Dropout layer with probability x

Deep Learning Models - Performance			
#	Jaccard Score*	AUC**	F1
1	-	-	-
2	-	0.8445	-
3	-	0.8634	-
4	-	0.8634	-
5	-	0.8588	-
6	-	0.8634	-
7	-	0.8634	0.473
8	0.3124	0.8548	0.4761
9	-	0.7782	-
10	-	0.6409	-

* I compared the deep learning models based on multiple metrics, not just Jaccard Score. These metrics are not all reported here for brevity. The best performing networks were the ones with embedding and Conv1D layers, but their performance in terms of Jaccard Score, AUC, and F1 was not comparable to the non-Deep Learning models.

** When calculating AUC, I tested multiple different threshold values for converting output layer activations to binary outputs. The best threshold was consistently 0.3.

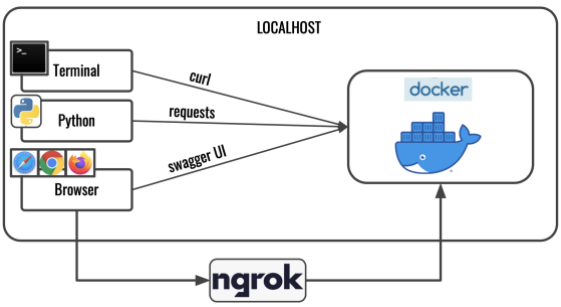
3.6 Discussion

Most notably, in the end the best performing model was the Linear Support Vector Classifier. It trains fast and performed best based on Jaccard Score and some of the other metrics mentioned. This supports the rule of thumb "when in NLP doubt do SVM". It is also somewhat logical in this case due to the size and dimensionality of our dataset.

Furthermore, I experienced first-hand how slow LSTMs train. They are sequential in nature and therefore the training process cannot be parallelized. Consequently, they cannot be feasible trained on large datasets. Ultimately, this hurts their performance.

Finally, it is worth noting that BERT did not perform very well. Moreover, without the use of Google Colab GPUs, training it would not have been at all possible on my laptop CPU. This supports our previous discussions that modern deep learning applications heavily rely on massive datasets and extensive computational resources.

As stated in the beginning, the final LinearSVC model can be retrieved and exposed as a REST API from the project GitHub repository. Once the Docker image is built, there are multiple connectivity options as described in the README.md and shown in the diagram below:



References

https://boardgamegeek.com

https://scikit-learn.org/stable/

https://radimrehurek.com/gensim/