|  |
| --- |
| **Categorizing Boardgames based on Descriptions**  [**GitHub Repository**](https://github.com/MSIA/kristiyan-dimitrov_msia_text_analytics_2020/tree/project) |
|  |
| **Kristiyan Dimitrov**  **kristiyandimitrov2020@u.northwestern.edu** |
|  |
|  |
|  |
|  |

Abstract

|  |
| --- |
| 000  001  002  003  004  005  006  007  008  009  010  011  012  013  014  015  016  017  018  019  020  021  022  023  024  025  026  027  028  029  030  031  032  033  034  035  036  037  038  039  040  041  042  043  044  045  046  047  048  049 |

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](https://github.com/MSIA/kristiyan-dimitrov_msia_text_analytics_2020/tree/project).

Introduction

The website [BoardGameGeek.com](https://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.

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.

Dataset

|  |
| --- |
| 050  051  052  053  054  055  056  057  058  059  060  061  062  063  064  065  066  067  068  069  070  071  072  073  074  075  076  077  078  079  080  081  082  083  084  085  086  087  088  089  090  091  092  093  094  095  096  097  098  099 |

BoardGameGeek.com provides an [XML API](https://boardgamegeek.com/wiki/page/BGG_XML_API2) for retrieving data from their database. Due to difficulties in working with XML, I have instead used a Python package called [boardgamegeek](https://lcosmin.github.io/boardgamegeek/#quick-install), 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](https://github.com/MSIA/Boardgame_Guru_Recommendation_Engine).

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.

Chart, histogram

Description automatically generated

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.

Chart, histogram

Description automatically generated

The descriptions were preprocessed using [gensim.utils.simple\_preprocess](https://radimrehurek.com/gensim/utils.html). 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.

Chart, histogram

Description automatically generated

* 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](https://radimrehurek.com/gensim/models/ldamodel.html) (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.

|  |
| --- |
| 100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149 |

Chart, bar chart, histogram

Description automatically generated

As an interesting aside, I also tried [Latent Semantic Analysis](https://radimrehurek.com/gensim/models/lsimodel.html) (LSA) and [Feature Agglomeration](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.FeatureAgglomeration.html) (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.

* 1. Method – Non-Deep Learning Models Evaluation & Parameter Tuning

|  |
| --- |
| 150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199 |

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](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html#sklearn.model_selection.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](https://en.wikipedia.org/wiki/Jaccard_index#Similarity_of_asymmetric_binary_attributes) 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.

|  |
| --- |
| 200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249 |

|  |
| --- |
| 250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285  286  287  288  289  290  291  292  293  294  295  296  297  298  299 |

It is important to note that the [Random Forests Classifier](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html) inherently supports multi-label tasks. However, the same is *not* true for all other models I tested. For them I used the [OneVsRestClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.multiclass.OneVsRestClassifier.html#sklearn.multiclass.OneVsRestClassifier) from the sklearn library.

Graphical user interface, text, application, chat or text message

Description automatically generated

The entire process can be summarized in the below graph:

Diagram

Description automatically generated

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

* 1. 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.

* 1. 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.

* 1. 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

|  |
| --- |
| 300  301  302  303  304  305  306  307  308  309  310  311  312  313  314  315  316  317  318  319  320  321  322  323  324  325  326  327  328  329  330  331  332  333  334  335  336  337  338  339  340  341  342  343  344  345  346  347  348  349 |

* 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.

* 1. 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:

|  |
| --- |
| 350  351  352  353  354  355  356  357  358  359  360  361  362  363  364  365  366  367  368  369  370  371  372  373  374  375  376  377  378  379  380  381  382  383  384  385  386  387  388  389  390  391  392  393  394  395  396  397  398  399 |

Diagram

Description automatically generated

References

*[https://boardgamegeek.com](https://boardgamegeek.com/)*

[*https://scikit-learn.org/stable/*](https://scikit-learn.org/stable/)

[*https://radimrehurek.com/gensim/*](https://radimrehurek.com/gensim/)