# Finding the Odd One Out

## Measuring Dissimilarity in Written Texts Using Natural Language Processing

Teodor Ganestål (teoga849)
Felix Nodelijk (felno889)
Jakob Norell (jakno732)
Sebastian Ragnarsson (sebra023)
William Stenberg (wilst118)
Gustaf Söderholm (gusso811)

December 18, 2020

---

https://gitlab.liu.se/tdde19-2020-2/tdde19-2020-divergent-text

**Abstract**

This paper evaluates how three models — Doc2Vec, Neural Network (NN), and Latent Dirichlet Allocation (LDA) — can identify dissimilarity between games. The three models were implemented and studied based on a common dataset. The dataset is a large catalogue of games from Steam[1] featuring user voted tags and game developer written descriptions for each game. In order to find the best model parameters, 5-fold Cross Validation was implemented. The results revealed that the Doc2Vec model outperformed both the LDA and the NN model. Furthermore, the LDA model outperformed the NN model. From this a conclusion is made that Doc2Vec is the preferred method when handling this type of data.

[1] https://store.steampowered.com/

# 1 Introduction

In the modern information age there are a lot of on-demand services that supply everything from newspapers to movie recommendations with the click of a button. However with the luxury of so many choices comes also the drawback of getting stuck trying to decide which one to go with. To avoid this problem many suppliers of on-demand services provide recommendations based on what the user has previously chosen in order to minimise the numbers of choices presented. This, however, creates a feedback loop, causing the user to consume the same kind of content indefinitely. This study intends to explore the methods contained within the field of Natural Language Processing (NLP) in order to suggest different content instead of similar content based on previously consumed data. Taking this popular concept and turning it on its head might result in a useful metric for on-demand services but also in other fields.

## 1.1 Natural Language Processing

Natural Language Processing has been around since the 1950s, when Alan Turing developed the Turing test [1]. As a part of passing the test, a machine would have to understand and create speech that could cause it to be mistaken for a human. This sparked ideas eventually resulting in fields analysing, generating and classifying the natural language of humans. This study intends to focus on the latter problem of classification. Classification relies, as the name implies, on the central concept of all texts being subject to categorisation and classification. Being able to derive metadata from a text based on the words it contains, the syntax it uses and the semantics it conveys. This is a process that usually relies on having lots of texts of the same kind that are all individually connected to a class. Then, by studying these texts from different perspectives it is possible to draw conclusions about which parts of the texts are relevant to its class. For example a text that is political in nature might contain the words *higher taxes* which in turn might give away which party the writer belongs to. Finding which class a text belongs to enables a scoring algorithm to figure out if other texts are similar or dissimilar to the original text.

## 1.2 The Dataset

The dataset used in this project is the Steam Store Games dataset [2]. This dataset contains metadata on all the games available on the Steam Content Delivery System, a popular online gaming platform. The features from this set that are useful to this project are the unique identifiers, game descriptions, and public votes on which genre the game belongs to — eighty-four different tags such as *strategy* or *first-person shooter*. The genre tags and the count of their votes can be used to determine if a game is similar to another. The vote distribution over tags will be treated as the gold standard, the true genre nature of the game. Comparing distributions of this sort can answer the question "Is this game similar to some other game?"

The relatively short but information packed textual descriptions on Steam serve as good bases on which to predict the genre distributions, which will be the objective of the machine learning models proposed herein. Using this data might result in biases such as some tags occurring more often than others. These have been addressed in the methods chapter as different approaches use different techniques in order to handle these issues.

## 1.3 Natural Language Processing - Different Approaches

The field of natural language processing (NLP) is broad with a lot of different techniques and

methods to choose from. They all come with their own pros and cons and they do not all produce the same result given a set of data. Using only one of these techniques is not a good approach due to some methods applicability being very narrow or in other ways not being optimal for the task at hand. In order to increase the chances of success, three different solutions will be explored and evaluated alongside each other to later on be compared by a common metric in order to find out which approach is the most effective. These methods will be referred to as Doc2Vec, Neural Network (NN), and Latent Dirichlet Allocation (LDA) and they are covered more in depth in the methods chapter.

# 2 Method

The three different methods chosen from the field of NLP were applied separately to the same set of data. All data was preprocessed from the original data [2] in order to only extract the relevant columns of unique identifiers, game descriptions, and tag (genre) votes. These were then subject to different types of tokenisation, lemmatisation, stop word removal, etc. depending on what the model required.

## 2.1 Doc2Vec

The Paragraph Vector model by Quoc Le et al. [3], was introduced as an extension to the Word Vector model to solve the issue about context. Traditionally, Word Vectors models utilise Bag-of-Words, which has the disadvantage that it does not take context or even the word order into account. This means that words that are semantically similar, may be represented as opposites in a Word Vector model. The Paragraph Vector model is implemented in the Python package *Gensim* [4] as *Doc2Vec*. The goal is to utilise the Doc2Vec model to predict dissimilarity of games from the Steam games dataset.

### 2.1.1 Parameters

The parameters of the Doc2Vec model are the following:

- **Alpha** – The initial learning rate.
- **Epochs** – The number of epochs used when training the model.
- **Vector Size** – The dimensionality of the vectors that each document is transformed into.

The Gensim implementation of Doc2Vec supports more parameters, but these three parameters were the ones relevant to modify in this project.

### 2.1.2 Training the Model

The Doc2Vec model was implemented from the Gensim package. The implementation was used to train the model on game descriptions from the Steam game database as described in Section 1.2. The model was designed with parameters defined in Section 2.1.1 in reference to the Gensim documentation [5]. All descriptions of the dataset are instantiated as *TaggedDocuments*, which is a requirement for the Doc2Vec model in Gensim. Each description is accompanied by a tag that represents the index of a game in the dataset, this is useful as it allows the model to predict the index of games in the dataset depending on the similarity of the description.

The Doc2Vec model is instantiated with all of the *TaggedDocuments* along with the parameters defined in section 2.1.1. The vocabulary, which creates the vector room of the model, is then created. The model is then trained for as many epochs as it was assigned during the initiation stage.

### 2.1.3 Prediction Process

The prediction process is represented in Figure 1, which utilises the trained Doc2Vec model to predict a set of $N$ similar game indexes by inferring a game description. As the indexes are ranked

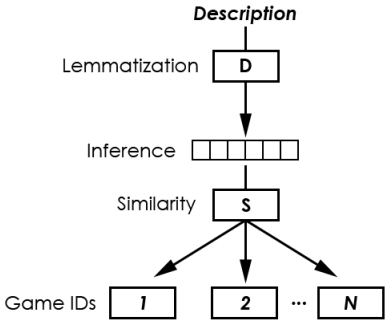


Figure 1: Representation of prediction of games using game descriptions as input

in order of similarity, it is a trivial task to get the dissimilar games, as this can be achieved by reversing the output.

In order to evaluate the model in a way that is comparable with the other methods, the most similar predicted game's true tag distribution was used as the predicted distribution for computing metrics.

## 2.2 Neural Networks

The second model is based on neural networks, and specifically transformer models [6]. Transformers are deep neural networks trained on a large amount of data and comprise the state-of-the-art for many NLP tasks today. A pre-trained instance of BERT, one such transformer, was chosen because it was small enough to run with the resources available in this project.

### 2.2.1 BERT

BERT is a transformer model proposed in 2018 for NLP tasks [7]. A transformer is a new kind of neural network architecture that has multiple levels of encoder-decoders and it uses attention functions to learn dependencies between inputs with different positions (time, placement in a text or similarly). It was shown that BERT was state-of-the-art on many different NLP-tasks [7]. This was done together with a technique called *fine-tuning* as well as a second approach they refer to as *feature-based approach*.

### 2.2.2 Fine-tuning

*Fine-tuning* can be used with a pre-trained BERT model to use BERT's general language representation for more specific tasks. Devlin et al. proposed to fine-tune the entire architecture by training the pre-trained model on the specific task only with a simple classification layer [7]. This approach is much faster than training the whole model from scratch, but it still required much time on the GPU used in this project. Experiments were instead conducted using the pre-trained BERT model without changing the internal parameters and only train a fully connected neural network on the features produced as outputs of BERT. This approach will still be referred to as fine-tuning.

### 2.2.3 Using BERT with Preprocessing

The model and training procedure is implemented in python with PyTorch for all the neural network related parts. First the data is tokenised using the tokeniser `BertTokenizerFast` from Huggingface[2]. An explicit preprocessing process was also implemented where the BERT model is run (perform forward passes) on all the given training data – which takes more than an hour even with a CUDA-enabled GPU. When the preprocessing is done, the fine-tuning network can be used on these BERT outputs in a much less time-consuming training procedure.

### 2.2.4 The Network

The fine-tuning network that is trained consists of one fully connected hidden layer and one output layer. The hidden layer has a variable amount of neurons and a dropout layer with a 50% dropout rate. The dropout layer is applied to the hidden layer during training to try and force the network to learn how to classify less representative data. A leaky ReLU activation function is used before the fully connected output layer with 84 outputs representing tags of games. The outputs of the network consists of raw logits and a softmax function needs to be applied to have the resulting probability distribution. To have raw logits as outputs is dependent on the loss function used in training, which needs modified logits to work.

### 2.2.5 Variable Parameters

The neural network has three parameters that need to be tuned in the training. These are:

- **Number of hidden neurons** – How many neurons to use in the hidden layer of the neural network.
- **Learning rate** – The learning rate for the neural network during training.
- **Number of epochs** – How many times the network should be trained on the whole training data.

## 2.3 Latent Dirichlet Allocation

Latent Dirichlet Allocation is a clustering method used for topic estimation in natural language processing [8]. The model contains a fixed number of initially unknown clusters known as *topics*. Through a generative process using documents as bags of words, each document is assigned a distribution over the topics, and each topic is assigned a distribution over the words. The idea is that there are similarities between documents that are modelled by grouping documents together.

### 2.3.1 Clustering

The generative algorithm is tasked with learning the topic distribution of documents, and the word distributions of topics. Samples from the set of documents are used to infer these distributions. When iterating over the corpus many times, the estimations are updated with respect to each other through new samplings. The hyperparameters $\eta$ and $\alpha$ (*eta* and *alpha*) are used to control the learning rate. A low alpha value means that documents will tend to be represented by few topics with high amplitude, rather than many topics with low amplitude. A high eta value means

[2] `https://huggingface.co/transformers/`

a topic is connected to many words, which introduces overlap when words are associated to many topics. This is not a problem due to the sampling techniques used.

The documents' distributions stabilise during iteration, meaning that the clustering is finished. Owing to the stochastic nature of LDA, it is unknown beforehand what type of similarities will be found and represented by the topics. In this work the topics will be *Steam tags*. A useful property of the LDA architecture is the possibility to assign a prior distribution to topics; specifying a tag's "affinity" for certain words. The process of pairing information-bearing words to key topics is called *seeding* and means the similarities are at least partly known before training.

### 2.3.2 Seeding the Clusters

For this study it is important that each cluster reflects the tags seen in the data. Based on this a seed was created for each desired cluster in order to guide the LDA algorithms clustering. These seeds were selected to be the top 50-100 words that describe a tag based on the data. Seed words were obtained by creating a corpus of all the words contained in the dataset. Using this corpus a table was created as seen in Figure 2.

Here each word in the corpus is assigned a term frequency–inverse document frequency score (TF-IDF score) based on how often it occurs in its own description compared to all the other ones. Since the word *wasteland* occurs a disproportional amount in a certain description it is given a high value.

| Word / Game | Gun | Wasteland | Sunny | Rainbow |
|---|---|---|---|---|
| Dota 2 | ... | ... | ... | ... |
| Tetris | ... | ... | ... | ... |
| Fallout 3 | ... | **0.94** | ... | ... |
| FIFA 2020 | ... | ... | ... | ... |
| Pacman | ... | ... | ... | ... |
| Civ 5 | ... | ... | ... | ... |

Figure 2: Each game description has a TF-IDF score for every word in the corpus.

Then a second table was created which links each word of the corpus to the tags in order to find the most relevant words for each tag. Using the tag distribution of each game a new table could be populated as seen in Figure 3. In this table the TF-IDF score is multiplied by 0.8, because Fallout 3 is considered to be 80% post-apocalyptic according to its votes, and then it is added to it's current seeding score of 0 to produce 0.75. After doing this for each word the new seeding scores are used to determine if a word is relevant to a tag. The word wasteland for example occurs often under the post-apocalyptic tag and not as often under the sports tag, this gives it a high seeding score for post-apocalyptic. Using this matrix, the model has a notion of similarity between games (in the form of bags of words) and uses it to perform recommendations.

During training, an additional parameter *confidence* determines for each training sample if it should

make use of the training matrix. It might make intuitive sense to always use the seed matrix, but the practise of sometimes distributing words randomly make for a smoother topic distribution over words, so that no topics can totally dominate certain words.

| Tag \ Word | Gun | Wasteland | Sunny | Rainbow |
|---|---|---|---|---|
| Action | ... | ... | ... | ... |
| Puzzle | ... | ... | ... | ... |
| Anime | ... | ... | ... | ... |
| Sports | ... | ... | ... | ... |
| Post-apocalytic | ... | **0.75** | ... | ... |
| Dark | ... | ... | ... | ... |

Figure 3: Each tag has a seed score for every word in the corpus.

## 2.4 Choosing Parameters

The three models use different parameters, and these parameters affect the performance of the models. To find the most suitable parameters, grid search was used. For each combination of parameters, 5-fold cross-validation was performed to determine the model's performance with those parameters.

| **Model** | **Parameter** | **Values** |
|---|---|---|
| Doc2Vec | Alpha | 0.002, 0.025, 0.05, 0.1, 0.4 |
| | Epochs | 5, 10, 20 |
| | Vector Size | 2, 5, 10, 69, 80, 100, 120, 160, 200, 300, 2000 |
| NN | Number of hidden neurons | 10, 100, 1000 |
| | Epochs | 10, 100, 1000 |
| | Learning rate | $10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$ |
| LDA | Alpha | 0.001, 0.01, 0.1 |
| | Eta | 0.01, 0.1, 0.4, 0.8 |
| | Confidence | 0.1, 0.4, 0.7, 0.95 |

Table 1: Parameters tested for each model

In Table 1 the parameters for the different models can be seen and the values that were tested for each parameter.

## 2.5 Comparing the Models

Each model was evaluated using the two following metrics for comparing probability distributions:

- **Jensen-Shannon** – An inversion of the Jensen-Shannon distance [9], $1 - JSD(P \mid Q)$ where $JSD(P \mid Q)$ is the traditional Jensen-Shannon distance between probability distributions $P$ and $Q$ over probability space $\chi$, defined as

$$\begin{aligned} &JSD(P \mid Q) = \sqrt{\frac{1}{2}D(P \mid M) + \frac{1}{2}D(Q \mid M)}, \text{ where} \\ &M = \frac{1}{2}(P + Q) \text{ and} \\ &D(P \mid Q) = \sum_{x \in \chi} P(x) \log\left(\frac{P(x)}{Q(x)}\right). \end{aligned} \tag{1}$$

- **Match Top 5** – The overlap in the five most probable outcomes of the two distributions. That is, how many outcomes do the distributions have in common, looking only at the five with highest probability?

The scores used for comparing models were the mean scores achieved for each metric in the 5-fold cross-validation.

The models were all evaluated by predicting a distribution of game tags and comparing those to the true distribution. Therefore these metrics represent similarity between games. These metrics also translate, inversely, to how well the models perform when predicting dissimilarity of games. This is because the closer the predicted distribution is to the true distribution, the further away it is from the distribution furthest away from the true distribution.

## 2.6 Demo Tool

To show the purpose of this project, suggesting games that seem new to the user, based on descriptions of the games in the database. The demo tool is an attempt to visualise the three different models: LDA, Doc2Vec, and NN. The demo gives the user the possibility to choose a subset of provided games that the user claims to have played before. A suggestion of games that are the least similar (in given the database) to the user's current preference is then provided by each model.

# 3 Results

The results of the three models will be presented separately, followed by a comparison. The Jensen-Shannon metric and Match Top 5-scores will be presented: For Jensen-Shannon, this means computing the average score per game between the ground-truth vote distribution and the predicted topic distributed by a model. For the Match Top 5 score, the average cardinality of the intersection of the top five tags for both distributions is computed.

## 3.1 Doc2Vec

The chosen Doc2Vec model used the parameters Alpha = 0.025, Epochs = 20, and Vector Size = 160. The parameters chosen for this model were not the ones that performed best. In particular, the best scoring Vector Size = 2000 was not chosen due to making the model very computationally heavy.

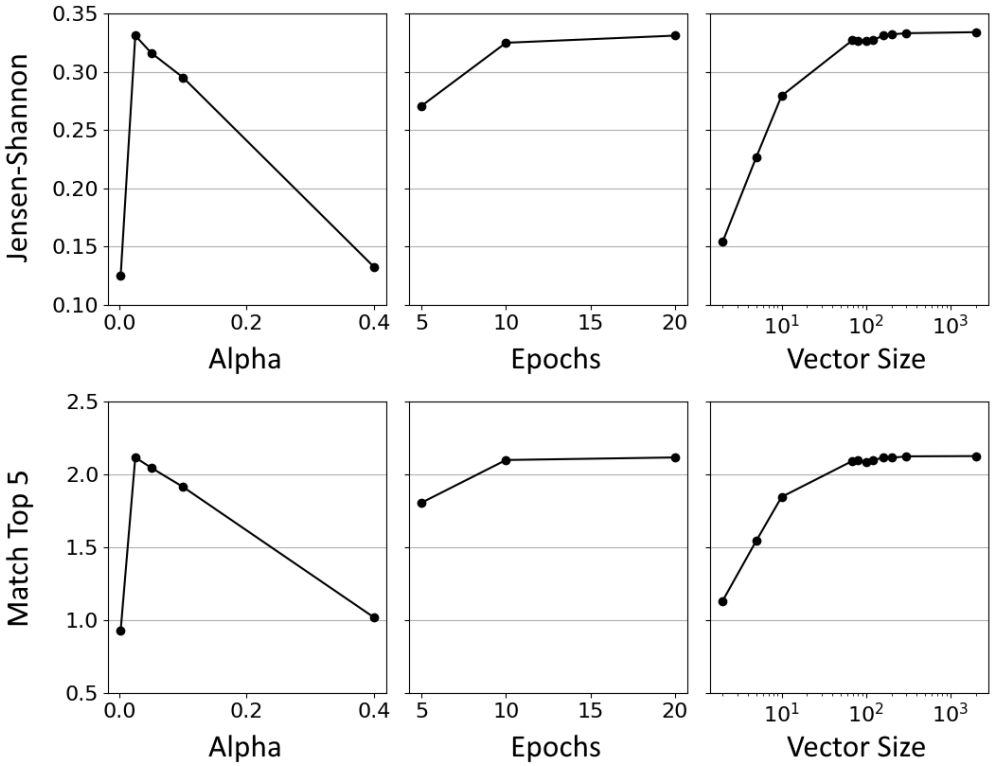


Figure 4: The validation score from varying each parameter Alpha, Epochs, and Vector Size respectively while keeping the other two parameters constant.

Using 5-fold Cross Validation, the model reached an average Jensen-Shannon score of 0.333 with Vector Size = 2000. The same score when using Vector Size = 160 was 0.331, which is more than 99% of the best score. The performance seems to converge as Vector Size increases, so Vector Size = 160 was chosen since it gave very similar performance to Vector Size = 2000, in only 27.7% of the computation time. In addition to high computation time, a model with large

Vector Size also uses more computational resources. Figure 4 shows the effect on performance from changing one parameter at a time on the chosen model.

## 3.2 Neural Networks

The parameters for the neural network with the best performance were as follows.

- epochs: 100
- learning rate: $10^{-3}$
- number of dense nodes: 1000

The best validation score for the epochs parameter was neither the highest nor the lowest value. Learning rate resulted in the second lowest value tested. The greatest number of nodes was the configuration with the highest validation score. To illustrate how the individual parameters affect the score, the validation score is plotted for each variable while keeping the other two parameters constant with their best value, as seen in Figure 5. What must be noted is that the best model was chosen based on the top-5 score and not on the Jensen-Shannon metric. There was another model with higher Jensen-Shannon score but lower top-5 score.

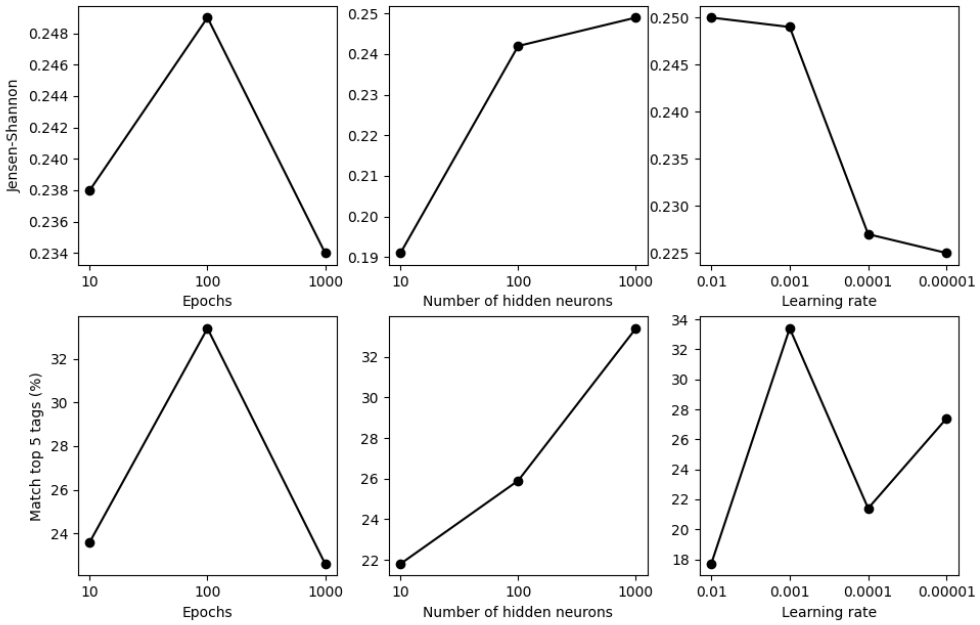


Figure 5: The Jensen Shannon score and procent of the 5 top matching tags from varying each parameter while keeping the other two constant.

## 3.3 Latent Dirichlet Allocation

The $\alpha$, $\eta$, and confidence parameters were determined to optimise performance for the values 0.001, 0.8, and 0.4 respectively. The numbers are interpreted as follows: The best $\alpha$-value was 0.001, the lowest tried. This means the model prefers for games to be represented by few topics with high amplitude, rather than having a little amount in every topic. This is congruent with actual vote distributions for many games, where some topics are absolutely dominant.

A confidence of 0.4 means that during training, two out of five words are directly allocated using the seeding matrix, and the rest distributed randomly according to the current topic-word distribution, this means that there is a balance between exploiting the known relation between words and topics, and exploring the connections between a word and other topics.

Lastly, a high (0.8) value of $\eta$ means each topic is associated with many words when iterating, in addition to the initial seeding words. This is congruent with the balanced seeding confidence, the model prefers to let a single word explore many topics instead of being statically locked to a single or a few topics.

Running these best parameters, a Jensen-Shannon metric of 0.325 is achieved, and a Match Top 5 score 2.008. On average, if presented with the description text of a Steam game, the LDA model's prediction will have within its top five tags two out of top five tags that the community voted.

## 3.4 Comparing the Models

| **Model** | **Jensen-Shannon** | **Match Top 5** |
|---|---|---|
| Doc2Vec | 0.331 | 2.111 |
| NN | 0.246 | 1.765 |
| LDA | 0.325 | 2.008 |

Table 2: Metrics for the best version of each model

The results in Table 2 show that Doc2Vec outperforms both the NN model and, albeit marginally, the LDA model. This is further reflected in the Match Top 5 score.

# 4 Discussion

Tags and descriptions are two distinctively different ways to predict a dissimilar game. It can be argued that descriptions may provide more information, however, it is not obvious that this information is relevant to the game, or if it is relevant, that it provides enough text. Conversely, tags may provide condensed, relevant, information. However, these tags are selected by the users and may not accurately represent the game to the fullest extent. It may be that too few relevant tags are used to describe the game, in which case it might be a less useful predictor. It might be that these two methods can complement each other to provide more information, and consequently, result in a better performing model.

An interesting example is *KovaaK's FPS Aim Trainer*[3]. Using the game descriptions in the Doc2Vec model, the most similar predicted games were other aim training games. On the other hand, simply selecting the games with the shortest Jensen-Shannon distance between their true tag distributions, other action and fps games were deemed most similar. In this example it can be seen that the tags present in the data do not capture every possible aspect of games, such as these "training"-type games.

As game description are written by the owners of the product, the descriptions may be prone to various grammatical mistakes as well as spelling mistakes and contextual mistakes. These errors may, undoubtedly, disturb the training or prediction process. However, it is difficult to assess the extent to which these affect the results.

The problem of finding dissimilar games is a trivial problem as a user is likely to select a dissimilar game at random, provided that the user has the entire catalogue of steam games in possession. Therefore, it is perhaps considered more interesting to optimise for relevant recommendations. However, it could be argued that there is a chance that a user may, at random, select a similar game. In this case, the model would perform better as it would guarantee that the most dissimilar games – which the model has deemed dissimilar – would be provided to the user. Furthermore, while the probability is high that selecting a game at random will result in a dissimilar game, it would not be evident that this game is, in-fact, the most dissimilar game. Moreover, games can be dissimilar in more than one way. The models base their assumption on the text description and/or the tags. It is important to note, however, that there may exist other factors which determine the dissimilarity.

A thing to consider is how the metrics chosen represent the actual similarity of games. The Jensen-Shannon metric, based on Jensen-Shannon distance, measures the difference between probability distributions. However, when seeing these distributions as vectors, each dimension represents a tag, and they are all perpendicular to each other. What this means is that a game with only *action* tags will be seen as similar to a game with only *fps* tags as to a game with only *puzzle* tags. Intuitively *action* and *fps* would be more similar than *action* and *puzzle*, but the metric in itself does not reflect this.

## 4.1 Results

The results showed that the Doc2Vec and LDA methods both had similar performances with BERT reaching a lower score, this is in line with results reached by Krithika Iyer [10] and Kovalev et al. [11]. While the Jenson-Shannon metric is not used in the aforementioned articles, correlations between the performance of the different models can still be inferred, which allows for a comparison with

[3]The game has since collecting the data been updated to *KovaaK 2.0*: `https://store.steampowered.com/app/824270/KovaaK_20/`

these findings. These papers report on having slightly higher accuracy for Doc2Vec, suggesting that it is more adapt at handling this task than LDA or BERT.

The results for the NN model were worse than expected and one could assume that there are better parameter values or designs for the NN model that would perform better. Based on the produced predictions it seemed that the NN model learned that the action tag is over-represented and thus guessed that most games were action games. The methods used in this project did not produce satisfactory results in that aspect, but there should be some way to solve this without changing too much in this design. It would for example be interesting to try with more hidden neurons, more hidden layers or with much more training. Using Jensen-Shannon divergence as loss function instead of Kullback-Leibler is another change that might yield some improvement.

The metrics shown as results for each model are, however, the average scores achieved during cross-validation. The models were not tested on a separate set of test data, but only on validation data. This might lead to misleading results as the models chosen are optimised for the validation data. Testing with independent data is needed in order to say whether the models generalise well.

## 4.2 Conclusion

Looking forward from this project, it is evident that the problem of finding online content from description texts only — similar or dissimilar — has many possible solution architectures. This work has presented a method of straight-forward vector representation, a matrix factorisation and clustering method, and a neural network. All three models produced satisfactory results (for the scope of this project) and it is clear that the problem is solvable with the proposed tools.

The considered dataset was preprocessed in a domain-specific way, but any body of text could be used. However, the Steam game dataset is rather small (having only just above twenty thousand games) with a comparatively large tag set of eighty-four. It would be interesting to see how these methods perform in other domains where the texts have other connections to the content. Another possible change is to let a Word2Vec/Doc2Vec model relate its dimensions to the tags, similar to how the LDA model proposed here uses seeding to relate its clusters to tags.

Other domains might necessitate different forms of preprocessing, and might not compare distributions and hence not use Jensen-Shannon as a metric, but there are still valid comparisons to be made.

# References


[1] A.M. Turing. Computing machinery and intelligence. *Epstein R., Roberts G., Beber G. (eds) Parsing the Turing Test*, 2009.

[2] Nik Davis. Steam store games (clean dataset). `https://www.kaggle.com/nikdavis/steam-store-games`, 2019.

[3] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, page II–1188–II–1196. JMLR.org, 2014.

[4] Radim Rehurek. Doc2vec paragraph embeddings. `https://radimrehurek.com/gensim/models/doc2vec.html`, 2020.

[5] Radim Rehurek. Doc2vec model. `https://radimrehurek.com/gensim/auto_examples/tutorials/run_doc2vec_lee.html`, 2020.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[8] Killian Tattank. Lda and document similarity. `https://www.kaggle.com/ktattan/lda-and-document-similarity/comments`, 2017.

[9] Dominik Maria Endres and Johannes E Schindelin. A new metric for probability distributions. *IEEE Transactions on Information theory*, 49(7):1858–1860, 2003.

[10] Krithika Iyer. Classification of legal text. 2020.

[11] A Kovalev, I Nikiforov, and P Drobintsev. An approach to semantic search on technical documentation based on machine learning algorithm for customer request resolution automation. 2020.