Understanding the Influence of Hyperparameters on Text Embeddings for Text Classification Tasks

Nils Witt[1(✉)] and Christin Seifert[2(✉)]

[1] ZBW-Leibniz Information Centre for Economics, Kiel, Germany
n.witt@zbw.eu
[2] University of Passau, Passau, Germany
christin.seifert@uni-passau.de

Abstract. Many applications in the natural language processing domain require the tuning of machine learning algorithms, which involves adaptation of hyperparameters. We perform experiments by systematically varying hyperparameter settings of text embedding algorithms to obtain insights about the influence and interrelation of hyperparameters on the model performance on a text classification task using text embedding features. For some parameters (e.g., size of the context window) we could not find an influence on the accuracy while others (e.g., dimensionality of the embeddings) strongly influence the results, but have a range where the results are nearly optimal. These insights are beneficial to researchers and practitioners in order to find sensible hyperparameter configurations for research projects based on text embeddings. This reduces the parameter search space and the amount of (manual and automatic) optimization time.

Keywords: Document embeddings · Hyperparameter optimization · Natural language processing

1 Introduction

Many applications in the natural language processing domain require the tuning of machine learning algorithms. Since there is no superior algorithm or model per se [19], machine learning models have to be carefully chosen and tuned. This tuning involves adaptation of learning parameters and model hyperparameters. A common approach to hyperparameter tuning is manual search: algorithm developers select sensible parameter choices from their experience and then repeatedly evaluate the models and adapt parameters. While manual search is the most time-consuming approach, it has the advantage that algorithm designers get some insights into parameter influences. But since humans are unable to reason in multi-dimensional spaces, they hardly achieve globally optimal results. A brute-force approach is grid search, where all (exponential many) parameter configurations from a pre-defined set are tested on a high-dimensional grid. For

J. Kamps et al. (Eds.): TPDL 2017, LNCS 10450, pp. 193–204, 2017.
DOI: 10.1007/978-3-319-67008-9_16

cases where some parameters are less important than others random search [1] has been shown to outperform grid search. Both, random search and grid search are easy to implement, easily parallelizable and generally provide better results than manual search [1]. The Gaussian Process Optimizaton (GPO) approach aims to predict the next promising parameter configuration based on the previously observed performance w.r.t. a loss function and therefore searches the parameter space efficiently [18]. Despite its superior performance, the disadvantages of GPO is that another set of hyperparameters must be tuned (e.g., the choice of the acquisition function and covariance functions) and less insight about the parameter influence to a machine learning model is generated.

As in every machine learning problem, the performance of text classification applications strongly depends on the representation of the features. The most common representation is the bag of words representation in the vector-space model for texts [11]. More recently, distributed representations have been proposed, most prominently, word2vec [12] and GloVe [14] for representing terms and doc2vec for representing documents [9]. Text embedding-based techniques have outperformed the state-of-the-art in several tasks. Among these are document classification [5] and word sense disambiguation [4]. But again, the training of such models requires hyperparameter tuning, for instance, the size of the embedding vector, the learning rate and the number of negative samples.

In this paper, we show that the doc2vec model is very sensitive to settings of hyperparameters in classification tasks and analyze their influence. The goal of this paper is to provide insights into sensible parameter configurations for researchers and practitioners. More specifically, our research questions are the following:

RQ1-: What is the overall variance in accuracy for different hyperparameter settings? (Sect. 5.1)
RQ-2: Are optimal hyperparameters found on small datasets predictive for optimal hyperparameter settings on larger training sets? (Sect. 5.2)
RQ-3: How significant is the influence of the hyperparameters on the accuracy? (Sect. 5.3)
RQ-4: Are there interrelation between hyperparameters? That is, are there parameters that depend on each other and thus need to be considered jointly? (Sect. 5.4)

2 Related Work

As our work focuses on hyperparameter optimization of document embeddings we review work from these research areas.

Text Embeddings. The representation of words as dense vectors (word embeddings) has become popular for many natural language processing tasks. The popularity roots in the ability of these embeddings to encode semantic linearities such as `king - man + woman = queen` and the resulting state-of-the-art performance in several natural language processing tasks. The first approach to

word embeddings is word2vec [12], which has been successfully used in applications like document classification [5] and zero-shot learning [16]. Word2vec can be trained using two different architectures, `cbow` and `skip-gram`. In `cbow` the vector of the input word is predicted by the vectors of the surrounding context words. The `skip-gram` architecture predicts the context word vectors by the input word vector. In both cases the number of context words is a hyperparameter of this model, such that vectors of words which appear in the same context will become similar whereas vectors from randomly sampled words (i.e., negative sampling) will become dissimilar [13]. Doc2vec is an extension of word2vec that learns embeddings for word sequences like paragraphs or entire documents [9]. In doc2vec, each document is treated as a word. Similar to word2vec, doc2vec can be trained using two different architectures `dbow` (distributed bag of words, similar to `skip-gram`) and `dmpv` (distributed memory model of paragraph vectors, similar to `cbow`). `Dbow` predicts the document words based on the document vector. `Dmpv` concatenates the document vector and the vectors of words in a window to predict a document word. Model hyperparameters of doc2vec are the choice of the architecture, the size of the context windows, the size of the embeddings, the number of negative samples and whether hierarchical sampling of words is used. We base our evaluation of hyperparamaters on document representations of doc2vec and embed the evaluation in a text classification task.

Hyperparameter Optimization. The majority of machine learning algorithms exposes hyperparameters which must be tuned carefully. Traditionally, the optimization is carried out by humans, which likely leads to suboptimal results because of inferior human intuition about multi-dimensional functions. In settings with sufficient computing resources, grid search approaches evaluate all combinations of hyperparameter values and are easily parallelizable. But, with an increasing number of parameters and values the computation becomes intractable. Bergstra and Bengio have found that randomly choosing values finds better models and requires less time than exhaustive grid-search [1] because less time is spent exploring parameters that have little influence. Bergstra et al. presented two hyperparameter optimzation algorithms and compared them to human experts and random search [2]. Their tree-structured Parzen approach showed superior performance over the Gaussian process (GP) [15] approach, while both outperformed manual and random search (in some cases with notable margins). Further advances were made by Snoek et al., who used Bayesian Optimization (BO) with GP priors to enhance the state of the art on the CIFAR-10 dataset by over 3% [18]. Despite its superior performance, the disadvantages of BO is that another set of hyperparameters have to be tuned (e.g., the choice of the kernel and the scopes for the hyperparameters) and – because of the missing manual parameter setting, test, evaluation cycle – less insight about the parameter influence to the machine learning model is generated. Thus, in order to understand hyperparameter influence, we apply grid search in our evaluation.

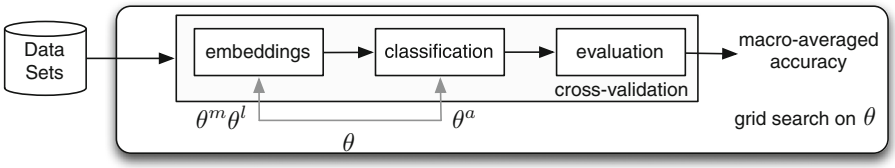


Fig. 1. Overview of the approach. Grid search is executed on the hyperparameter vector θ, the model is evaluated using cross-validation.

3 Approach

Our goal to investigate the influence of hyperparameters on the model performance, lead to questions about (i) overall performance variation, (ii) influence of single variables, the (iii) interrelation between variables and (iv) how the size of the training set influences optimal hyperparameters. To assess model performance, we choose the task of text classification, as text classification has been extensively studied [17] and comparative baselines are available. Also, this is a task for which doc2vec embeddings are especially suited, as doc2vec tends to build clusters of similar documents. The overall approach is depicted in Fig. 1. First, for each dataset feature representations are calculated using doc2vec. This step is governed by model hyperparameters of doc2vec θ^m (e.g., the size of the embeddings) and learning parameters θ^l (e.g., the learning rate). Then, a classification model is trained, whereas the classifier is governed by the hyperparameters of θ^a (such as k when using the k-Nearest neighbor classifier), composing the complete vector of hyperparameters θ. The grid search is employed on the hyperparameter vector θ. We use 5-fold cross-validation and report the results in terms of (macro-averaged) accuracy – the most common evaluation measure for text classification.

Because exhaustive search, even on a moderate number of parameters[1] requires considerable computational and memory resources, we use a two-stage strategy. In the first stage, we exhaustively search the hyperparameter space on a limited number of training samples. The initial parameters are derived from the literature [8,12]. In the second stage we train models on a larger training set but on a restricted grid using the best performing hyperparameter combinations. This allows us to compare the performance against state-of-the-art approaches using a similar amount of training data. Subsequently we also carry out a Bayesian optimization to find an optimal hyperparameter configuration and contrast the corresponding results to the results obtained using grid search.

4 Experimental Settings

We perform the experiments on well-known datasets for text classification and evaluate the overall accuracy, the influence of single parameters, their interrelation

[1] For example, 6 parameters with 3 values each results in $3^6 = 729$ combinations. Even worse, with 5-fold cross validation this results in $5 \cdot 3^6 = 3645$ models

Table 1. Overview of the hyperparameter space $\boldsymbol{\theta}$ in the stages S1, S2.

$\boldsymbol{\theta}$	**Description**	**Type**	**Values (S1)**	**Values (S2)**
θ_{arch}	doc2vec architecture	nominal	dbow, dm	dbow
θ_{hs}	hierarchical sampling (off/on)	boolean	0, 1	1
θ_{ns}	number of negative samples	integer	1, 5, 20	5, 20
θ_{d}	embedding size	integer	2, 3, 8, 24, 64	24, 64, 256
θ_{win}	context window size	integer	5, 20, 50	5, 30, 100
θ_{ts}	number of training documents	integer	10^3, 10^4	10^4, 10^5
θ_{α}	learning rate (log scale)	real	0.001, 0.01, 0.1	0.01, 0.1, 1
θ_{epoch}	number of training iterations	integer	5, 30, 50	30, 100, 250
θ_{k}	number of nearest neighbors	integer	1, 5, 25	10, 50
Number of models			**4860**	**324**

and the stability of optimal parameters with varying sizes of the training set. The source code of the experiments as well as additional material is available online[2].

Parameter Configuration. Table 1 provides an overview of the parameter settings for the two stages of the experiments. Stage 1 trains more models than Stage 2 but with fewer training examples, thus satisfying memory and time constraints. The values for the first stage were chosen to cover a wide range from extremely low to extremely high values also including values used in related work. The values for the second stage are refinements over the first stage, while the most promising configurations were chosen and their range was narrowed. Some values were omitted (e.g. $\theta_{arch} = 1$) whereas others were added ($\theta_d = 256$ because larger embeddings sizes seem to be promising). Generally, the accuracy of this model increases with the size of the training data [3]. Hence, θ_{ts} is considered an external constraint rather than a value to be optimized. For the classifier we used k-nearest neighbor approach, thus the hyperparameters set for the classification model θ_a only consists of the number of neighbors.

Datasets. For the experiments, we chose two datasets from the domain of text classification, that exhibit different characteristics, i.e., the order of magnitude of contained documents and number of classes ,and are well-studied (e.g. [5–7]), The amazon[3] dataset, which was also used in [5] contains 12.8M user reviews for products assigned to four different categories (*Home and Kitchen*, *Electronics*, *Books* and *Movies and TV*). The dataset is strongly imbalanced with 8.9M *Books*, 1.69M *Electronics*, 1.7M *Movies and TV* and 0.55M *Home and Kitchen* reviews. This implies that a trivial categorizer can achieve 69% accuracy. The average length of the reviews is 796 tokens (i.e. sequences of characters

[2] http://doi.org/10.5281/zenodo.495086
[3] http://jmcauley.ucsd.edu/data/amazon/

surrounded by whitespaces). The 20newsgroups[4] dataset consists of 18.846 newsgroup articles categorized into 20 groups with an average length of 1902 tokens per article. The same preprocessing was carried out for both datasets. In order to reduce the noise in the text, short tokens (with less than 3 characters), quotation marks, punctuation marks, whitespaces (except for space characters) etc. were omitted[5]. Also, tokens that appeared less than three times where ignored. We did not use the available train/test splits but randomly generated the splits during the 5-fold cross validation runs. Further, we generated subsets with 1k, 10k (20newsgroups, amazon) and 100k documents (amazon) by random sampling.

5 Results

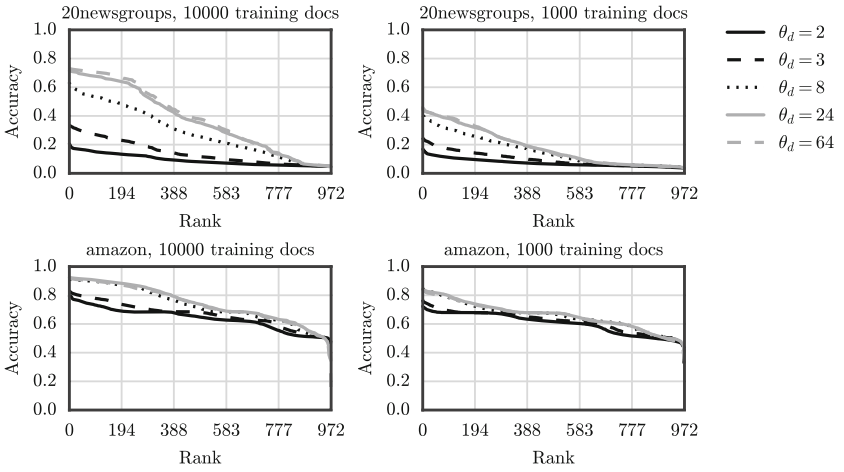
In this section we first describe the results of the experiments w.r.t. the overall performance, the influence of single parameters and the interrelation of parameters. We report these results on the training dataset with 10k documents. Further, we assess the stability of optimal parameters across varying sizes of the training dataset (1k, 10k and 100k documents).

5.1 Overall Performance

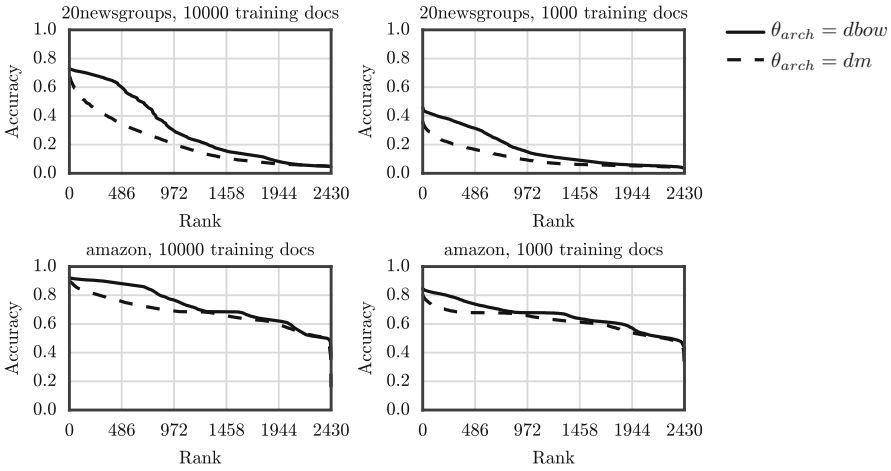
In terms of overall accuracy, results vary greatly across different parameter settings. Table 2 provides an overview of the results obtained on the 20newsgroup dataset with 10k training examples sorted by the rank of the model, where rank 1 is assigned to the best performing model. Table 2 also shows that the influence of single parameters on the model accuracy is not obvious, e.g., while worst performing models tend to have larger values for the number of negative samples θ_{ns}, some of the best performing models also have a value of $\theta_{ns} = 20$. The variation of accuracy across models is equally prominent on the amazon dataset (best performing model accuracy 0.9244, worst 0.1540) when trained on 10k documents, confirming that model hyperparameter settings are crucial for successful application of the learning algorithm. While the accuracy varies greatly across hyperparameter settings, the behavior depends on the dataset. As shown in Fig. 2 for some settings (e.g., 20newgroups, 10k training documents $\theta_d = 64$) accuracy decreases slowly from 73% and then drops rapidly (at approximately rank 230). This means that most of the models are quite similar in accuracy, i.e., most parameter configurations are "good", but some yield very low accuracy. In comparison, other settings (e.g., 20newsgroups, 1k training documents $\theta_d = 64$) show a steady decrease of accuracy, meaning that there are only some very good models, but many average- and bad-performing models.

We also observed that a well tuned doc2vec model achieves performance comparable to approaches using more complex features and the same classifier (k-NN). On the amazon dataset doc2vec achieves an accuracy of 0.924

[4] http://qwone.com/~jason/20Newsgroups/20news-bydate.tar.gz
[5] Details available in ipython notebooks https://doi.org/10.5281/zenodo.809860



(a) Varying embedding size θ_d



(b) Varying model architecture θ_{arch}

Fig. 2. Classifier rankings for (a) different embedding sizes (θ_d) and (b) model architectures θ_{arch}. Accuracy obtained with 5-fold cross validation. Similar plots for other variable combinations are available via http://doi.org/10.5281/zenodo.495086

(with 10k training examples) comparable to 0.926 in [5]. On the 20newsgroups doc2vec achieves an accuracy of 0.73 with grid-search and 0.74 with Bayesian optimization, which is comparable to 0.73 in [5].

Table 2. Excerpt of the classifier results for the 20newsgroups dataset with 10k training examples ordered by model performance, showing accuracy a and standard deviation (averaged over cross-validation folds) of best and worst performing models for grid search (GS) and results from the Bayesian optimization (BO).

Meth.	Rank	θ_k	θ_α	θ_{arch}	θ_{hs}	θ_{epoch}	θ_{ns}	θ_d	θ_{win}	a (stdev)
GS	1	5	.10	dbow	1	30	20	64	5	**.7335** (.006)
GS	2	5	.01	dbow	1	50	20	64	5	**.7296** (.007)
GS	3	5	.01	dbow	1	50	5	64	20	**.7294** (.005)
GS	4	5	.01	dbow	1	50	5	64	50	**.7292** (.006)
GS	5	5	.10	dbow	1	30	20	64	20	**.7280** (.006)
GS	4856	25	.001	dbow	0	5	20	8	5	**.0449** (.004)
GS	4857	5	.001	dbow	0	5	5	24	20	**.0446** (.002)
GS	4858	25	.001	dbow	0	5	20	24	5	**.0445** (.006)
GS	4859	5	.001	dbow	0	5	5	8	20	**.0440** (.000)
GS	4860	25	.001	dm	0	5	5	8	20	**.0424** (.001)
BO	1	10	.0396	dbow	1	14	11	245	40	**.7415** (.003)
BO	2	8	.0893	dbow	1	15	15	58	48	**.7389** (.002)
BO	3	15	.0223	dbow	1	28	17	96	13	**.7341** (.004)

Table 3. Best classifications accuracy w.r.t the training set size using grid search. 18.8k is the size of the 20newsgroups dataset. Order of elements of optimal θ as in Table 2.

	10k	**100k/18,8k**
amazon	.9244	.9516
	$\theta^* = (25, 0.1, cbow, 1, 50, 5, 24, 20)$	$\theta^* = (10, 0.1, cbow, 1, 30, 5, 256, 5)$
20newsgroups	.7335	.8034
	$\theta^* = (5, 0.1, cbow, 1, 30, 20, 64, 5)$	$\theta^* = (10, 0.1, cbow, 1, 30, 5, 256, 5)$

5.2 Varying Training Data Set Size

The experiments confirm, that machine learning models benefit from more training data [3] (see Table 3). The accuracy gain of 7% on the 20newsgroups dataset when the training size is raised from 10k to 18.8k indicates that the accuracy could be enhanced even further, if more training data were available. The scaling on the amazon dataset is different: 10k training examples are sufficient to obtain good results. A tenfold increase in training samples only leads to a 2.7% accuracy gain. As the amount of training data increases, the ideal value for the learning rate also changes. Using 100k training documents, configurations using $\theta_\alpha = 0.01$ outplay those with $\theta_\alpha = 0.1$. But correspondingly, more iterations are necessary (100 or more compared to 30). Altogether, the learning rate is crucial; Fig. 3b (right) depicts that models with $\theta_\alpha = 1$ fall back to the performance of trivial classifiers. θ_d changes only slightly with respect to the training size, as shown by Fig. 3b (left) and Fig. 3a (left). With 100k training models using 64

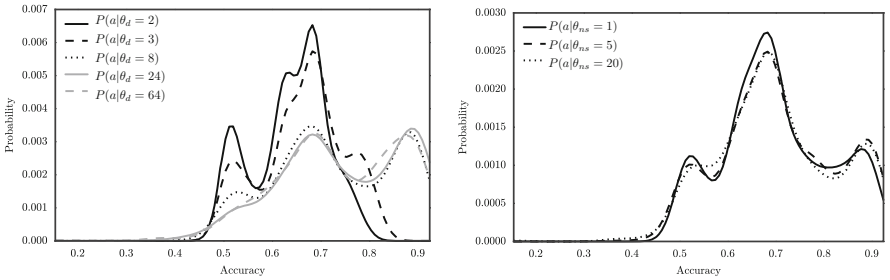
dimensions perform marginally better than 24 dimensions, whereas the situation with 10k training examples is vice versa. A similar relation can be observed on the 20newsgroups dataset but with $\theta_d = 64$ and $\theta_d = 256$, respectively. In Fig. 3b (left) we also see that models using $\theta_d = 256$ overfit the data on the amazon dataset, which leads to a declining performance.

5.3 Parameter Influence

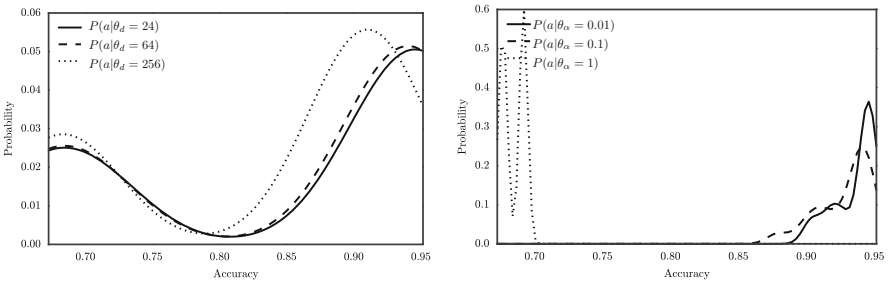
We plotted the model accuracy for different hyperparameter values as exemplified in Fig. 2 for the model architecture θ_{arch} and the embedding size θ_d. The plots were created by collecting the results of all models where one parameter was set to a specific value (e.g. the solid black line in Fig. 2a depicts the accuracy of all models that used two dimensional embeddings). These models were then ordered by accuracy. In terms of the model architecture, the distributed bag of words ($\theta_{arch} = dbow$) models outperformed the *distributed memory* ($\theta_{arch} = dm$) models in every scenario as depicted in Fig. 2b. Similar behaviour was observed for parameter θ_{hs}, models using hierarchical sampling which generally outperformed models not using hierarchical sampling. Very small embedding sizes ($\theta_d \in \{2, 3, 8\}$) have a strong negative impact on the accuracy. Larger embeddings sizes ($\theta_d \in \{24, 64\}$) yield more accurate classifiers as depicted in Figs. 2a and 3. Interestingly, the best results with comparatively small embedding sizes (e.g. 24 dimensions) are similar to those achieved with higher embedding sizes. Further, we found little to no effect of window size θ_{win}, the number of negative samples θ_{ns} and the number of nearest neighbors θ_k on the accuracy. Finally, we found a strong influence and interrelation on the accuracy when varying the learning rate θ_α and the number of epochs θ_{epoch}. Thus, when choosing these parameters they must be considered jointly as discussed subsequently.

5.4 Interrelation of Parameters

The experiments show a interrelation of the learning rate (θ_α) and the number of epochs (θ_{epoch}) (see Fig. 4 (left)). Good accuracy is achieved when a high learning rate is combined with few epochs. Likewise, a small learning rate in combination with many epochs gains similar results (see Fig. 4 at 50 epochs). It must be pointed out though that the training time mainly depends on the epochs, which makes a setting with a high learning rate and few epochs favorable when training time is crucial, since models with high learning rates are prone to overfitting. But in scenarios where accuracy is the top priority and the training time is negligible, a smaller learning rate with more epochs is favorable. Apart from the interrelation between the learning rate and the epochs our experiments found no additional interrelations (as exemplified by Fig. 4 (right)).



(a) 10k training samples, left: varying θ_d, right: varying θ_{ns}



(b) 100k training samples, left: varying θ_d, right: varying θ_α

Fig. 3. Approximation of classifier accuracy a on amazon dataset as probability density estimated using Gaussian Kernel Density Estimation. Similar plots for other parameters are available via http://doi.org/10.5281/zenodo.495086

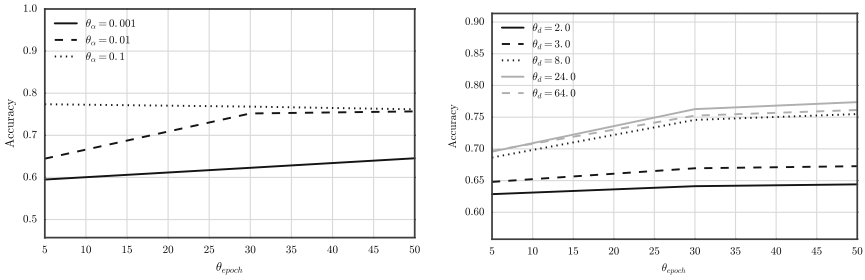


Fig. 4. Parameter interrelations on amazon dataset with 10k training samples. Left: To achieve optimal results θ_α and θ_{epoch} must be considered jointly, the parameters depend on each other. Right: To achieve optimal results θ_d is tuned without considering θ_{epoch}, the parameters are independent of each other.

6 Discussion

The experiments showed that, in general, hyperparameter settings have a huge impact on the accuracy (Sect. 5.1). Further, we found four categories

of hyperparameters: those with no influence on the accuracy, those with a clear optimal value, those with many near-optimal values and those with a strong interrelation among them. Hyperparameters of the first category are the windows size θ_{win} and the number of negative samples θ_{ns}. The missing effect of θ_{win} indicates that for this application a larger context is not predictive. Lau and Baldwin report $\theta_{ns} = 5$ being the best choice for the task of duplicate detection and determining semantic textual similarity [8], but performance for other values is not reported. The second category, hyperparameters with a clear optimal value, were found to be the type of architecture and whether hierarchical sampling was used. In our experiments the `dbow` architecture outperformed the `dm` architecture in every scenario (see Fig. 2), which accords with the literature [8–10]. Similarly, classifiers with hierarchical sampling performed better than those without. Thus, these parameters do not require much consideration, as they are binary and one configuration always outperforms the other. The size of the embeddings belongs to the third category, hyperparameters with many reasonable values. Very small embedding sizes ($\theta_d = \{2, 3, 8\}$) have a strong negative impact on the accuracy. But beyond that magnitude, there is a broad range of reasonable values (24 to 256) that can yield good classifiers (Sect. 5.1). The best results are achieved when θ_d is tuned according to the difficulty of the task at hand. On the relatively simple amazon dataset (four classes) an embedding size of 24 to 64 was optimal whereas the 20newsgroups dataset (20 classes) required 64 to 256 dimensions. All the learning hyperparameters fall in the fourth category. The individual parameters from the parameter subset θ_l (i.e., θ_{ts}, θ_α and θ_{epoch}) must be considered jointly and in connection with the desired scenario.

7 Summary

We presented a study on hyperparameters for document classification tasks using document embeddings, concretely doc2vec. Experiments on a text classification task showed that the window size and the number of negative samples have negligible influence, while the `dbow` and hierarchical sampling yield the best performance. For the size of the embeddings vectors there is a range of reasonable values (24–256). Model parameters and learning parameters showed no interrelation and can be tuned separately, while all learning parameters (θ_{ts}, θ_α, θ_{epoch}) must be considered jointly. Those insights can be used in further research or by practitioners to sensibly select initial hyperparameter configurations manually or restrict grid-search or Bayesian optimization approaches, which reduces optimization time substantially.

References

1. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Mach. Learn. Res. **13**, 281–305 (2012)
2. Bergstra, J.S., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Advances in Neural Information Processing Systems, pp. 2546–2554 (2011)

3. Halevy, A., Norvig, P., Pereira, F.: The unreasonable effectiveness of data. IEEE Intell. Syst. **24**(2), 8–12 (2009)
4. Iacobacci, I., Pilehvar, M.T., Navigli, R.: Embeddings for word sense disambiguation: an evaluation study. In: Proceedings of Annual Meeting of the Association for Computational Linguistics, vol. 1, pp. 897–907 (2016)
5. Kusner, M.J., Sun, Y., Kolkin, N.I., Weinberger, K.Q., et al.: From word embeddings to document distances. ICML **15**, 957–966 (2015)
6. Lan, M., Tan, C.L., Low, H.B.: Proposing a new term weighting scheme for text categorization. AAAI **6**, 763–768 (2006)
7. Larochelle, H., Bengio, Y.: Classification using discriminative restricted Boltzmann machines. In: Proceedings of the 25th International Conference on Machine Learning, pp. 536–543. ACM (2008)
8. Lau, J.H., Baldwin, T.: An empirical evaluation of doc2vec with practical insights into document embedding generation. CoRR abs/1607.05368 (2016)
9. Le, Q.V., Mikolov, T.: Distributed representations of sentences and documents. In: Proceedings of International Conference on Machine Learning. JMLR Workshop and Conference Proceedings, vol. 32, pp. 1188–1196 (2014). JMLR.org
10. Liu, Y., Liu, Z., Chua, T.S., Sun, M.: Topical word embeddings. In: AAAI, pp. 2418–2424 (2015)
11. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press, New York (2008)
12. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. CoRR abs/1301.3781 (2013)
13. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in Neural Information Processing Systems, pp. 3111–3119 (2013)
14. Pennington, J., Socher, R., Manning, C.D.: Glove: global vectors for word representation. In: Proceedings of Empirical Methods in Natural Language Processing, pp. 1532–1543. EMNLP (2014)
15. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning). The MIT Press, Cambridge (2005)
16. Sappadla, P.V., Nam, J., Loza Mencía, E., Fürnkranz, J.: Using semantic similarity for multi-label zero-shot classification of text documents. In: Proceedings of European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, vol. ESANN. d-side publications, Bruges, Belgium, April 2016
17. Sebastiani, F.: Machine learning in automated text categorization. ACM Comput. Surv. **34**(1), 1–47 (2002)
18. Snoek, J., Larochelle, H., Adams, R.P.: Practical Bayesian optimization of machine learning algorithms. In: Proceedings of International Conference on Neural Information Processing Systems, pp. 2951–2959. NIPS, USA (2012)
19. Wolpert, D.H.: The lack of a priori distinctions between learning algorithms. Neural Comput. **8**(7), 1341–1390 (1996)