Bootstrapping a Flash Card Retrieval System Without Feedback Data

Ralph Edezhath
redezhath@chegg.com
Chegg, Inc.

José P. González-Brenes
jgonzalez@chegg.com
Chegg, Inc.

ABSTRACT

Online flashcards simulate the traditional experience of having a question on one side, and an answer overleaf. They are a popular tool for students to learn or review material. However, building them from scratch is a time consuming experience for learners. We present `FlashSpace`, a neural architecture that aids study preparation by retrieving relevant flash cards which others students have created. The contributions of this work are two-fold: (i) we propose a way to implement an information retrieval system in the absence of user feedback data, (ii) we leverage peer-learning in the context of study preparation with flash cards. We report our work on a dataset of digital flashcards labelled by students with course and chapter names, where there is an extremely large number of labels which are also highly noisy. Our approach embeds the text of these labels and the flash card content in a common space using a character-level convolutional network. Our empirical evaluation suggests that `FlashSpace` outperforms recent general-purpose neural network models.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; • **Information systems** → *Information retrieval*.

KEYWORDS

peer learning, flash cards, neural networks, text tagging, study retrieval

1 INTRODUCTION

Flash cards—pairs of short questions with their answers overleaf—are a popular tool for memorizing facts, and have measurable impact on learning [1]. Online flash cards are a popular alternative to traditional paper, and have been shown to be as effective as their physical counterpart [2]. Unfortunately, creating flash cards from scratch is a time-consuming activity, even in their online format. For example, students first need to identify useful concepts in a topic that they are studying. Secondly, they have to formulate factual questions from scratch which can assist in learning these concepts. Finally, the students need to type or transcribe the contents of the flash card. This process has to be repeated multiple times for each concept the student wants to study with flash cards. Unfortunately, it is unclear to what extent these burdensome steps are important for learning.

In this work we aim to enable peer-learning, in order to aid students in leveraging the benefits of studying using flash cards created by other learners. We do this by automating the retrieval of existing flash cards relevant to the topic of interest. For example, our working hypothesis is that a learner studying *'infectious diseases'* will find it useful to retrieve flash cards that have been created by peers that have studied this topic. Unfortunately, the task of retrieval is complicated by the fact that student-generated labels are often extremely noisy (eg. a flash card would be labeled 'Chapter 3', instead of a meaningful name such as 'infectious diseases'). Moreover, unlike traditional collaborative filtering or information retrieval systems, we do not have user feedback data. Our main contribution is a neural network architecture that retrieves flash cards learned from user-generated noisy flash card data.

The rest of this paper is outlined as follows. In § 2, we describe the dataset of online flash cards that we use. In § 3, we review prior work. In § 4, we present `FlashSpace`, a model based on convolutional networks that is effective for automatic retrieval of flash cards. In § 5, we evaluate against general-purpose retrieval models, and provide evidence showing that our model allows retrieval of flash cards with significantly higher precision.

2 DATASET

We leverage a dataset of millions of flash cards collected by a leading educational technology firm. These cards are created online by students, who may decide to categorize them into a particular course, which can be further organized into topics. On average, a student creates 147 flash cards, and there are on average 18 topics per course. Examples of topic and course names can be seen in Table 1.

Our model needs to be able to learn from flash cards that are only partially labeled and highly noisy. We argue that a substantial number of topics have labels that are not useful for a retrieval task. While the entire dataset covers many courses for many years, we restrict our experiments to a period of time, and a single subject—biology. For this, we use the heuristic of filtering courses that contain the string "bio", which yields 17.5 million flash cards. Altogether, these cards contain over 296K unique topic labels and 21K unique course names. For this paper, we have anonymized the dataset and removed any personal information.

3 RELATION TO PRIOR WORK

A variety of successful models have been proposed in the 'learning to rank' [3, 4] and 'collaborative filtering' [5] literature—these presuppose a list of items with user feedback, for example, documents

, ,

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Table 1: Examples of flash cards

Card front	Card back	Topic	Course
what bacteria does the camp test allow you to identify?	streptococcus agalactiae	lab quiz 8: respiratory tract	microbiology 273
large intestine (colon)	largest diameter intestine 1.absorption of water and minerals don't absorb water = diarrhea. 2.bacterial fermentation (gas production)	ch3 part 2	biology 126
aden(o)- aden(i)-	relating to a gland	prefixes and suffixes	biology 2401
opiod peptide	a type of endogenous peptide that mimics the effects of morphine in binding to opioid receptors and producting marked analgesia and reward	chapter 4: chemical bases of behavior	bio. psych.

that have been clicked. But in our case, each set of labels is associated with a set of flash cards with no user feedback. In fact, the online flash card platform does not yet allow for students to retrieve other users' flash cards based on topic queries. These learning to rank models may be useful after deployment of our retrieval system at scale and a significant amount of user interaction data has been collected.

As discussed in the previous section, our dataset contains an extremely large number of labels. There is a rich literature on 'extreme multi-label learning', where models are designed to predict a small subset of labels out of a very large number. These models include 1-vs-all classifiers [6], local embeddings [7] and tree-based models[8]. While these models achieve impressive performance, to our knowledge they are limited to a fixed set of labels. In contrast, we aim to retrieve flash cards based on free text entered by students, rather than pick from a fixed set—thus our model must be able to predict on unseen topic and course labels. Of the unique topic labels in our dataset, 90.3% are not shared among students, i.e. occur in the flash cards created by at most one student.

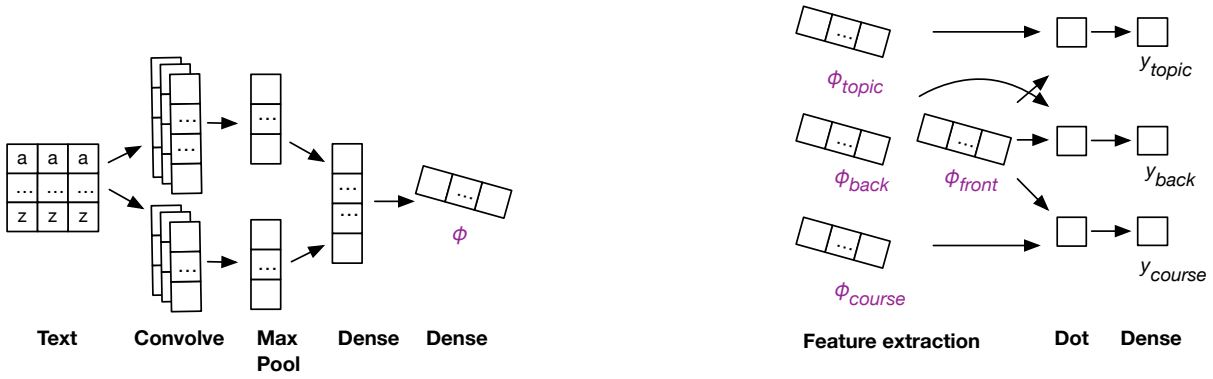
Self-supervised methods are a promising way to address the cold-start problem of flash card retrieval. They typically work by projecting the one-hot encodings of words from a document into an *embedding*—a smaller dense vector. Documents (in our case flash cards), and their labels, can be embedded into a common space and compared by a distance function. Thus, one can choose the label that is closest to the document.

A recent self-supervised method called `fastText` [9] reportedly achieves state-of-the-art performance on word similarity and analogy tasks and enables inference for words not seen during training by using sub-word information. On the other hand, self-supervised methods that incorporate contextual information into word embeddings have achieved state-of-the-art performance on language modeling tasks [10, 11]. Due to time constrants, and since both our labels and documents are typically short sentence fragments, we have not evaluated these models. A different line of work includes `Feat2Vec` [12], a generic approach for learning self-supervised embeddings of features (not necessarily words). It works by iterating over each feature and treating it as if it were the target variable of the model (using the remaining features to predict it)—the authors prove that this is equivalent to optimizing a convex combination of predictive models with different targets. `FlashSpace` builds on the self-supervised literature; it differs from `Feat2Vec` in two ways (i) it uses a simpler multi-objective optimization target, and (ii) the output tasks are manually engineered. In § 5, we compare to different variations of `fastText`; however, a limitation of our study is that we do not compare against `Feat2Vec` or BERT [11] because of time-constraints.

4 FLASHSPACE

A retrieval system for flash cards, in order to provide a good user experience, must run in real time. This means that the model most be able to perform inference on millions of samples in less than a second. This can be achieved by requiring that the model factorizes into 'query' and 'document' parts which are computed independently, and where the output of these parts are combined by a simple operation, such as a dot product, to form the final prediction. For models following this design, the vectors representing all the flash cards can be pre-computed, and in production we only compute the output of the 'query' section of the model, and calculate the dot-products with the stored vectors.

We architect `FlashSpace` to represent each flash card with four feature groups—the card front, back, topic and course. We use *convolutional neural networks* (CNNs) to learn representations of each of these features, since they have been shown to be effective for various natural language tasks [13–15]. Our motivation is that character-level CNNs can learn sub-word features, which can also generalize to unseen labels. This is useful since we have an extremely large number of labels which are not shared among users. Due to time constraints, we did not tune the hyper-parameters but followed recommendations from a previous study on tuning CNNs [16].



(a) Feature extraction function ϕ

(b) Network architecture (feature extraction omitted)

Figure 1: FlashSpace architecture

Figure 1 depicts the architecture of the model. To simplify the diagram, we separate the feature extraction functions from the rest of the model. Our model consists of 4 feature extraction functions, one for each feature, which we denote as - $\phi_{front}, \phi_{back}, \phi_{topic}, \phi_{course}$. Figure 1a depicts these feature extraction functions, which do not share parameters among each other, but have a nearly identical structure—varying only in the length of text that they accept:

- Each text is represented by one-hot encoding the characters from within an allowed set of 37 (alphabets, numbers and space) and up to a maximum length of L characters. The length L is fixed to cover at least 90% of flash cards. Thus each string is represented as a $37 \times L$ matrix. The strings are padded with 3 spaces at their start and end, and strings longer than the L are truncated. L is set to 256 characters for the front and back, and 64 for the topic and course names.
- The next layer is a set of convolutional filters [17] (also called feature maps), which are used to extract features from the characters. A filter is a 2-dimensional matrix which is applied to m adjacent characters. Thus each of our filters is a $37 \times m$ matrix of learned parameters. Filters are applied by computing the element-wise dot product of the filter along a sliding window of the entire input. The resulting output for each filter is a vector of length $L-m+1$. The ReLU activation, which is the $max(0, x)$ function, is applied to each output. In order to learn both longer and shorter n-gram features, the network has 1024 filters each of width 4 and 8 characters.
- The 1-max pooling function is applied to the output of each filter, which means we take the largest value from the $L-m+1$ outputs generated by each filter. The pooled outputs from the two sets of filters is concatenated to form a vector of length 2048.
- Next we learn higher-level features from these filters with a dense layer of size 4096 with ReLU activations.
- The next layer is a dropout layer [18], which is a type of regularization that for each mini-batch randomly drops units with a specified probability, which we set to 0.2.
- the final embedding for the feature is computed by a dense layer of size 256 with PReLU [19] activations

Figure 1b shows the rest of the architecture of FlashSpace. Our goal is to compute an embedding for the 'query' section of the model (topic), and compare its similarity with the embedding of 'documents', which we compute in different combinations of the remaining features as described in § 5.4.

We use multi-objective optimization as shown in shown in Table 2: for each pair of features, we take the dot product of their embedding layers to compute their similarity, followed by a sigmoid activation function. These embedding layers are calculated from the text input by the feature extraction functions.

Learning the parameters of probabilistic neural models that predict a large number of classes can be computationally costly since it typically requires summing over the entire number of labels on each gradient descent update. A common work-around is to use negative sampling or noise contrastive estimation [20]: we first transform our multi-class classification problem (i.e., which card to show given a query?) into binary classifications (i.e., are the query and the card a match?); we then generate random samples of queries and cards and label them as negative examples. In other words, a positive example is a flash card observed in the training dataset, and a negative example is generated by sampling one of the features. We now explain how we learn the parameters of multiple tasks using negative sampling. For the negative examples for each learning task, we randomly sample a value for Feature 2 (see Table 2) according to its distribution in the training set. Thus, the number of negative examples thus generated is balanced with the number of positive examples. The parameters are learned by gradient descent using the Adam optimizer [21] and we implement the model using the `Tensorflow` [22] package.

5 EMPIRICAL RESULTS

In this section we evaluate how well FlashSpace retrieves flash cards, and benchmark it to baselines. For this, we compare the performance of our model to TF-IDF, since it is the foundation of retrieval algorithms in popular search engines such as SOLR [23]. We also compare with `fastText`, since it is the state of the art for general-purpose word embeddings. We evaluate these models on the task of retrieving the correct flash card given its topic label.

Table 2: Learning Tasks

Feature 1	Feature 2 (randomly sampled)	Task output
Card front	Topic	$\sigma\left(\phi_{front}(f) \cdot \phi_{topic}(t)\right)$
Card front	Course	$\sigma\left(\phi_{front}(f) \cdot \phi_{course}(c)\right)$
Card front	Card back	$\sigma\left(\phi_{front}(f) \cdot \phi_{back}(b)\right)$

The retrieval of flash cards by course labels is not studied in this work, since we have restricted the dataset to only biology courses, so there is considerably less variety in these labels.

For each model, we calculate a topic (i.e. query) vector, and a document vector which combines the text of the flash card in different ways. The similarity (dot-product) between the query and document vectors is used to rank the flash cards in a held-out set. This test set is queried only once, and was created by randomly sampling 1% of the data ($\sim 175k$ cards). However, some evaluation metrics require negative examples; for this, we created an extended test set that includes incorrect flash cards for each topic. We generated these negative examples by sampling flash cards from the test set.

We evaluate models on two metrics—area under the receiver operating curve (AUC) and precision at top-K. The AUC requires negative examples, and thus we use the extended test set, which has a balanced number of positive and negative samples. For precision at top-K, if the correct flash card is contained within the first K cards, the sample receives a score of 1, or 0 if it is not within this set. Note that the scores calculated here will underestimate the true relevance for a user, since many flash cards may have relevant content but different labels.

We run each model for 10 epochs - while the number of epochs could be optimized with a validation set, we leave this for future work. We provide details of how we set-up the TF-IDF, `fastText` and `FlashSpace` experiments below. The results are summarized in Table 3.

5.1 TF-IDF

To assess the effectiveness of straightforward keyword search, we evaluate ranking flash cards by TF-IDF [24], a popular method for document retrieval. This means that for a given vocabulary of terms, we first compute the inverse document frequency (IDF) - the fraction of documents containing the term, scaled logarithmically. Then for each query or document, we form a TF-IDF vector by scaling the counts of each term within the document, by its IDF, and then normalize the vector. The 'documents' are the words of the front, back and course name concatenated. For a given topic, we compute the TF-IDF vector and rank the documents by the dot product with the documents' TF-IDF vectors. We use the implementation of TF-IDF in the `scikit-learn` [25] package. We set the vocabulary size to 10000, and build the vocabulary and compute the IDF for each term only using the documents in the training set.

5.2 fastText

We use the implementation of FastText available in the Python package `gensim` [26]. This model learns dense representations for words that incorporate subword information - each word is a sum of the vectors of its n-grams and a single token representing the whole word. During training, we concatenate the words from the flash card front, back, topic and course labels to form a document. The embedding dimension for the word vectors was set to 256, the same size as the vectors generated by `FlashSpace`. The word 'window', which is the maximum distance between the reference and predicted word, was set to 21 - this was chosen because 50% of flash cards have less than 21 tokens. The remaining hyper-parameters are set to their default values.

For testing, we construct a vector for the topic and a vector for the flash card 'document' which is the text of the front, back and course name concatenated. To form the document vector, we take the average of the vectors for all its tokens. The flash cards are then ranked by similarity of their document vectors to the topic vector being queried for.

We evaluated 3 variations of `fastText`:

- The topic and document vectors are formed by taking the average of all the constituent tokens.
- The model is trained with the topic vector concatenated with a special character to form a single token. The token vectors are normalized and averaged for the document
- The constituent token vectors are normalized, and averaged to form both the document and the topic vectors

The results for these are listed as `fastText` averaged, `fastText` concatenated and `fastText` normalized respectively, in Table 3. In each case the model was trained for 10 epochs.

5.3 IDF weighted `fastText`

We hypothesize that the under-performance on `fastText` relative to TF-IDF was due to the fact that when the word vectors are normalized, all words are weighted equally, which given our highly noisy dataset is ineffective. On the other hand, using non-normalized vectors is not effective since the word lengths learned by `fastText` are not tuned to the task at hand. To address this issue, we weight the normalized `fastText` vectors by the IDF weights computed in § 5.1. When a word is not found in the vocabulary of TF-IDF, we use the average IDF across all words in the vocabulary.

5.4 `FlashSpace`

As in the case of `fastText`, we train the model for 10 epochs and use an embedding size of 256 for each of the features. The output of ϕ_{topic} is used to generate the query vector, and the flash cards are ranked by dot-product similarity to this vector. We evaluated two ways of forming the document vector using our model

- Only the text of the 'front' of the flash card, embedded using ϕ_{front}, is used as the document vector
- The vectors for front, back and course name of the card are computed using ϕ_{front},ϕ_{back} and ϕ_{course} respectively, and averaged to form the document vector.

Table 3: Evaluation of models. Rounded at two most significant digits

	Multiple-word weight	AUC	Precision at Top-5	Precision at Top-10
Random baseline	-	0.50	0.00%	0.00%
BOW	TFIDF	0.61	0.28%	0.59%
`fastText` averaged	uniform	0.57	0.00%	0.00%
`fastText` concatenated	uniform	0.58	0.06%	0.01%
`fastText` normalized	uniform	0.57	0.09%	0.20%
`fastText`	TFIDF	0.58	0.39%	0.66%
`FlashSpace` - Front Vector only	-	**0.92**	0.34%	0.62%
`FlashSpace` - all features	-	0.89	**0.52%**	**1.2%**

5.5 Discussion

As shown in Table 3, `FlashSpace` outperforms both `fastText` and TF-IDF by a large margin on all metrics. We notice that precision at top-K and AUC are not perfectly correlated—this means that a model that performs best across the whole distribution may not necessarily have the best performance at the top of the list. Precision at the first few elements has a far greater impact on the user experience, so this metric is of greater importance for our use case. We find that weighting `fastText` word vectors according to IDF significantly improves performance at top K.

6 CONCLUSION

We propose `FlashSpace`, an architecture for representing flash cards. We use it to bootstrap the implementation of a new retrieval system in the absence of user-relevance feedback. We compare it to the time-tested TFIDF baseline, as well to a recent sub-word embedding model. Our experimental results suggest that `FlashSpace` can achieve substantially better performance.

Future work may study the reasons why `FlashSpace` has better precision—are the task-specific embeddings important? Or is it the way it combines multiple words into a single entity? We are particularly interested in the relationship of `FlashSpace` with general-purpose self-supervision frameworks, such as `Feat2Vec`. Furthermore, future research may compare against additional baselines, specially once user feedback is available. Additionally, it may be interesting to study issues related to content quality of flash cards.

Although we focused on flash cards in this study, the task of bootstrapping a new retrieval system is very prevalent and we hope it could be interesting to a wider audience. We also hope that the general structure of our model—convolutional networks learned through classification of pairwise features—could be useful in other scenarios where the documents have an extremely large number of partial or highly noisy labels.

REFERENCES

[1] Jonathan M. Golding, Nesa Wasarhaley, and Bradford Fletcher. The use of flashcards in an introduction to psychology class. *Teaching of Psychology*, 39:199–202, 06 2012.
[2] Gilbert Dizon and Daniel Tang. Comparing the efficacy of digital flashcards versus paper flashcards to improve receptive and productive l2 vocabulary. *The EuroCALL Review*, 25(1), 2017.
[3] Tie-Yan Liu et al. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331, 2009.
[4] Christopher Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Gregory N Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine learning (ICML-05)*, pages 89–96, 2005.
[5] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009.
[6] Rohit Babbar and Bernhard Schölkopf. Dismec: Distributed sparse machines for extreme multi-label classification. In *Proceedings of the tenth ACM international conference on web search and data mining*, pages 721–729. ACM, 2017.
[7] Kush Bhatia, Himanshu Jain, Purushottam Kar, Manik Varma, and Prateek Jain. Sparse local embeddings for extreme multi-label classification. In *Advances in neural information processing systems*, pages 730–738, 2015.
[8] Himanshu Jain, Yashoteja Prabhu, and Manik Varma. Extreme multi-label loss functions for recommendation, tagging, ranking & other missing label applications. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 935–944. ACM, 2016.
[9] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.
[10] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proc. of NAACL*, 2018.
[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
[12] Luis Armona, Ralph Edezhath, and José P. González-Brenes. Beyond word embeddings: Dense representations for multi-modal data. In *Proceedings of the Thirty-Second International Florida Artificial Intelligence Research Society Conference, FLAIRS 2019*. AAAI Press, 2019.
[13] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *CoRR*, abs/1404.2188, 2014.
[14] Jason Weston, Sumit Chopra, and Keith Adams. #tagspace: Semantic embeddings from hashtags. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1822–1827, 2014.
[15] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657, 2015.
[16] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 253–263. Asian Federation of Natural Language Processing, 2017.
[17] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
[18] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.
[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.
[20] Chris Dyer. Notes on noise contrastive estimation and negative sampling. *arXiv preprint arXiv:1410.8251*, 2014.
[21] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
[22] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al.

Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[23] Dikshant Shahi. Solr scoring. In *Apache Solr*, pages 189–207. Springer, 2015.

[24] Gerard Salton and Chris Buckley. Term weighting approaches in automatic text retrieval. Technical report, Cornell University, 1987.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[26] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. http://is.muni.cz/publication/884893/en.