Proceedings of the LREC2020 Industry Track , pages 21–25
Language Resources and Evaluation Conference (LREC 2020), Marseille, 11–16 May 2020
© European Language Resources Association (ELRA), licensed under CC-BY-NC

On the Importance of Text Classification Pipeline Components for Practical Applications: A Case Study

Andrej Švec, Katarína Benešová, Marek Šuppa
Slido
Bratislava, Slovakia
{asvec, kbenesova, msuppa}@slido.com

Abstract

The worlds of academia and industry have different priorities for machine learning models. In the academic world, the model's performance is often the main focus, whereas finding the balance between the model's performance, resource requirements, and the ease of its deployment is often deemed more important in the production environment of the industry. In this paper we consider a real world text classification problem, compare the specifics of different parts of natural language processing pipelines and investigate their contribution to the final model's performance. We also take into consideration the practical aspects of the model's use and deployment, such as the size of the model and preprocessing time. Our case-study shows that in this particular scenario the performance of simpler models can be on par with the more complex ones. We find this result valuable, as simpler and smaller models are normally also easier to deploy in practice, e.g. in a serverless environment. To showcase the practical usefulness of our final model, we deploy it to AWS Lambda and show that its execution time in this environment scales linearly with the input text's length.

Keywords: text classification, NLP pipeline, production environment, serverless

1. Introduction

Text classification is one of the most common use-case of Natural Language Processing (NLP) models. The choice of a model type and architecture varies significantly from task to task. It is also greatly affected by the requirements for the final model. In production environments, these often comprise training time, latency and resource requirements (for instance CPU, memory or model size).

Many academic and state-of-the-art works propose solutions incorporating different steps of preprocessing, data augmentation and large model architectures, rendering them time-consuming and resource-intensive to develop and train (Strubell et al., 2019). On the other hand, there is a great demand for simplicity and ease of setup in the production environment, which renders some of the academic-world solutions unsuitable for real-world use.

In this paper we compare several approaches to a specific instance of text classification. Given its practical applicability, we also investigate the deployment of the final model in a serverless environment.

1.1. Use-case prediction task

We compare these approaches in a real-world multi-class classification task. The task is to distinguish between different types of use-cases in which an online Q&A application[1] is used.

We call a single usage of the Q&A application an *event*. Thus, the task is to predict the use-case of an event by looking at the data associated with it. A use-case in this context refers to the types of events at which this Q&A system gets used, such as a conference, a team meeting or a lecture at a university.

Formerly the events used to be classified into use-cases by a set of 118 expertly chosen keyword-based regular expressions matched against the event name. This approach is based on the assumption that a keyword (for instance *conference*) in the event name implies that the Q&A system was used at a specific type of event (in this case a conference). A sample of these regular expressions is shown in Table 1. The main disadvantage of this approach lies in its low coverage (around 33%), as not every event name contains a keyword. For example, an event with the name *LREC 2020* would not be classified as *conference*, despite being an abbreviation for *Language Resources and Evaluation Conference 2020*.

Use-case	Regular expressions
Conference	`/conference/`, `/summit/`
Company meeting	`/all(\s*)hand/`, `/staff(.+)meet/`
Team meeting	`/team(\s*)meet/`, `/team(\s*)sync/`
Learning	`/workshop/`, `/l&d/`

Table 1: A sample of regular expressions used for use-case classification, based on a match against the event name.

1.2. Serverless environment

Serverless (Function-as-a-Service) environments provide a simple way of deploying code to production environments by providing an abstraction over many standard operational concerns. They are particularly suitable for stateless applications, such as small machine learning models (Ishakian et al., 2018). Furthermore, they require minimal maintenance, as the cloud infrastructure takes care of provisioning their computational resources, as well as ensuring that they only run when necessary. On the other hand, there are numerous practical limitations as to what can be currently deployed in a serverless environment. These are mainly related to stor-

[1] In our case the Q&A application is called Slido, reachable at `https://slido.com`

age (250 - 500 MB) or RAM (2 - 3 GB), forcing the models to be small in size and to also have a small RAM footprint.

2. Related work

Text classification (sometimes also referred to as text categorization) is a widely studied technique in Natural Language Processing (Aggarwal and Zhai, 2012). Due to its great versatility, numerous practical tasks can be viewed as text classification problems. This results in a wide variety of applications, ranging from linguistically-inspired problems, such as Word Sense Disambiguation (Raganato et al., 2017), classification of sentiment or opinion polarity of documents (Liu, 2015), to direct downstream applications in which text classification can be helpful, such as online suicide prevention (Desmet and Hoste, 2018).

In broad terms, the text classification pipeline can be viewed as a three stage process: at first the *pre-processing* part ensures that the inherent noise gets removed from the raw input text, then the *feature extraction* part converts the clean input text into representation that is then taken as the input of the final *classification* part, in which a classification algorithm predicts the category of the text using the provided representation.

The standard approaches in *pre-processing* relevant to the presented work include tokenization (Verma et al., 2014) and utilization of pre-trained models for extracting part of speech tags (Batanović and Bojić, 2015) as well as named entities (Trask et al., 2015) from the input text and using this information in further parts of the pipeline.

There exist a multitude of options for feature extraction, such as Bag-of-Words, TF-IDF (Jones, 1972), word embeddings, e.g. word2vec (Mikolov et al., 2013), GloVe (Pennington et al., 2014), FastText (Bojanowski et al., 2017) and various contextualized word representations (Peters et al., 2018). A different approach to representing text is called sentence embeddings. These aim to provide semantically meaningful vector representations of sentences, comparable using cosine distance. Some of the popular methods include Skip-Thought vectors (Kiros et al., 2015), InferSent (Conneau et al., 2017) and the Universal Sentence Encoder (Cer et al., 2018).

A sizable body of work has been published on the topic of classification algorithms. Popular choices include logistic regression (Lee and Liu, 2003), Naïve Bayes Classifier (Kim et al., 2006), Support Vector Machines (Kwok, 1998) as well as various approaches based on neural-networks (Joulin et al., 2017), particularly in combination with Deep Learning approaches (LeCun et al., 2015).

3. Dataset

Our dataset is composed of 80 000 events. An event consists of various types of interactions with the Q&A application, incorporating mainly questions posted by the audience to the presenter, as well as the live polls used by the presenter to find out the opinion of the audience about a topic.

Only a subset of the interaction data has been selected for the purpose of our task. A preview of such data can be seen in Figure 1. The event data can be categorized as follows:

- **Aggregated features.** These comprise high-level information about the events, for instance the number of participants, the duration of the event or the number of posted questions and activated polls.
- **Textual features.** The texts of the questions and polls belonging to the events. All the texts are in English language. We exclude the event name from the texts to prevent the model from essentially extracting the regular expressions from the event name.

Aggregated: participants=95, n_questions=52, duration_days=1, . . .
Questions: "How much time / hours have you saved so far in these 23 processes?", "How do you pay invoices? Do you have any automated process?", . . .
Polls: "You are from", "How did you like this talk?", "What do you want me to elaborate more on?", . . .
Label: Conference

Figure 1: An example of data about event.

Text length. The length of a question is limited to 160 characters in most cases but a few events use a maximum length of up to 300 characters. The number of questions ranges from zero to a few thousands, with a median of 16.

A poll consists of the question asked to the audience and either poll options or audience replies (for a multiple-choice poll or open-text poll, respectively). A poll question can have up to 256 characters. The number of polls in an event ranges from zero to a few thousand, with a median of 1.

The distribution of number of tokens per event is shown in Figure 2. It suggests that the texts we are trying to classify can easily contain hundreds of words.

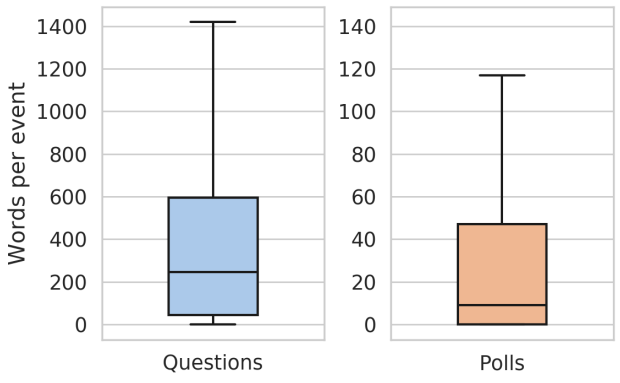

Figure 2: The distribution of the number of words in concatenated questions and polls in our dataset.

Labels. The events are assigned a use-case based on the matching regular expressions described in Section 1.1. We exclude events that are assigned more than one use-case, as they are not numerous and doing so would needlessly turn our task into multi-label classification.

Our task considers four use-cases: *company meeting* (37% of data), *conference* (31%), *learning* (17%) and *team meeting* (15%). The main deficiency of these classes is the inevitable blending of team meetings into other use-case cat-

egories. Since some companies are smaller and some are larger, a meeting with 20 attendants can either be a team meeting or a whole company meeting depending on the size of the company. We also note the inherent limits of this labeling scheme, as an event where a team is participating in a workshop is both a team meeting and a learning event.

Train-test split. We split the dataset into train and test parts using a stratified split with the 80 : 20 ratio.

4. Method

We experiment with multiple aspects of NLP pipelines. In order to measure the importance of an aspect, we conduct an ablation study by keeping the model fixed and only altering the studied aspect. Model performance is measured by calculating the accuracy and the micro-averaged F_1 score of the model on the test set.
Two types of models were used in our experiments:

- **FastText model** (Joulin et al., 2017). The inputs to the model are concatenated and preprocessed texts of all questions and all polls. The model does not consider the aggregated features.
- **Two-stage pipeline model.** The first step produces a document embedding for both questions and polls and in the second step we take the embeddings along with the aggregated features and classify them using a single-layer softmax neural network. The document embedding layer is trained on a concatenation of preprocessed questions and polls. The training of this layer takes place separately from the rest of the network.

4.1. Preprocessing

To assess the effect of various preprocessing approaches, we study their effect on the final performance, using the **FastText model** for the classification part. We focus on the following areas of preprocessing:

Tokenization. Three types of tokenizers are compared: simple whitespace with special characters striping, spaCy (Honnibal and Montani, 2017) and Bling Fire[2].

Part-of-speech (POS) tags. We use the spaCy POS tagger to assign POS tags to each token. The tokens are then concatenated with their POS tag, e.g. `work|NOUN` or `work|VERB`, and used for training similarly to the `sent2vec` method (Trask et al., 2015).

Named entities (NE). The spaCy NLP pipeline can also find NEs in a text. We leverage this information to join the tokens associated with a NE into a single token and concatenate the joined token with the NE type, e.g. `01-04-1997|DATE`. It is also possible to use both POS and NE tags simultaneously. If this is the case, NE tags take precedence over POS tags.

4.2. Embedding types

We also experiment with different types of embeddings used in the **two-stage pipeline** model:

- **Fast sentence embeddings (FSE)** (Borchers, 2019). We base the FSE embeddings on FastText word embeddings (Bojanowski et al., 2017) that are then composed into higher-level embeddings. The FSE library allows for three composition methods: Deep Averaging Network (DAN) (Iyyer et al., 2015), Smooth Inverse Frequency (SIF) (Arora et al., 2017) and Unsupervised Smooth Inverse Frequency (USIF) (Ethayarajh, 2018).
  The main advantage of the FSE embeddings is their fast computation speed.
- **Universal sentence encoder (USE) embeddings** (Cer et al., 2018). We experiment with multiple flavors of pre-trained USE models available on TensorFlow Hub[3]. The flavors of the USE model differ mainly in size and performance, but also in the maximum number of tokens considered by the model: the *large* flavor based on the Transformer architecture (Vaswani et al., 2017) supports a maximum of 128 tokens whereas the *default* flavor based on DAN is unbounded.
  The advantage of the USE model is that is was trained on a large corpus resulting in a fair coverage of different domains. However, unless the model is fine-tuned on the target domain, its performance can be lower than that of a dataset-specific model.
- **FastText embeddings.** We use the average of the FastText tokens in the texts to create the sentence embedding as specified in (Joulin et al., 2017).
  A possible advantage of the FastText embeddings is that they can be trained in a supervised manner.

5. Results

In this section we present and discuss the results of different experiments described in Section 4. We group these results according to the aspects described above.

Method	Size	Time	Acc	F_1
Whitespace	-	4.5s	0.656	0.642
Bling Fire	1.4MB	12.6s	0.650	0.628
spaCy	80KB	6m 1s	0.657	0.642
spaCy+POS	3.7MB	42m 34s	0.654	0.636
spaCy+NE	4.0MB	1h 1m	0.656	0.644
spaCy+POS,NE	7.7MB	1h 26m	0.653	0.635

Table 2: Performance of the different preprocessing methods in combination with the FastText model on the test set. We also report the size of the preprocessing model and the time needed for preprocessing of the training set due to their implications for practical usage.

5.1. Preprocessing

Table 2 shows the effect of testing various preprocessing approaches on the **FastText model**. The simple method incorporating a whitespace tokenizer and punctuation striping is competitive with more complex methods like Bling

[2] `https://github.com/microsoft/BlingFire`

[3] `https://tfhub.dev/google/collections/universal-sentence-encoder/1`

Fire or spaCy tokenizers. Furthermore, the latter methods require increased preprocessing time. When they do POS and NE tagging on top of tokenization, they can be up three orders of magnitude slower than the baseline.
Models trained on data with tags (especially POS) also take longer to converge, which can be seen in Figure 3. We hypothesize that this can be caused by the model needing more epochs to cope with increased vocabulary size.

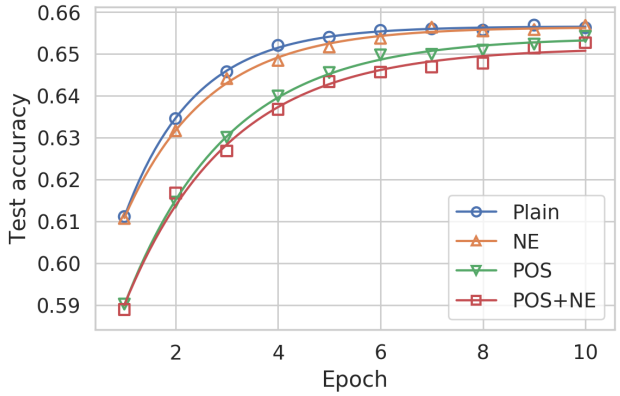

Figure 3: Test accuracies of models using various token tags at different points of the training process.

5.2. Embedding types

Table 3 shows the performance comparison of the **two-stage pipeline model** using different types of embeddings trained on texts preprocessed using whitespace tokenizer and punctuation stripping.

Embedding type	Dim	Size	Acc	F_1
FSE DAN	300	2.5GB	0.590	0.595
FSE SIF	300	2.5GB	0.585	0.593
USE default	512	1.0GB	0.575	0.584
USE large	512	850MB	0.557	0.562
FT unsupervised	150	1.3GB	0.584	0.590
FT supervised	**20**	**180MB**	**0.645**	**0.647**

Table 3: Performance of the different models on the test set along with the respective embedding dimensions. FT stands for FastText.

Despite using the largest embedding dimension and being trained on the largest corpus, the USE model achieves the lowest performance on our task. We hypothesize that this is due to its training corpora being quite different from the one available for our task.
We note that the *default* DAN-based USE flavor performs better than the *large* Transformer-based flavor. This seems to be due to the Transformer limiting the maximum number of tokens to 128. With the median of 285 words per event, the words necessary for discrimination between classes may not be considered by the Transformer.
The FSE and FastText unsupervised models reach higher performance than USE. We think that this is because they are trained on our data and manage to capture its specifics.
The model based on supervised FastText embeddings reaches the highest performance. We assume this stems from the fact that the supervised embeddings are tailored to the task. As a result, the embedding layer filters out unimportant words and makes the task of the classifier simpler.

Embedding size. We observe that unsupervised embedding models require much higher embedding size than the supervised model. It seems that only a fraction of the features captured in the unsupervised embeddings is finally useful for the classification task.

5.3. Discussion and Deployment

The results listed in Section 5 show that the F_1 score is the highest in case of the two-stage pipeline model which uses supervised FastText embeddings. On the other hand, a simple FastText model (Table 2) trained end-to-end consistently reaches higher accuracy than various two-stage pipelines (Table 3). We find this particularly notable, as it utilizes a subset of the available data: the model only considers the texts of questions and polls associated with an event. Furthermore, thanks to its simplicity and small size (300 MB in RAM), we managed to deploy this model in the AWS Lambda severless environment. As Figure 4 shows, its execution time scales linearly up to 25 000 tokens.

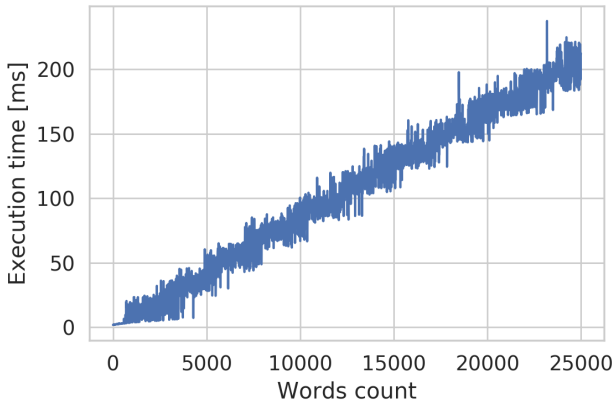

Figure 4: Execution time of the FastText model deployed in AWS Lambda as a function of the input text's length.

6. Conclusion

In this work we present a case study of a text classification task on longer texts. We evaluate the performance of various preprocessing and feature representation approaches, and show that an end-to-end trained FastText model is able to match the performance of more complex pipelines. To showcase its practicality, we deploy it to AWS Lambda.
As the resulting model is still quite large, quantization methods could be used to further reduce its size (Joulin et al., 2016). Another interesting avenue of future research would be distilling the text classification model back to regular expressions (Bui and Zeng-Treitler, 2014).

7. Bibliographical References

Aggarwal, C. C. and Zhai, C. (2012). A survey of text classification algorithms. In *Mining text data*, pages 163–222. Springer.

Arora, S., Liang, Y., and Ma, T. (2017). A simple but tough-to-beat baseline for sentence embeddings. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Batanović, V. and Bojić, D. (2015). Using part-of-speech tags as deep-syntax indicators in determining short-text semantic similarity. *Computer Science and Information Systems*, 12(1):1–31.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Borchers, O. (2019). Fast sentence embeddings. `https://github.com/oborchers/Fast_Sentence_Embeddings`.

Bui, D. D. A. and Zeng-Treitler, Q. (2014). Learning regular expressions for clinical text classification. *Journal of the American Medical Informatics Association*, 21(5):850–857, 02.

Cer, D., Yang, Y., Kong, S.-y., Hua, N., Limtiaco, N., John, R. S., Constant, N., Guajardo-Cespedes, M., Yuan, S., Tar, C., et al. (2018). Universal sentence encoder. *arXiv preprint arXiv:1803.11175*.

Conneau, A., Kiela, D., Schwenk, H., Barrault, L., and Bordes, A. (2017). Supervised learning of universal sentence representations from natural language inference data. *arXiv preprint arXiv:1705.02364*.

Desmet, B. and Hoste, V. (2018). Online suicide prevention through optimised text classification. *Information Sciences*, 439:61–78.

Ethayarajh, K. (2018). Unsupervised random walk sentence embeddings: A strong but simple baseline. In *Proceedings of The Third Workshop on Representation Learning for NLP*, pages 91–100, Melbourne, Australia, July. Association for Computational Linguistics.

Honnibal, M. and Montani, I. (2017). spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear.

Ishakian, V., Muthusamy, V., and Slominski, A. (2018). Serving deep learning models in a serverless platform. In *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pages 257–262. IEEE.

Iyyer, M., Manjunatha, V., Boyd-Graber, J., and Daumé III, H. (2015). Deep unordered composition rivals syntactic methods for text classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1681–1691, Beijing, China, July. Association for Computational Linguistics.

Jones, K. S. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*.

Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., and Mikolov, T. (2016). Fasttext.zip: Compressing text classification models. *CoRR*, abs/1612.03651.

Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2017). Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 427–431. Association for Computational Linguistics, April.

Kim, S.-B., Han, K.-S., Rim, H.-C., and Myaeng, S. H. (2006). Some effective techniques for naive bayes text classification. *IEEE transactions on knowledge and data engineering*, 18(11):1457–1466.

Kiros, R., Zhu, Y., Salakhutdinov, R. R., Zemel, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Skip-thought vectors. In *Advances in neural information processing systems*, pages 3294–3302.

Kwok, J. T.-Y. (1998). Automated text categorization using support vector machine. In *In Proceedings of the International Conference on Neural Information Processing (ICONIP*. Citeseer.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436–444.

Lee, W. S. and Liu, B. (2003). Learning with positive and unlabeled examples using weighted logistic regression. In *ICML*, volume 3, pages 448–455.

Liu, B. (2015). *Sentiment analysis: Mining opinions, sentiments, and emotions*. Cambridge University Press.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Raganato, A., Camacho-Collados, J., and Navigli, R. (2017). Word sense disambiguation: A unified evaluation framework and empirical comparison. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 99–110.

Strubell, E., Ganesh, A., and McCallum, A. (2019). Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*.

Trask, A., Michalak, P., and Liu, J. (2015). sense2vec-a fast and accurate method for word sense disambiguation in neural word embeddings. *arXiv preprint arXiv:1511.06388*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *CoRR*, abs/1706.03762.

Verma, T., Renu, R., and Gaur, D. (2014). Tokenization and filtering process in rapidminer. *International Journal of Applied Information Systems*, 7(2):16–18.