Optimality of state-of-the-art language representation models in NLP applications

Author:

Larisa Poghosyan

larisa_poghosyan@edu.aua.am

BS in Data Science

American University of Armenia

Yerevan, Armenia

Supervisor:

Vahe Hakobyan

vahe.hakobyan@zalando.de

.

Abstract

Pre-trained models like BERT, Word2Vec, FastText, etc., are widely used in many NLP applications such as chatbots, text classification, machine translation, etc. Such models are trained on huge corpora of text data and can capture statistical, semantic, and relational properties of the language. As a result, they provide numeric representations of text tokens (words, sentences) that can be used in downstream tasks. Having such pre-trained models off the shelf is convenient in practice, as it may not be possible to obtain good quality representations by training them from scratch due to lack of data or resource constraints. That said, in the practical setting, such embeddings are often used as inputs to models to serve the purpose of the task. For example, in a sentence classification task, it is possible to use a Logistic Regression on the top average Word2Vec embeddings. Using such embeddings on real-life industrial problems could produce some optimistic improvements over baselines; however, it is not clear whether those improvements are reliable or not. In our study, we intend to check the question at hand by formulating multiple applicable and viable tasks in the industry and replicating the workflow of data scientists. Our goal is to construct various models (different in sophistication) that use embeddings as inputs and use a methodology to report the confidence bounds of the metric of interest. With this experiment we hope to develop an understanding of the phenomenon of having optimal results on the paper that might not be optimal in reality; thus, aiming to find a reliable method, that will aid the decision making process and facilitate the model selection.

1. Introduction

Many NLP applications rely on word vector representations, where a vector is the projection of a word into a continuous vector space. Some of the well-known deep learning architectures that output those embeddings, like Word2Vec, FastText, ELMo and BERT have already been established as state-of-the art. Word2Vec (Mikolov et al., 2013) introduced a task of word similarity, representing words as continuous

vectors. In Word2Vec the idea of distributed word representations was applied, grouping semantically similar words in the vector space; which allows the learning algorithms to achieve better performance. In an attempt to fix minor issues in Word2Vec (i.e. accurate representations of rare words, adding subword information), Facebook AI Research team proposed FastText (Bojanowski et al., 2017), a continuation of Word2Vec, with character level n-grams, instead word n-grams.

Word2Vec and FastText yield embeddings for individual words/phrases, unlike the more complex architectures that can output vectors to represent sequences. For example long-short term memory (LSTM) networks are designed to handle long sequences as well as inputs and outputs of variable length sequences. LSTMs can handle the long-term dependency problem, that arises when the desired output depends on inputs that were given far in the past. ELMo (Sarzynska-Wawer et al., 2021) is an LSTM based architecture, that works by computing vectors on top of a two layer bidirectional language model (BiLM). Compared to previously discussed architectures, that return a single vector for a given word, based on the literal spelling of that word; ELMo produces context-sensitive embeddings.

Encoder-decoder architectures (Vaswani et al., 2017), that emerged later, are also known to handle variable length sequences while also adding positional information of tokens. These architectures also employ parallelization within training examples, achieving computational efficiency. These architectures also provide information about the token's context and use self-attention (or intra-attention), that relates different parts of an input, in order to compute a valid representation of the sequence. BERT (Devlin et al., 2018) is a transformer-based language model that employs the idea of self-attention, as a stack of encoders. BERT has roughly 110 million parameters which makes it a large model, rather challenging for fine-tuning or retraining for most applications. In order to adapt it to solve other problems, several variants of BERT emerged, each applying several changes to the original model (i.e. to reduce the model size, have fewer parameters, solve other tasks etc); these variants

.

are RoBERT (Liu et al., 2019), ALBERT (Lan et al., 2019), DistilBERT (Sanh et al., 2019), TinyBERT (Jiao et al., 2019) and Sentence-BERT (Reimers and Gurevych, 2019). Some of these variants allowed NLP practitioners to run the model on CPUs, or on the edge (i.e. mobile devices), while still consuming an adequate amount of computational resources.

All the above mentioned approaches are currently the best architectures at hand, trained on huge text corpora, providing good word vector representations for natural language understanding/processing. A common approach among NLP practitioners is to use the embeddings outputted by these pre-trained models, in solving downstream tasks. This method is convenient to use, but to achieve a boost in accuracy, another common practice is to fine-tune or retrain the model on a given dataset. However, fine-tuning the models is not an easy and computationally friendly task and problems such as data insufficiency or memory constraints could preclude the process. Such problems can be caused by large architectures, since some of the deep learning based models are more complex, with millions of parameters (i.e. BERT). Given the size of BERT and similar architectures, retraining them will require a large amount of resources that might not be available.Taking the computational constraints into account, many NLP practitioners decide to use the pre-trained models as a first step, often neglecting to resort to baselines. In our analysis, we used pre-trained word embeddings outputted by deep learning models, and also obtained results using some basic approaches like TF-IDF or CountVectorizer, then measured their performance on the given datasets. We found that most of the models yield comparable results, with TF-IDF and CountVectorizer performing slightly better, only under-performing RoBERTa. Thus we put forward the following question:
"Is the slight improvement in the metric of interest sufficient evidence to consider the given model as the best one?"

In academic papers improvement is based on point estimation, that mostly measures the average performance of the model. However, when using these models in

production settings, we are not only interested in the average value of the performance metric, but also in its variance. It is also vital for us to know whether the relative improvement of the metric of interest is significant or not, which brings us naturally to the task of estimating the distribution of the metric of interest.

Any given dataset can be considered a random sample from a union of a huge corpora of similar tasks, thus a metric derived for the sample can be a fair estimate for the population it is sampled from. One can determine the better model based on the models' respective metric of performance (i.e. Accuracy for binary classification). In order completely assess which model is better, one would need to understand the uncertainty around model predictions, and therefore metric of performance. Currently, two approaches exist - frequentist and Bayesian.

In the frequentist approach one assumes the parameters to be fixed and data to be random. In order to obtain uncertainty around predictions with the frequentist approach, one would need either explicitly assume distribution over the outcome variable or sample N amount of such datasets. However in practice it is not possible to do in most of the cases, and having $metric_1 > metric_2$, is sufficient to conclude that the first model is better.

Bayesian inference differs from the frequentist setting by expressing uncertainty through a probability distribution over the parameters (i.e. each model parameter is considered to be a random variable). Eventually one can obtain distributions of the performance metrics with their respective summary statistics and credible intervals. Therefore when comparing the models the information about the distributions should be taken into account.

Another important measure of model performance can be considered the inference time, since many practitioners might prefer a faster running algorithm over any slight improvement. To address that, we finish our experiments by measuring the inference time of the selected models, and compare them on that criteria as well.

.

2. Methodology

Depending on the given task, machine learning models offer several approaches of analyzing data and making decision based on it. Examples of such models are simple regression or classification methods, that can be used to obtain point estimates for the metric of interest. The most widely used point estimates are maximum likelihood (MLE) and maximum a posteriori (MAP) estimations. Mathematically speaking, these estimates are obtained from the Bayes' theorem (Bishop and Nasrabadi, 2006, Chapter 1.2.3), (Blum et al., 2020)

$$P(w \mid D) = \frac{P(D \mid w)P(w)}{P(D)} \tag{1}$$

The numerator of the Bayes' formula is composed of the prior distribution and the likelihood function. The prior distribution is our prior belief about the parameters, without yet observing the data, where w denotes the parameters; while the likelihood function $P(D \mid w)$ is the joint probability of the observed data, with a specified set of parameters w that maximize the function. To estimate MLE, one shall use the likelihood function $P(D \mid w)$ and choose the set of parameters that will maximize the probability of the observed data. That quantity expresses the probability of evaluating the observed data for different settings of the parameter vector w. All the quantities except the denominator are observed as functions of parameters w. The denominator $P(D)$ (total probability of D) is known as the normalization constant that can be expressed as an integral, in terms of prior distribution and the likelihood function. The normalization constant can be quite challenging to calculate, thus it is often skipped and the posterior distribution is approximated as follows (Bishop and Nasrabadi, 2006, Chapter 1).

$$posterior \propto likelihood \times prior \tag{2}$$

To derive the MAP estimation, one should maximize the following product $P(D \mid w)P(w)$, which is proportional to the posterior probability $P(w \mid D)$. The maximum of the derived distribution will be an estimate of the posterior's maximum, called MAP, which also corresponds to the mode of the posterior distribution. As discussed above, both MLE and MAP yield point estimates for the metric of interest, however, one might be interested in the model performance, given different settings for the parameter vector w. To obtain the desired information, the metric of interest should be represented in the form of a random variable instead of a point estimate. To achieve this, the problem should be approached from a Bayesian point of view. In the Bayesian approach there is just one fixed dataset, where the uncertainty in parameters w is expressed by a probability distribution over the parameter vector w, in the form of the posterior probability $P(w \mid D)$.

2.1 Sampling Methods

Bayesian statistics became widely used after the sampling methods were introduced, which are used to draw samples from a target distribution (in Bayesian computation it corresponds to the posterior distribution). In general, there are three main methods to obtain the posterior distribution, namely Expectation-Maximization algorithm (EM), Variational Inference and Monte Carlo Markov Chains (MCMC). EM and variational inference are not exact methods, since they use an approximation of the posterior distribution to update the model parameters. The MCMC method is exact, in that it allows sampling directly from the posterior distribution.

2.2 Monte Carlo Markov Chain (MCMC)

MCMC allows sampling from a large class of distributions, scaling well with the dimensionality of the sample space; unlike other sampling algorithms that can suffer

.

from limitations especially in spaces of high dimensionality (Bishop and Nasrabadi, 2006, Chapter 11), (Blum et al., 2020).

Before discussing MCMC in more detail, it is useful to highlight some properties of Markov chains in general. A Markov chain is a sequence of random variables $W_1, W_2, W_3...$, having a Markov property.

$$P(W_{i+1} = w_{i+1} \mid W_i = w_i, ..., W_0 = w_0) = P(W_{i+1} = w_{i+1} \mid W_i = w_i) \tag{3}$$

$$\forall i = 1, 2, 3...$$

Meaning that at any given time, the future state W_{i+1} depends on the current state W_i of the Markov chain, and not on the rest of the previous states. A state space S of the chain is the set of values the random variables W_i can have.

When sampling with MCMC, suppose we wish to sample from a distribution $p(z)$ that does not come from a family of known distributions, and that sampling directly from $p(z)$ is difficult. Furthermore suppose that we are easily able to evaluate $p(z)$ for any given value of z, up to some normalizing constant Z, so that

$$p(z) = \frac{1}{Z_p}\tilde{p}(z) \tag{4}$$

Because $p(z)$ is difficult to sample from, a proposal distribution $q(z)$ is introduced, which should be easier to draw samples from. The current state is $z(\tau)$, a record of which should be kept; and the proposal distribution $q(z \mid z(\tau))$ depends on the current state. Thus, the sequence of samples $z^{(1)}, z^{(2)}, \ldots$ forms a Markov Chain. From the formulation $p(z) = \dfrac{\tilde{p}(z)}{Z_p}$ we will assume that $\tilde{p}(z)$ can be easy to evaluate for a given value of z, although the value of Z_p may be unknown. The proposal distribution is also chosen to be straightforward to directly sample from. Thus, at each step we generate a candidate sample z^* from the proposal distribution $q(z)$ and then accept the sample according to an appropriate criterion.

If the candidate sample is accepted, then we say $z^{+1} = z^*$, if not, the candidate sample is discarded and the value for $z^{\tau+1}$ is set to z^τ, then another candidate point is sampled from the distribution $q(z \mid z^{\tau+1})$. It is important to note that for $\tau \to \infty$, the distribution $p(z^{(\tau)})$ converges to the required stationary distribution $p(z)$, irrespective of the choice of initially chosen distribution $p(z^{(0)})$. Thus, for simplicity we assume that as long as $q(z_A \mid z_B)$ is positive for any z_A and z_B (which is a sufficient but not necessary condition), the distribution of $z(\tau)$ will tend to $p(z)$ as $\tau \to \infty$. Hence $p(z)$ is the stationary distribution which in the case of Bayesian inference is considered to be the posterior distribution (Hyvönen and Tolonen, 2019, Chapter 4.3)

2.3 Predictive Distributions

Having obtained the posterior distributions of the parameters after observing the given dataset, one then obtains a posterior predictive distribution, which can reflect the expected behavior of new data. The posterior predictive distribution is the distribution of possible unobserved values conditional on the observed values.(Hyvönen and Tolonen, Nicenboim et al., Chapter 3.5).
The predictive distribution of a label y_{pred} is given by:

$$p(y_{pred} \mid y) = \int_w p(y_{pred} \mid w, y) p(w \mid y)\, dw \tag{5}$$

Because we have independent and identically distributed (i.i.d.) data, we assume that past and future observations are conditionally independent given w, i.e.,

$p(y_{pred} \mid w, y) = p(y_{pred} \mid w)$, the above equation can be written as

$$p(y_{pred} \mid y) = \int_w p(y_{pred} \mid w) p(w \mid y)\, dw. \tag{6}$$

.

Our approach is taking the linear combination of the unobserved, validation data with the parameters w of the posterior distribution and the intercept, and then applying a sigmoid function on top of it in the case of binary targets and softmax for multi-label data.

3. Dataset

The experiments were conducted on the IMDb Movie Reviews Sentiment Classification task (Maas et al., 2011), which is publicly available. This is a dataset for binary sentiment classification, with 50,000 highly polar movie reviews and 2 sentiments, positive ($class_1$) and negative ($class_0$).

4. Experiments

4.1 Embedding Tensors and TF-IDF/CountVectorizer Matrices

Following the problem formulation,the first step in the experiments was initializing the models we are interested in. First we obtained sentence embeddings for the BERT model and then spread to the other BERT variants (RoBERT, DistilBERT, TinyBERT, Sentence-BERT);for each of them using the pre-trained models available at Huggingface. Initially we extracted the pooled outputs, which did not yield satisfactory results; therefore we moved on to explore other approaches. A common practice was to average the token embeddings from the last hidden layer and obtain an array of size ($dataset\ length, \quad embedding\ size$).

Another widely used approach was to use the CLS tokens of the models (those are the first tokens of each sentence in the embeddings array), which carry information about the sentence. We concatenated the CLS tokens from 4 hidden layers, and later sampled data for the last 3, 2 and 1 hidden layers from the 4-layers concatenated array.

We noticed that BERT models yielded better results on the IMDb reviews dataset, when the output type was CLS tokens concatenation. However, the token vector averages from the last hidden state performed noticeably good.

An alternative approach was taking the token vector averages on each layer until the fourth hidden layer, and concatenating them. However, no noticeable boost in performance was observed when applying that method. For the sake of consistency, we decided to perform the experiments on the last hidden state token vector averages for both datasets.

Having the embeddings for BERT and the other models at hand, the next step was to extract Word2Vec and FastText word vectors, from their pre-trained models. Lastly, using TF-IDF and CountVectorizer from scikit-learn, we obtained the respective matrices, to complete the baseline models' list.

4.2 Logistic Regression

To obtain point estimations, we implemented logistic regression on top of the embeddings and TF-IDF CountVectorizer matrices. Our metric of interest was the accuracy score, measured for BERT and its variants using the pooler outputs, token vector averages, CLS concatenations from the last 4 to 1 layers; then for Word2Vec, FastText embeddings, and TF-IDF, CountVectorizer matrices. The resulted scores were then recorded (1, 2) for easy access and comparison.

4.3 Generating Probabilistic Results

After the point estimates for all models were available, we could get the probabilistic results for them. For BERT, RoBERT, DistilBERT, TinyBERT and Sentence-BERT we had several embeddings extracted from the models, so we chose the outputs that yielded the best results, which was the last hidden state token vectors averaging method.

.

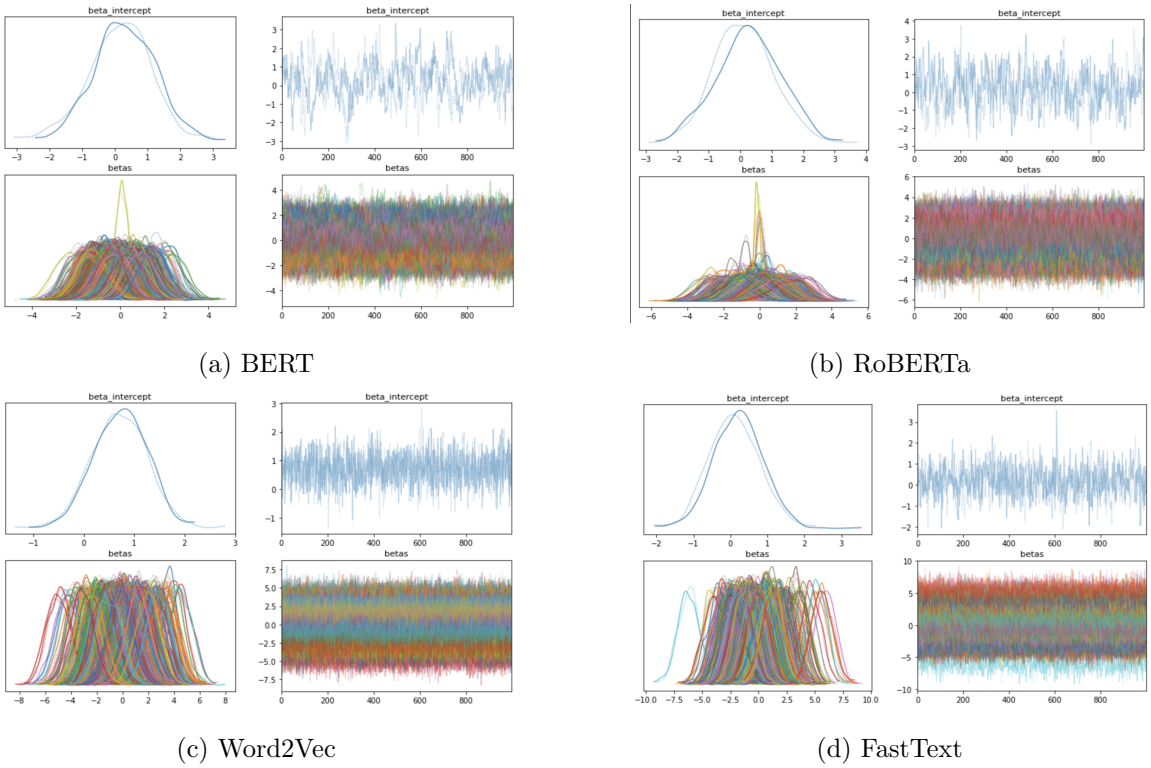
Before applying any probabilistic model, we needed to sample from the full dataset; to address the problem of data insufficiency as well, in which case fine-tuning or re-training might not be a possibility. Therefore we took 20% samples from each array of embeddings. To have a fair comparison, it was necessary to obtain point estimates for those samples.

In order to obtain probabilistic results we needed to construct a pipeline. The most widely used probabilistic programming libraries in Python are PyMC3 (Davidson-Pilon, 2015), Pyro (Bingham et al., 2019) and NumPyro (Phan et al., 2019), thus we carried out some experiments to choose one of the three. In the experimental part we applied all three models and measured the algorithm running time, results and other criteria. NumPyro and Pyro eventually beated PyMC3 in terms of running time and convenience. NumPyro is a NumPy backed library while Pyro is PyTorch backed. Because we implemented the code in PyTorch, Pyro was more convenient to work with, in order to be consistent with the rest of our experiments.

As Bayesian inference suggests, to begin, we needed to define priors for the intercept and coefficients. We made an assumption that the samples are expected to come from a normal distribution, with mean 0.5 and standard deviation of 1. Then the NUTS sampler (Hoffman et al.) was applied, using Hamiltonian MCMC (Betancourt). The sampler was applied with two parallel chains, to ensure the model convergence.

Figure 1 represents the traces of posterior distributions, after obtaining the posterior samples. The top plot depicts the ditributions of model intercepts with the bottom one showing how the model parameters are distributed.

Lastly we needed the distributions of the metric of interest for each model (i.e. accuracy). For the binary case we took the predictive distribution, which was the output of a sigmoid function over the linear combination of validation set and model parameters; defined a threshold of 0.5 and manually transformed the probabilities to be either 0 or 1. With the accuracy score method from sklearn, the accuracy



(a) BERT

(b) RoBERTa

(c) Word2Vec

(d) FastText

Figure 1: Posterior Distributions (Top subplot: intercept, Bottom subplot: parameters)

score for each column vector was calculated and combined in a list to represent the accuracy distribution of the model. For each accuracy distribution the kernel density estimation is plotted, to visualize how it is distributed. Then from each distribution we randomly sample 10,000 data points with replacement; to later compare them in a pairwise manner.

4.4 Highest Density Interval

Figure 2 represents the distributions of accuracy scores; and we can identify the pairs that do not have significant difference. So we propose a hypothesis about the significance of that difference and test it using the highest density interval values. Then the top performing models were chosen, to make inference based on them. From the Kernel Density plots it is worth noticing that the better performing models

.

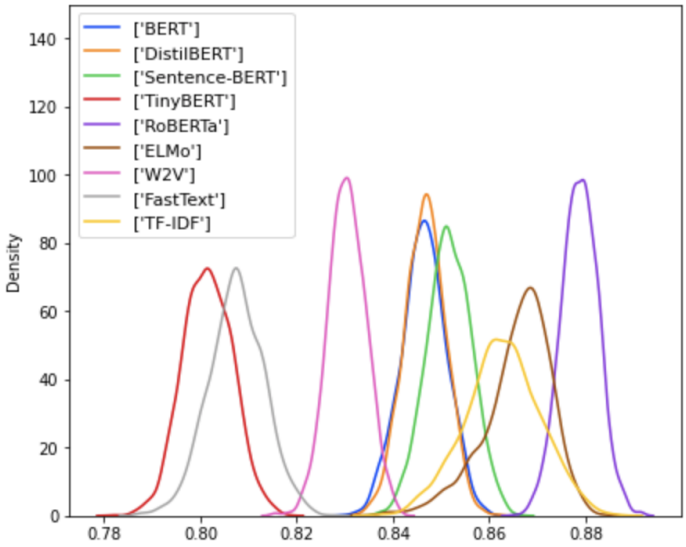


Figure 2: Kernel Density Estimation plots of Accuracy Distributions fro each Model on IMDb Movie Sentiment Classification task

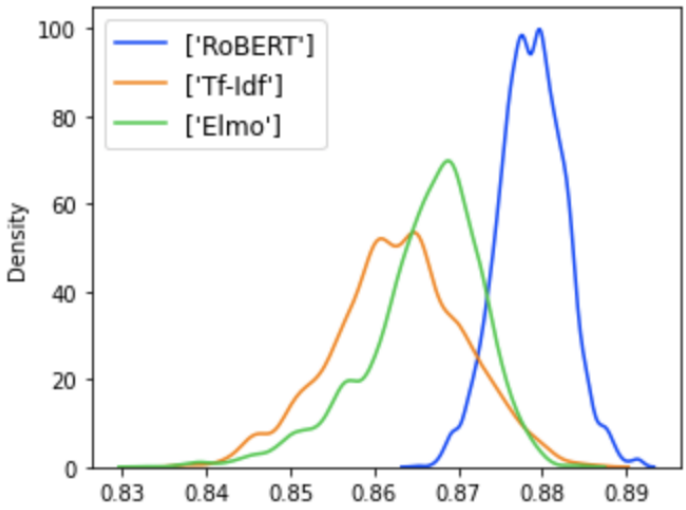
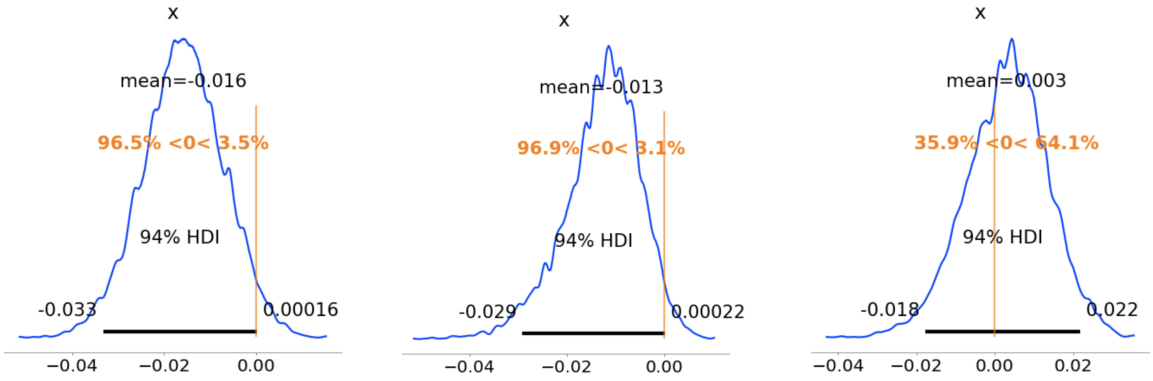


Figure 3: Accuracy Distributions of the 3 Best Performing Models



(a) RoBERTa vs TF-IDF (b) RoBERTa vs ELMo (c) ELMo vs TF-IDF

Figure 4: Highest Density Intervals

are RoBERTa, ELMo and TF-IDF; all three having an average value within the range 87-89%. In Figure 3 we can see the distributions of accuracies of the chosen 3 models.

Our decision is based on the highest density interval for posterior distribution (HDI/HPD); which summarizes the range of most credible values of the measurement. The credible interval is a way to quantify the uncertainty of the parameters, in other words, it is a statement about the probability of the parameter (Kruschke, 2018), (Hyvönen and Tolonen, 2019, Chapter 3,1). In our experiment we take 94% density interval, which gives us the region with the 94% most credible values of the parameter. While generating the intervals, we fix the reference value at 0, to understand the significance of the difference between two distributions (4).

5. Results

In this section we explore the results yielded by all models on the IMDb Movie Reviews dataset. As discussed in 4.1 and in 4.3, we were interested in the results with the highest accuracy score, which in our setting was the token vectors averaging method. Then, to address the problem of data insufficiency we sample from each output matrix and perform the probabilistic experiments on those samples.

.

5.1 IMDb Movie Movie Reviews Sentiment Classification task

For the IMDb Reviews dataset first we performed the experiments on the full data and recorded the results (1), then randomly sampled 20% from the dataset and performed the same experiments on the sample. From the Table 1 and Table 2 we see that the accuracy scores are generally comparable, all ranging between 82-88%.

The samples were then used to perform MCMC sampling and obtain a posterior predictive distribution. We converted the values of predictive distribution to be in [0,1] as represented in section 4.3. The column-wise accuracy distributions for each dataset were obtained and plotted in Figure 2. From the plot we can identify the comparable distributions and test the proposed hypothesis on them, to see if they are significantly different.

As stated in section 4.3, in Figure 3 we plotted the accuracy distributions of the 3 top performing models. Then we performed a pairwise subtraction among the 3 distributions, and measured the HDI values for each difference-distribution, as shown in Figure 4.

All 3 plots are constructed on 94% HDI intervals and the reference value is fixed at 0. All 3 plots suggest that the difference in the observed models is not significant. For example, in case if RoBERTa and TF-IDF 4a, 96.5% of the values are below 0 and 3.5% are above 0. It is worth noticing that the reference value 0 is contained within that interval, suggesting that the difference between the observed models is non-significant. Similarly, we come to the same conclusion for RoBERTa vs ELMo and ELMo vs TF-IDF.

Logistic Regression Accuracy Score on Validation sets		
Output Type	Full Dataset	20% Random Sample
BERT Last Hidden State (without CLS, word tokens averaged)	≈ 87.6%	≈ 85.5%
BERT Last 4 Hidden Layers (CLS concatenated)	≈ 84.1%	≈ 80.5%
BERT Last 3 Hidden Layers (CLS concatenated)	≈ 83.9%	≈ 79.7%
BERT Last 2 Hidden Layers (CLS concatenated)	≈ 82.2%	≈ 78.9%
BERT Last Hidden Layer (CLS concatenated)	≈ 77.8%	≈ 75.8%

(a) Only token averaging method is chosen

Logistic Regression Accuracy Score on Validation sets		
Token Vectors Averaged	Full Dataset	20% Random Sample
RoBERTa Last Hidden State	≈ 89.4%	≈ 88.6%
DistilBERT Last Hidden State	≈ 86.6%	≈ 85.3%
TinyBERT	≈ 83.1	≈ 81.9
Sentence-BERT Last Hidden State	≈ 87.9%	≈ 86.3%

Table 1: Point Estimates (The highlighted values are used in further experiments)

.

Logistic Regression Accuracy Score on Validation sets		
Output Type	Full Dataset	20% Random Sample
Word2Vec Embeddings	≈ 85.9%	≈ 83.3%
FastText Embeddings	≈ 84.8%	≈ 81.1%
ELMo Embeddings	≈ 88.1%	≈ 87.3%
TF-IDF	≈ 90.2%	≈ 87.3%
Count Vectorizer	≈ 89.2%	≈ 87.3%
TF-IDF (bigram)	≈ 89.5%	≈ 84.8%
Count Vectorizer (bigram)	≈ 89.7%	≈ 86.3%

Table 2: Point Estimates (The highlighted values are used in further experiments)

5.2 Inference Time of the Models

As there was no significant difference in the accuracy distributions of the observed models, we suggest approaching to the problem from a new angle.

Inference Time of the Models			
Model	Time on 1000 data-points	Time on 1 Datapoint	Device type
ELMo	7.5 minutes	0.45 seconds	CPU
RoBERTa	18.5 minutes	1,11 seconds	CPU (0.05 on GPU)
TF-IDF	10 seconds	0.01 seconds	CPU

Table 3: Inference Time of the Best Performing models

Starting with a list of complex and baseline models, as a result of our experiments the list was narrowed down to only 3 models, RoBERTa, a transofrmer based architecture, ELMo, an LSTM based model and TF-IDF, a baseline model based on word frequency. It is still not straightforward to choose the better performing model,

even if the choice is between 3 models, thus we suggest looking at the problem from another dimension. If the practitioner does not have enough time to construct a complex model, and quick results are needed, from the 3 better performing models one can choose the one with the fastest running time. Thus we measured the inference time for each model and evaluated the time each model requires to perform a single iteration. Table 3 summerizes the inference time of each model, ran on CPU, and GPU for RoBERTa. The results show that TF-IDF outperforms RoBERTa and ELMo in terms of inference time by an order of magnitude. We also notice that when executed on GPU RoBERTa shows comparable speed. However this suggests, that in order to bring such a model to production more considerations should be taken into account.

6. Conclusion

In this paper we have experimented with a different approach of selecting a best model based on the model performance metric and execution speed. By leveraging Bayesian techniques we obtained a full distribution over the accuracy metric of a binary classifier and have shown that deep learning approaches, while showing slightly higher performance based on the point estimates, show no significant improvement over less sophisticated methods. We also have shown that implementing simple TF-IDF based model leads to faster execution time on a less demanding production environment. The deep learning based methods require much more considerations with that respect. This method can be extended to multi-class classification (implementation is provided in the github repository) and to other parametric models.

While MCMC methods are less demanding with respect to computational resources, we have to point out that training such a model might require multiple hours on a moderate to big datasets.

.

References

Michael Betancourt. A conceptual introduction to hamiltonian monte carlo. *arXiv preprint arXiv:1701.02434*, 2017.

Eli Bingham, Jonathan P Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D Goodman. Pyro: Deep universal probabilistic programming. *The Journal of Machine Learning Research*, 20(1):973–978, 2019.

Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

Avrim Blum, John Hopcroft, and Ravindran Kannan. *Foundations of data science*. Cambridge University Press, 2020.

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.

Cameron Davidson-Pilon. *Bayesian methods for hackers: probabilistic programming and Bayesian inference*. Addison-Wesley Professional, 2015.

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.

Matthew D Hoffman, Andrew Gelman, et al. The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *J. Mach. Learn. Res.*, 15(1): 1593–1623, 2014.

Ville Hyvönen and Topias Tolonen. Bayesian inference 2019. 2019.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*, 2019.

John K Kruschke. Rejecting or accepting parameter values in bayesian estimation. *Advances in methods and practices in psychological science*, 1(2):270–280, 2018.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P11-1015`.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013.

Bruno Nicenboim, DJ Schad, and Shravan Vasishth. An introduction to bayesian data analysis for cognitive science, 2021.

Du Phan, Neeraj Pradhan, and Martin Jankowiak. Composable effects for flexible and accelerated probabilistic programming in numpyro. *arXiv preprint arXiv:1912.11554*, 2019.

.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

Justyna Sarzynska-Wawer, Aleksander Wawer, Aleksandra Pawlak, Julia Szymanowska, Izabela Stefaniak, Michal Jarkiewicz, and Lukasz Okruszek. Detecting formal thought disorder by deep contextualized word representations. *Psychiatry Research*, 304:114135, 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.