**1 INTRODUCTION**

Text classification is an important problem in Natural Language Processing (NLP). Real world use cases include spam filtering or e-mail categorization. It is a core component in more complex systems such as search and ranking. Recently, deep learning techniques based on neural networks have achieved state of the art results in various NLP applications. One of the main successes of deep learning is due to the effectiveness of recurrent networks for language modelling and their application to *speech recognition* and *machine translation* (*Mikolov, 2012*). However, in other cases including several text classification problems, it has been shown that deep networks do not convincingly beat the prior state of the art techniques (*Wang & Manning, 2012; Joulin et al., 2016*).

In spite of being (typically) orders of magnitude slower to train than traditional techniques based on n-grams, neural networks are often regarded as a promising alternative due to compact model sizes, in particular for character based models. This is important for applications that need to run on systems with limited memory such as smartphones. This paper specifically addresses the compromise between classification accuracy and the model size. We extend our previous work implemented in the fastText library 1. It is based on n-gram features, ***dimensionality reduction***, and a fast approximation of the softmax classifier (*Joulin et al., 2016*). We show that a few key ingredients, namely feature pruning, quantization, hashing, and retraining, allow us to produce text classification models with tiny size, often less than 100kB when trained on several popular datasets, without noticeably sacrificing accuracy or speed.

We plan to publish the code and scripts required to reproduce our results as an extension of the fastText library, thereby providing strong reproducible baselines for text classifiers that optimize the compromise between the model size and accuracy. We hope that this will help the engineering community to improve existing applications by using more efficient models.

This paper is organized as follows. *Section 2* introduces related work, *Section 3* describes our text classification model and explains how we drastically reduce the model size. *Section 4* shows the effectiveness of our approach in experiments on multiple text classification benchmarks.

**2 RELATED WORK**

**Models for text classification.** Text classification is a problem that has its roots in many applications such as web search, information retrieval and document classification (*Deerwester et al., 1990; Pang & Lee, 2008*). Linear classifiers often obtain state-of-the-art performance while being scalable (*Agarwal et al., 2014; Joachims, 1998; Joulin et al., 2016; McCallum & Nigam, 1998*). They are particularly interesting when associated with the right features (*Wang & Manning, 2012*). They usually require storing embeddings for words and n-grams, which makes them memory inefficient.

**Compression of language models.** Our work is related to compression of statistical language models. Classical approaches include feature pruning based on entropy (*Stolcke, 2000*) and quantization. Pruning aims to keep only the most important n-grams in the model, leaving out those with probability lower than a specified threshold. Further, the individual n-grams can be compressed by quantizing the probability value, and by storing the n-gram itself more efficiently than as a sequence of characters. Various strategies have been developed, for example using tree structures or hash functions, and are discussed in (*Talbot & Brants, 2008*).

**Compression for *similarity estimation* and *search*.** There is a large body of literature on how to compress a set of vectors into compact codes, such that the comparison of two codes approximates a target similarity in the original space. The typical use-case of these methods considers an indexed dataset of compressed vectors, and a query for which we want to find the nearest neighbors in the indexed set. One of the most popular is *Locality-sensitive hashing (LSH)* by Charikar (*2002*), which is a binarization technique based on random projections that approximates the cosine similarity between two vectors through a monotonous function of the Hamming distance between the two corresponding binary codes. In our paper, LSH refers to this binarization strategy 2.

Many subsequent works have improved this initial binarization technique, such as spectal hashing (*Weiss et al., 2009*), or Iterative Quantization (ITQ) (*Gong & Lazebnik, 2011*), which learns a rotation matrix minimizing the quantization loss of the binarization. We refer the reader to two recent surveys by Wang et al. (*2014*) and Wang et al. (*2015*) for an overview of the binary hashing literature. Beyond these binarization strategies, more general quantization techniques derived from Jegou et al. (*2011*) offer better trade-offs between memory and the approximation of a distance estimator. The Product Quantization (PQ) method approximates the distances by calculating, in the compressed domain, the distance between their quantized approximations.

This method is statistically guaranteed to preserve the Euclidean distance between the vectors within an error bound directly related to the quantization error. The original PQ has been concurrently improved by Ge et al. (*2013*) and Norouzi & Fleet (*2013*), who learn an orthogonal transform minimizing the overall quantization loss. In our paper, we will consider the Optimized Product Quantization (OPQ) variant (*Ge et al., 2013*).

***Softmax* approximation.** The aforementioned works approximate either the Euclidean distance or the cosine similarity (both being equivalent in the case of unit-norm vectors). However, in the context of fastText, we are specifically interested in approximating the maximum inner product involved in a *softmax* layer. Several approaches derived from LSH have been recently proposed to achieve this goal, such as Asymmetric LSH by Shrivastava & Li (*2014*), subsequently discussed by Neyshabur & Srebro (*2015*). In our work, since we are not constrained to purely binary codes, we resort a more traditional encoding by employing a magnitude/direction parametrization of our vectors. Therefore we only need to encode/compress an unitary d-dimensional vector, which fits the aforementioned LSH and PQ methods well.

***Neural* network compression models**. Recently, several research efforts have been conducted to compress the parameters of architectures involved in computer vision, namely for state-of-theart Convolutional Neural Networks (CNNs) (*Han et al., 2016; Lin et al., 2015*). Some use vector quantization (*Gong et al., 2014*) while others binarize the network (*Courbariaux et al., 2016*). Denil et al. (*2013*) show that such classification models are easily compressed because they are overparametrized, which concurs with early observations by LeCun et al. (*1990*).