Exploring the Classification of Security Events using Sparse and Dense Representation of Text

Eduan Kotzé
Department of Computer Science and Informatics
University of the Free State
Bloemfontein, South Africa
kotzeje@ufs.ac.za

Burgert Senekal
Department of Computer Science and Informatics
University of the Free State
Bloemfontein, South Africa
burgertsenekal@yahoo.co.uk

Walter Daelemans
Center for Computational Linguistics and Psycholinguistics (CLiPS)
University of Antwerp
Antwerp, Belgium
walter.daelemans@uantwerpen.be

***Abstract*—With the availability of open source intelligence systems, such as the Integrated Crisis Early Warning System (ICEWS), news stories are constantly being monitored and analyzed. However, with the increasing amount of data available on the Internet, automated methods such as machine learning can be used to quantify and speedup analysis. Although projects such as ICEWS have a global focus, it is important that in South Africa, we focus on the local situation. Using WhatsApp as our new source, we evaluated a machine learning approach to automatically classify violent events, such as protests of mass violence, by using dense vector representations. Our experimental results indicated that dense vector representation did not necessarily improve the performance of the machine learning classifier.**

***Keywords*—text classification, word2vec, doc2vec, dense vectors, machine learning, WhatsApp**

I. Introduction

Text mining has become important in extracting meaning from user-generated data (UGD) as the amount and velocity with which text is produced increase. Additionally, new information channels such as social media have become more relevant in terms of the dissemination of information. WhatsApp's popularity and ease of use in South Africa made it an indispensable platform for disseminating emergency news, and a variety of institutions, from neighborhood watches to non-governmental organizations, are currently using the platform. Automated classification using methods such as machine learning has become important for extracting useful information from large amounts of text. We present a classification method for machine learning to identify WhatsApp messages related to security issues in a South African context. The remainder of the paper is structured as follows. A brief overview of related work is presented in Section II. Section III presents an overview of concepts related to our study and discusses the methodology. Section IV provides the experimental results, and Section V concludes the study and presents directions for further research.

II. Related work

Extensive research has been conducted on the automatic classification of text in the domain of natural language processing (NLP) [1]. However, in the domain of security and defense, more work is required to automatically classify events relating to security matters. Automated and semi-automated event coder projects are of particular relevance to this study. Automated event coders code natural language text (usually from new sources) as categories of events, for example, they would extract "engage in violent protest" from the sentence "Zimbabwean riots continue in Harare". Examples of such projects include the Integrated Crisis Early Warning System (ICEWS), Global Data on Events Language and Tone (GDELT), and Social, Political, and Economic Event Database (SPEED). GDELT and ICEWS are fully automated global event coder systems, while SPEED is a semi-automated system [2]. Other similar projects include the Armed Conflict Location and Event Data (ACLED) project [3], the CIA's Political Instability Task Force (PITF) [4], the Social Conflict Analysis Database (SCAD) [5], and the Uppsala Conflict Data Program (UCDP) [6].

III. Methodology

Text classification is traditionally an NLP task. In this section, we briefly describe the fundamental concepts related to creating text features and text classification. We will present background knowledge on word and document embeddings, followed by the data collection and annotation process, feature engineering, and finally, model selection and tuning.

A. Word Embedding

Text data is known for its high dimensionality of text features [7]. To overcome this shortcoming, feature extraction and feature selection are necessary before any text classification can be performed. Popular feature extraction techniques include *bag-of-words*, *bag-of-n-gram* (n-gram) and *Term Frequency-Inverse Document Frequency (TF-IDF)*. However, these models have several shortcomings, such as data sparsity and high dimensionality [8].

A recent advance in the domain of NLP is the use of word embedding, which is a dense representation of text using a feed-forward neural network. A language modeling technique is employed to encode every word into a continuous higher dimensional vector space. A popular language model to produce such word embeddings is Word2Vec, which was created by a team of Google researchers led by Tomáš Mikolov [9]. Word2Vec is a group of two-layer neural networks models that embeds words in a lower-dimensional vector space. The

978-1-7281-4162-6/20/$31.00 ©2020 IEEE

algorithm trains word representations from a large scale corpus using either a continuous-bag-of-words (CBOW) or a skip-gram (SG) model [9]. In both these models, techniques are used to learn word weights that act as word vector representations. The CBOW model predicts a central target word from a window of surrounding words. We refer to these surrounding words, where each word is represented as a feature vector, as context words. After training the CBOW model, these vectors become the word vectors. The skip-gram model does not feed previous words into the model, but predicts the surrounding ("context") words, given the current word. In other words, instead of using the context words to predict the center word (as with CBOW), the model predicts context words from the center target word.

B. Document Embedding

More recently, Mikolov and his colleagues introduced Doc2Vec as an extension of Word2Vec [10]. Doc2Vec (*aka Paragraph Vector or Sentence Embedding*) is an unsupervised algorithm that learns fixed-length feature representations from large blocks of texts. The texts can be of variable length, ranging from sentences to documents. The algorithm represents each document by a dense vector that is trained to predict words in the document. The algorithm also takes into consideration the word order within a narrow context, similar to an *n*-gram model. The combined result has the potential to overcome the weakness of bag-of-words and/or bag-of-*n*-grams models because its generalization is better, and it creates a low-dimension representation while still having a fixed length [10]. Two approaches are available to learn paragraph and document representations, namely the distributed memory (PV-DM) model and the distributed bag-of-words model (PV-DBOW).

The *Distributed Memory Model of Paragraph Vectors* (PV-DM) approach is similar to the CBOW approach in Word2Vec. In the Word2Vec CBOW model, the model learns to predict a center word based on the context of the words. Similarly, PD-DM predicts a center word from a paragraph. For example, given the sentence "The cat sat on sofa", the PV-DM classifier takes a paragraph matrix, which is a matrix where each column represents the vector of a paragraph. These paragraph and word vectors can either be averaged or concatenated. The classifier then uses this hidden layer vector as input to predict the center word ("on"). In the model discussed below, the word vectors represent the concept of a word, while the paragraph (document) vectors intend to represent the concept of a document.

The *Distributed Bag of Words Model of Paragraph Vectors* (PV-DBOW) is slightly different from the PV-DM model described earlier. This model ignores the context words in the input, and forces the prediction of words randomly sampled from the paragraph in the output. In other words, instead of predicting the next word, the PV-DBOW model uses a paragraph vector to classify entire words in the document. For example, given the same sentence as earlier "The cat sat on sofa", a paragraph vector is trained to predict the words in a small window. This approach is similar to the skip-gram approach in Word2Vec. According to Le and Mikolov [10], this algorithm is considered faster (as opposed to Word2Vec) and consumes less energy as the word vectors need not to be saved. The authors also recommended a combination of both PV-DBOW and PV-DM, although it was argued that the PV-DM model could achieve good results on its own.

C. Data Collection and Annotation

The University of the Free State granted ethical clearance and permission for the authors to be added to WhatsApp groups that share information on violent incidents for this project (UFS-HSD2019/0175). Between 30 May 2018 and 18 February 2019, we collected and analyzed data from 15 different WhatsApp groups that share information on protests and mass violence. These groups are closed groups and do not share information about other forms of crime [11]. The groups cannot be identified in the paper for ethical reasons. The pre-processing of the WhatsApp messages and the annotation process are described in [11]. The final tokenized and pre-processed annotated dataset consisted of 7,889 different messages. This dataset was split into 80/20 training (n=6,311) and testing (n=1,578) datasets. The detailed quantity and related context information are shown in Table I.

TABLE I. DATASET ATTRIBUTE

	Labeled Message	***Train***	***Test***
Land Grab	50	38	12
Farm Attack	271	223	48
Crime	431	352	79
Protest	2949	2363	586
Safe	4188	3335	853
Total	**7889**	**6311**	**1578**

As shown in Table I, the experimental corpus had a strong class imbalance distribution where the vast majority of posts were not events (i.e. ' safe '). In machine learning classification, class imbalance can lead to decreased performance and accuracy [13]. We were also concerned that the machine learning algorithm would be skewed towards the majority class ("safe"), and that the minority classes ("Land Grab" and "Farm Attack") would be viewed as outliers and therefore be ignored. We applied the Python package imbalanced-learn [14] as data resampling to counter this as part of the classification pipeline. Due to a relatively small number of instances, we opted for random over-sampling, which produced new samples in minority classes by randomly selecting samples with replacement.

D. Feature Engineering

In this study, we used embedding features (Word2Vec and Doc2Vec) as well as lexical and TF-IDF features. Since two languages (English and Afrikaans) are involved, and a WhatsApp message often contained mixed-language, feature vectorization created a bilingual representation of the text, and

did not make use of an automatic language identification in the pipeline.

Words and characters from the training corpus formed an n-grams set of lexical features. We extracted word unigrams and bigrams (w1, w2) as well as character four-grams (ch4) as binary, spare features. The n-grams were created from raw words rather than lemmas to train morphological information. Preliminary experiments on our dataset indicated that unigrams would produce better results and we discarded the bigram features. The set of lexical features were transformed into TF-IDF features using scikit-learn, a machine learning library for Python [15]. TF-IDF is the composite weight of two statistics, *term frequency* (TF) and *inverse document frequency* (IDF), where weights are created instead of frequency counts. The reason we opted for TF-IDF instead of frequency counts (bag-of-words) is because TF-IDF also considers the IDF of each term when performing raw term frequency computations, often producing better results [16]. Default settings were used for min_df, max_df, smooth_tf, as well as maximum number of features. Inverse-document-frequency reweighting was enabled (use_idf=True) and to reduce document length bias, we set norm to 'L2'. The final vector space resulted in a total of 15,110 unigram word and 48,990 character four-gram tokens.

For Word2Vec, we adopted two approaches when creating the word embedding features. In the first approach, each WhatsApp message was represented by the average of the word embedding vectors of the words that composed the message. The second approach also averaged the word embedding vectors, but each embedding vector was now weighted (multiplied) by the TF-IDF. We ran the Word2Vec algorithm on the training corpus applying both the skip-gram (SG) and continuous bag-of-words (CBOW) models. A minimum word count of 1, a context window size of 6, and a word vector dimensionality of 500 features were used for each model. Since messages can each have a different number of vectors depending on the number of words it contains, a function is required to perform document-level task. An *arithmetic mean* function was used to unify all word vectors and create a single vector representing the entire document [17]. This function also checked whether the words in a message occurred in the vocabulary of both Word2Vec models. If the word was found, the average of the word vectors was returned, otherwise 0.0 was returned. Both CBOW and skip-gram models produced a vocabulary of 16,929 words from the training corpus.

For TF-IDF weighted sum of embedding vectors, we drew inspiration from the work of Lilleberg et al [18]. Their results showed that the combination of Word2Vec weighted by TF-IDF could outperform either of them individually and achieve higher accuracy. For the purpose of this paper, we constructed a Word2Vec weighted by TF-IDF without stop words vector space. Since the term frequency (TF) weight was already considered when we constructed a single vector representing the entire document, the TF weight was now omitted. Instead, only the Inverse Document Frequency (IDF) weight was used when building the TF-IDF model to compute each word's IDF as its weight. A similar approach has been used in Zhao et al [19] and Corrêa et al [20].

E. Model Selection and Training

This study used the Gensim [21] Python implementation of Doc2Vec. Several Doc2Vec algorithms were also used to produce paragraph models for our experiment. The models were trained on the entire dataset, reason being that the algorithm training is completely unsupervised and therefore there is no need to hold out any data, as it is unlabeled [22]. Algorithms included Distributed Bag of Words (DBOW), Distributed Memory with default averaging of the context word vectors (DM/M), and Distributed Memory with concatenation (DM/C). Each paragraph model was trained with a particular algorithm using an explicit multiple-pass (epochs=30) and alpha-reduction (alpha=-0.002) approach with added shuffling. Model selection was done using 10-fold cross validation in an exhaustive grid search over all possible hyperparameter configurations. The hyperparameters that we were interested in included *window* (maximum distance between the current and predicted word in a sentence), *size* (dimensionality of the feature vectors), *min_count* (ignores all words with total frequency lower than this) and negative (negative sampling specifies how many "noise words" should be drawn).

For text classification, we used a Logistic Regression (LogReg) classifier (*aka logit, MaxEnt*) from scikit-learn [15], which is a popular classification model used in machine learning for binary or multiclass classification. Logistic regression models performed well when used with high dimensional and sparse textual data [23]. We employed a multiclass LogReg classifier, where a document is assigned to one class among several possible classes. The algorithm used in the optimization problem (solver) was set to 'newton-cg' since it was a multiclass classification problem. Additionally, multiclass was set to 'multinomial', meaning the loss minimized is the multinomial loss fit across the entire probability distribution. Since the 'newton-cg' solver only supports L2 regularization with primal formulation, the value of the penalization parameter was set to the default value ('L2').

As with Doc2Vec, model selection was done using 10-fold cross validation in an exhaustive grid search over all possible hyperparameter configurations. The hyperparameters that we were interested in included *C* (inverse of regularization strength) and *tol* (tolerance for stopping criteria). D2VTransformer was used for model selection when tuning the Doc2Vec and LogReg hyperparameters. D2VTransformer is a scikit-learn wrapper for paragraph2vec models and facilitates grid-searching using Gensim [21] along with scikit-learn [15].

Table II shows the hyperparameters combinations for Doc2Vec and LogReg model selection.

TABLE II. HYPERPARAMETERS IN GRID SEARCH MODEL SELECTION

Model	Hyperparameter	Values	Optimal Value
doc2vec	window	[5, 10, 15]	5
doc2vec	min_count	[1 ,2]	1

doc2vec	size	[200, 300, 400, 500]	500
doc2vec	negative	[5, 10, 15]	15
LogReg	C	$1e^{\{-3,-2,-1,0,1,2\}}$	100
LogReg	tol	$1e^{\{-3,-2,-1,0,1\}}$	0.1

Le and Mikolov [10] noted that combining a paragraph vector from Distributed Bag of Words (DBOW) and Distributed Memory (DM), improves classification performance. We adopted a similar strategy of pairing models for evaluation, as was demonstrated in the Gensim Doc2Vec tutorial on the IMDB sentiment data set [24]. We concatenated the paragraph vectors obtained from each model with the help of a thin wrapper class (ConcatenatedDoc2Vec) included in a Gensim [21] test module. Note that this is the concatenation of output-vectors and differs from the input-window-concatenation of DM/C enabled by the setting *dm_concat*=1 during Doc2Vec model training.

F. Evaluation Metrics

Precision, recall, F1-score, and accuracy are evaluation metrics used to assess the performance of machine learning algorithms in binary text classification problems. Since we were using a multi-class classifier, we had to use measures based on the generalization of the binary metrics. This can be accomplished using either macro-averaging or micro-averaging. For the purpose of this study, we implemented a multi-class classifier and decided to use macro-averaging that calculates the mean of the binary metrics. Precision$_M$ is therefore defined as the average per-class agreement of the data class labels with those of the classifier. Recall$_M$ (or Sensitivity) is the average per-class effectiveness of a classifier to identify class labels. F-score$_M$ is the relationship between the sample's positive labels and those predicted by the classifier on a per-class average [25].

IV. Experiment and Analysis

A. Experimental Setting

The following section discusses the performance of the different embedding models (Word2Vec, Doc2Vec) as well as the LogReg classifier on the training data set, obtained in 10-fold cross validation, and on the test data set. Because we were using model_selection.cross_validate, we implicitly used a Stratified K-Folds cross-validator object. This cross-validation object is a variation of KFold and returns stratified folds. These folds are made by preserving the percentage of samples for each class. The accuracy performance evaluation parameter was calculated and used during cross-validation to determine the best performance. We report the accuracy, macro-averaged precision, recall and F-scores for the five event classes. As mentioned earlier, this experimental dataset was split into 80% training and 20% testing datasets.

B. Results and analysis

The experimental results on the test data set are shown in Table III.

TABLE III. Results on the test data set for doc2vec

Model	Unigram			
	Acc	***R***	***P***	***F1***
DBOW	0.771	**0.664**	0.561	0.598
DM/M	0.673	0.605	0.468	0.508
DM/C	0.533	0.584	0.382	0.421
DBOW & DM/M	0.770	0.597	0.556	0.573
DBOW & DM/C	**0.773**	0.661	**0.570**	**0.604**
DM/M DM/C	0.676	0.616	0.480	0.521

*boldface indicates the best evaluation score in each column

In case of unigram mode features, concatenating document vectors in different combinations boosted model performance. The best training accuracy for the logistic regression classifier was from DBOW at 0.771 (std. dev. 0.03) with a training F-score of 0.578 (std. dev. 0.08). With concatenated vectors, the highest training score was 0.775 (std. dev. ±0.02) from the DBOW+DM/C model. The best test accuracy score for single model was again from DBOW at 0.771 with a training F-score of 0.598. Similarly, with concatenated vectors, the high test score of 0.773 and test F-score of 0.604 were from the DBOW+DM/C model. In addition to concatenating document vectors, we also experimented with phrase (collocation) detection, which is part of the Gensim library [21]. Phrase detection automatically identifies or detects common phrases (bigrams, trigrams, four-grams, etc.) from a stream of sentences. This work was introduced by Mikolov and colleagues [9], who presented a simple method for identifying phrases in text, and then created a learning vector representation for phrases. We conducted tests using phrase (collocation) detection to create bigram and trigram Doc2Vec models. Only the test scores of the models are reported in Table IV.

TABLE IV. Test set comparison of different doc2vec models

Model	Best model						Best test result F1
	Unigram		***Bigram***		***Trigram***		
	Acc	***F1***	***Acc***	***F1***	***Acc***	***F1***	
DBOW	0.771	**0.598**	0.764	0.602	0.760	**0.623**	0.623 (trigram)
DM/M	0.673	0.508	0.691	0.500	0.674	0.521	0.521 (trigram)
DM/C	0.533	0.421	0.523	0.411	0.490	0.369	0.421 (unigram)
DBOW & DM/M	0.770	0.573	0.761	0.573	0.772	0.616	0.616 (trigram)
DBOW & DM/C	0.773	**0.604**	0.767	0.604	0.758	**0.621**	0.621 (trigram)
DM/M DM/C	0.676	0.521	0.693	0.497	0.683	0.528	0.528 (trigram)

The best performance of the logistic regression classifier with a single model was DBOW (trigram) with a test accuracy score of 0.623, while the best concatenated vectors model was the DBOW+DM/C (trigram) model with a test accuracy score of 0.621.

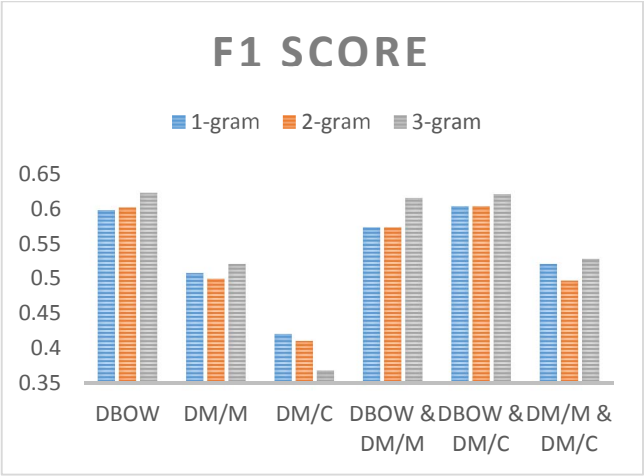

Fig. 1. Test F1 scores of different Doc2Vec models

From the test results (Figure 1), it is clear that the single models DBOW and DM/M tend to perform better with increased n-gram, while DM/C tends to perform worse with increase n-gram. All the concatenated models tend to perform better with increased n-gram, which was to be expected. We also created concatenated document vectors from different n-grams (unigrams, bigrams and trigrams). We used vectors from the *unigram DBOW* model and joined this with the *trigram DM/M* model. This concatenated document vector space was then used with unigram word TF-IDF vectors (w1) as well as four-gram character TF-IDF vectors (ch4). For Word2Vec models, each WhatsApp message was represented by the average of the word embedding vectors. Thereafter, each embedding vector was weighted (multiplied) by the TF-IDF factor. We also concatenated vectors of the CBOW and skip-gram (SG) models into a 1000 dimension vector space. The models were trained and tested using 10-fold cross validation. We will report the training and testing score of all the models.

TABLE V. TRAIN SET COMPARISON OF DIFFERENT MODELS

Model		Concatenated Models (Training)			
		Acc	*R*	*P*	*F1*
Word2Vec	W2V (CBOW & SG)	0.792 (± 0.03)	0.695 (±0.07)	0.540 (±0.04)	0.580 (±0.05)
	W2V (CBOW)*TF-IDF	0.765 (±0.03)	0.691 (±0.10)	0.519 (±0.04)	0.557 (±0.05)
	W2V (SG)*TF-IDF	0.776 (± 0.02)	0.647 (± 0.11)	0.521 (± 0.04)	0.555 (±0.05)
TFIDF	TF-IDF (w1)	0.881 (±0.02)	0.677 (±0.11)	0.782 (±0.18)	0.710 (±0.13)
	TF-IDF (ch4)	0.887 (±0.02)	0.699 (±0.08)	0.800 (±0.15)	0.731 (±0.10)
	TF-IDF (w1+ch4)	**0.893 (±0.02)**	0.698 (±0.09)	**0.816 (±0.15)**	**0.735 (±0.10)**
Word2Vec & TFIDF	W2V+TF-IDF (w1)	0.876 (±0.02)	0.700 (±0.07)	0.757 (±0.13)	0.716 (±0.09)
	W2V+TF-IDF (ch4)	0.884 (±0.02)	0.713 (±0.07)	0.744 (±0.13)	0.718 (±0.08)
	W2V+TF-IDF(w1+ch4)	0.887 (±0.02)	**0.714 (±0.07)**	0.778 (±0.13)	0.731 (±0.08)
Doc2Vec & TFIDF	D2V+TF-IDF (w1)	0.861 (±0.03)	0.664 (±0.09)	0.672 (±0.15)	0.659 (±0.10)
	D2V+TF-IDF (ch4)	0.867 (±0.02)	0.668 (±0.08)	0.666 (±0.14)	0.662 (±0.10)
	D2V+TF-IDF(w1+ch4)	0.877 (±0.02)	0.694 (±0.10)	0.686 (±0.11)	0.686 (±0.10)

*boldface indicates the best evaluation score in each column

The performance of the classifier improved when a concatenated TF-IDF vector space (w1 + ch4) was used instead of only unigram word or four-gram character vectors. The classifier using TF-IDF (w1+ch4) features obtained the highest training accuracy score of 0.893 (std. dev. 0.02) and training F-score of 0.735 (std. dev. 0.10). This was to be expected as character n-grams features are known for their robustness to noise (i.e. misspellings, obfuscations and unseen words). Surprisingly, the paragraph embedding vectors did not outperform the word embedding vectors. We also noted that the weighted TF-IDF word embedding vectors did not outperform the individual TF-IDF or word embedding.

TABLE VI. TEST SET COMPARISON OF DIFFERENT MODELS

Model		Concatenated Models (Testing)			
		Acc	*R*	*P*	*F1*
Word2Vec	W2V (CBOW & SG)	0.807	0.790	0.567	0.621
	W2V (CBOW)*TF-IDF	0.794	0.809	0.559	0.614
	W2V (SG)*TF-IDF	0.813	0.703	0.573	0.616
TFIDF	TF-IDF (w1)	0.898	0.730	0.771	0.748
	TF-IDF (ch4)	0.902	0.743	**0.827**	0.771
	TF-IDF (w1+ch4)	**0.907**	0.771	0.808	0.787
Word2Vec & TFIDF	W2V+TF-IDF (w1)	0.891	0.810	0.753	0.779
	W2V+TF-IDF (ch4)	0.897	0.794	0.782	0.787
	W2V+TF-IDF(w1+ch4)	0.904	**0.817**	0.810	**0.813**
Doc2Vec & TFIDF	D2V+TF-IDF (w1)	0.871	0.732	0.688	0.707
	D2V+TF-IDF (ch4)	0.871	0.731	0.686	0.706
	D2V+TF-IDF(w1+ch4)	0.880	0.736	0.720	0.728

*boldface indicates the best evaluation score in each column

From the results, it is clear that the classifier with TF-IDF vector models performed the best in terms of accuracy. However, the concatenated Word2Vec with word unigram and character four-gram features scored the highest test recall (0.817) and test F-score (0.813). To summarize, for the test set, the logistic regression classifier using TF-IDF (w1+ch4) features obtained the highest test accuracy score of 0.907 and second highest F-score of 0.787. Surprisingly, all the word2vec with TF-IDF models outperformed doc2vec with TF-IDF models. Given that WhatsApp messages are considered to be longer than a sentence (see Figure 2), one would expect paragraph models (doc2vec) to perform better than word models (word2vec).

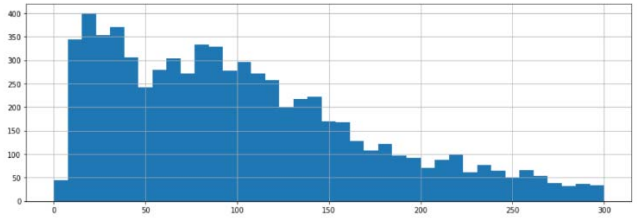

Fig. 2. Character Length of WhatsApp Messages

V. CONCLUSION

In this paper, we presented an approach using distributed word vectors with a machine learning classifier to automatically classify violent incidents taking place in South Africa. We experimented with word embeddings, a numeric semantic representation of words based on unsupervised learning. Word embeddings are low dimensional as they represent tokens as dense floating point vectors as opposed to bag-of-words or n-gram models, which are sparse and high-dimensional.

Our word embedding experiments included both Word2Vec and Doc2vec. Word2Vec implements a word embedding model that basically consists of a neural network that tries to learn a language model. Doc2Vec, which is an extension of Word2Vec, uses a paragraph vector with an unsupervised algorithm that learns fixed-length feature representations from variable length documents. Furthermore the result should be more effective than bag-of-words or n-gram models because word order (within a narrow context) is taken into consideration. Our experiment shows that this is not always the case as Word2Vec models outperformed Doc2Vec models, while traditional TF-IDF models outperformed Word2Vec (with TF-IDF weights), Word2Vec and Doc2Vec models. From the results, one can conclude that the logistic regression classifier using TF-IDF (w1+ch4) features obtained the highest test accuracy score of 0.907 compared to an accuracy score of 0.904 for Word2Vec and 0.880 for Doc2Vec.

Possible future research includes employing pre-trained English and Afrikaans word vectors, such as fastText, in creating a multilingual pre-trained word vector space for the classifiers. These pre-trained word vectors can be used to create an embedding layer and train a deep learning model, such as a Recurrent Neural network (RNN), with the pre-trained word embedding. Additionally, since the models used in this paper are known for losing word order, we will also investigate BERT embeddings and similar language models.

REFERENCES

[1] F. Sebastiani, "Machine learning in automated text categorization," *ACM Comput. Surv.*, vol. 34, no. 1, pp. 1–47, 2002.

[2] W. Wang, R. Kennedy, D. Lazer, and N. Ramakrishnan, "Growing pains for global monitoring of societal events," *Science (80-.).*, vol. 353, no. 6307, pp. 1502–1503, Sep. 2016.

[3] C. Raleigh, A. Linke, H. Hegre, and J. Karlsen, "Introducing ACLED: An Armed Conflict Location and Event Dataset," *J. Peace Res.*, vol. 47, no. 5, pp. 651–660, 2010.

[4] J. A. Goldstone *et al.*, "A Global Model for Forecasting Political Instability," *Am. J. Pol. Sci.*, vol. 54, no. 1, pp. 190–208, 2010.

[5] I. Salehyan *et al.*, "Social Conflict in Africa: A New Database," *Int. Interact.*, vol. 38, no. 4, pp. 503–511, 2012.

[6] M. Croicu and R. Sundberg, *UCDP GED Codebook version 17.1*. Uppsala: Department of Peace and Conflict Research, Uppsala University, 2017.

[7] B. Liu and L. Zhang, "A survey of opinion mining and sentiment analysis," in *Mining Text Data*, C. C. Aggarwal and C. Zhai, Eds. Springer, 2012, pp. 415–463.

[8] Y. Yang and J. O. Pedersen, "A comparative study on feature selection in text categorization," in *In Proceedings of ICML-97, 14th international conference on machine learning*, 1997, pp. 412–420.

[9] T. Mikolov, I. Sutskeve, K. Chen, G. Corrado, and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality," in *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013*, 2013, pp. 3111–3119.

[10] Q. V. Le and T. Mikolov, "Distributed Representations of Sentences and Documents," in *Proceedings of the 31st International Conference on Machine Learning*, 2014, vol. 32, pp. 1188–1196.

[11] E. Kotzé, B. Senekal, and W. Daelemans, "Automatic classification of social media reports on violent incidents in South Africa using machine learning" *South African Journal of Science*, In Press.

[12] J. Cohen, "A Coefficient of Agreement for Nominal Scales," *Educ. Psychol. Meas.*, vol. 20, no. 1, pp. 37–46, 1960.

[13] H. He and E. A. Garcia, "Learning from Imbalanced Data," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 9, pp. 1263–1284, 2009.

[14] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning," *J. Mach. Learn. Res.*, vol. 18, no. 17, pp. 1–5, 2017.

[15] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[16] M. V. Mäntylä, D. Graziotin, and M. Kuutila, "The evolution of sentiment analysis—A review of research topics, venues, and top cited papers," *Comput. Sci. Rev.*, vol. 27, pp. 16–32, 2018.

[17] F. Enríquez, J. A. Troyano, and T. López-Solaz, "An approach to the use of word embeddings in an opinion classification task," *Expert Syst. Appl.*, vol. 66, pp. 1–6, Dec. 2016.

[18] J. Lilleberg, Y. Zhu, and Y. Zhang, "Support vector machines and Word2vec for text classification with semantic features," in *2015 IEEE 14th International Conference on Cognitive Informatics & Cognitive Computing (ICCI*CC)*, 2015, pp. 136–140.

[19] J. Zhao, M. Lan, and J. F. Tian, "ECNU: Using Traditional Similarity Measurements and Word Embedding for Semantic Textual Similarity Estimation," in *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, 2015, pp. 117–122.

[20] E. A. Corrêa Júnior, V. Q. Marinho, and L. B. dos Santos, "NILC-USP at SemEval-2017 Task 4: A Multi-view Ensemble for Twitter Sentiment Analysis," in *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, 2017, pp. 611–615.

[21] R. Řehůřek and P. Sojka, "Software framework for topic modelling with large corpora," in *Proceedings of the LREC Workshop on New Challenges for NLP Frameworks*, 2010, pp. 45–50.

[22] J. H. Lau and T. Baldwin, "An Empirical Evaluation of doc2vec with Practical Insights into Document Embedding Generation," in *Proceedings of the 1st Workshop on Representation Learning for NLP*, 2016, pp. 78–86.

[23] R. Caruana, N. Karampatziakis, and A. Yessenalina, "An empirical evaluation of supervised learning in high dimensions," in *Proceedings of the 25th international conference on Machine learning - ICML '08*, 2008, pp. 96–103.

[24] R. Řehůřek, "Gensim Doc2Vec Tutorial on the IMDB Sentiment Dataset," 2018. [Online]. Available: Gensim Doc2Vec Tutorial on the IMDB Sentiment Dataset. [Accessed: 01-May-2019].

[25] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Inf. Process. Manag.*, vol. 45, no. 4, pp. 427–437, 2009.