A Comparative Study of Neural Network models for Sentiment Analysis of Code-Mixed Tweets

Ugo Loobuyck
Master's student at Uppsala University
Department of Linguistics and Philology
Uppsala, Sweden
ugo.loobuyck.0900@student.uu.se

Abstract

Sentiment classification of social media text is an active area of research that improves insight retrieval for customer satisfaction or marketing strategy adaptation purposes, for example. Nowadays, social globalization induces noisy data from users who use more than one language at the same time to express their thoughts and feelings, this is a type of code-switching called code-mixing. In this paper, we implement and compare the impact of several preprocessing techniques for that type of noisy data, as well as different types and depths of neural network models to classify English-Spanish tweets into positive, neutral and negative classes. Additionally, we compare two pre-trained multilingual word embeddings (GloVe Twitter and fastText) and implement character embeddings as a complement. We train three shallow baselines, a CNN and an attention-based LSTM network to figure out what type of model captures best the sentiment of unseen tweets. Our low amount of data doesn't allow a tremendous improvement of overall performance, but we manage to drastically reduce the number of out-of-vocabulary (OOV) words thanks to preprocessing and partly deal with unbalanced classes during training.

1 Introduction

With an always increasing growth of demand for text classification efficiency, it is important to keep track of which subproblems have or have not been tackled already, and to measure their performance. Although social media like Facebook, Twitter or Reddit offer unlimited quantities of data that we can use to build tools that can detect sentiment or satisfaction, the amount of noise in this data forces us to adapt these tools and make them more robust.

Code-mixed sentences, that are composed of at least two different languages, induce several issues that one must deal with when building a text classifier. Intuitively, it is complicated to thoroughly preprocess text that uses two main languages at once, and state-of-the-art tools that would deal with monolingual text will not necessarily be adequate for code-mixed text. For example, how do we perform spellchecking on a sentence that contains an English word, a Spanish word and a word used in both languages? In the same way, the use of pre-trained word embeddings such as GloVe[1] or word2vec was proved to augment NLP-oriented neural network models, but is it still the case on code-mixed data, and if not, how can we still benefit from these meaningful vectors?

In this comparative study, we will investigate such questions for the case of code-mixed "Spanglish" tweets (containing at least one word in both English and Spanish) to figure out which features and components are worth using. The results obtained will be discussed and will hopefully open the doors to more efficient ways of adapting neural network architectures to sentiment extraction of code-mixed noisy data.

2 Related work

Micro-blogging emotion analysis has been one of the key NLP domains of study in the past decade, and a certain amount of previous work has been done to improve the performance of models ranging from the simplest sentiment classifier for short text to complex neural networks for cross-lingual or code-mixed sentiment analysis.

The major part of the previous work has been done on monolingual datasets, usually from tweets (Kouloumpis et al., 2011), and with, for example, the use of emoticons and support vector machines in distant learning (Go et al., 2009). More generally, Kharde and Sonawane (2016) provides a good comparative survey of some available tech-

[1] https://nlp.stanford.edu/projects/glove/

niques for sentiment analysis of Twitter text, including preprocessing, feature extraction and evaluation methods.

More recently, researchers have tackled the problem of resource disparity between languages by using more contemporary state-of-the-art models involving neural networks. These networks generally manage to capture cross-lingual information with, for example, the attention mechanism (Bahdanau et al., 2014). Zhou et al. (2016) uses this mechanism as well as LSTM layers on both word- and sentence-level to determine their relative importance and therefore predict the overall sentiment. Dos Santos and Gatti (2014) investigate deep convolutional networks to exploit character- to sentence-level information and obtain good results on several datasets.

What interests us most is the use of machine learning techniques for code-mixed short texts, which has been partly investigated. Mishra et al. (2018) mostly used SVM classifiers to deduce polarity, while Ghosh et al. (2017) used Multilayer Perceptrons and yield good results. Wang et al. (2015) and Wang et al. (2016) focus on emotion prediction via bilingual information extraction using attention mechanisms. Joshi et al. (2016) introduces sub-word level representation of tweets to capture the sentiment value of important morphemes.

3 Approach

In this paper we focus on three main points of this subproblem of sentiment analysis: preprocessing code-mixed data, the shortcomings of several pre-trained word embeddings, and deep-learning model components. The overall aim is to determine which existing tools are most efficient for automatically classifying noisy and scarce data into several classes. We will therefore implement a number of shallow and deep neural networks, compare them and reflect on these experiments.

3.1 Data

We use the annotated data set provided by the SentiMix 2020 competition for Spanglish, which allows us to perform supervised learning. This data is composed of 15000 code-mixed tweets with positive, neutral or negative annotations. Additionally, the language or type of each word is annotated following the work of Molina et al. (2016). The three classes are unbalanced since 7500 tweets are annotated as positive, 5000 as neutral and 2500 as negative. To counter this problem, we compute class weights with *scikit-learn* that we pass on to our enhanced models (except one) during training time in order to manually give a higher focus onto classes proportionally to their under-representation. To be more accurate, we assign to our loss function a lower weight for the over-represented positive class and a higher weight for the under-represented negative class.

	preprocessing steps	**% of OOV**
(1)	no preprocessing	38.78
(2)	(1) − stopwords	39.18
(3)	(2) + tweet tokenizer	37.06
(4)	(3) + wordplay	36.74
(5)	(4) − usernames	29.76
(6)	(5) − URLs	**14.49**

Table 1: Percentage of out-of-vocabulary words in the GloVe word embedding depending on preprocessing tools

3.2 Preprocessing

We partly follow the work of Dutta et al. (2015) in order to maximize the clarity of our data. Table 1 displays the percentage of out-of-vocabulary words when using the GloVe Twitter pre-trained embeddings.

Our first step is to remove the stopwords, which are usually not informative for our task. We concatenate the English and Spanish stopword sets from the NLTK library, but leave out some useful words like negations which can indicate polarity, or for example "many" which can intensify opinions. We then use the NLTK tweet tokenizer which captures entire smileys and emoticons, usernames and URLs, and splits punctuation marks.

We use this tokenizer to deal with "wordplay", which is when some characters in a word are multiplied to mark intensity. We reduce these repetitions to 3 characters (the word "looooool" becomes "loool") in order to have better chances to get a match in the pre-trained embeddings while saving some degree of intensity that we hope to capture thanks to character embeddings. Finally, we remove Twitter usernames and URLs which are often not informative.

In a concern of time, we leave additional tools such as spell check, shortcut correction and

acronym expansion to future work. It is overall extremely hard to perform a complete preprocessing given the noisiness of the data. In total, we reduce the ratio of OOV words by 24.29% on the GloVe embeddings, and 23.38% on the FastText[2] embeddings (from 40.91% of OOV without preprocessing to 17.53%) which will affect the performance of subsequent models.

3.3 Word embeddings

In this project, we choose to integrate pre-trained word embeddings which have been proved to help grasping the semantic relationships between words by giving them a high-dimensional vector representation. It would be intuitive to pick GloVe Twitter as it was trained using data corresponding to ours, but even though GloVe is multilingual, the words are not aligned between languages, which in theory means that the more words from several languages occur in a sample, the more confused our system will be, as it will try to draw meaning from word representations of different high-dimensional areas. In practice, this means that the word "dog" in English and "perro" in Spanish will potentially be far away in the word embedding even though they should be very close, therefore creating noise.

To counter this, we use the FastText aligned word embeddings for English and Spanish described in Joulin et al. (2018) and Bojanowski et al. (2017). These embeddings are aligned on the English language and trained on Wikipedia data. When concatenating the English and Spanish embeddings, we assume that words occurring in both language embeddings are closely represented, and we therefore deal with them by averaging their respective vectors in the newly created space. The result is a 300-dimensional aligned bilingual word embedding.

3.4 Character-based embedding

As a complement of pre-trained word embeddings, we use randomly initialized character embeddings that we concatenate with the word embedding after passing through a bidirectional recurrent network, as described in figure 1. This method makes sense for a task using Twitter data as it usually is noisy and emotions are expressed through spelling or character multiplication, which induces a lot of OOV words. Character embeddings allows a better representation of each word in the sentence, which we hope will improve the overall classification performance. We use 25 LSTM units on each layer, resulting in a 50-dimensional character embedding for each word. The character embedding is then concatenated to the GloVe or fastText word vector, bringing the final word representation to 250 dimensions (using Glove) or 350 dimensions (using fastText).

[2]https://fasttext.cc/

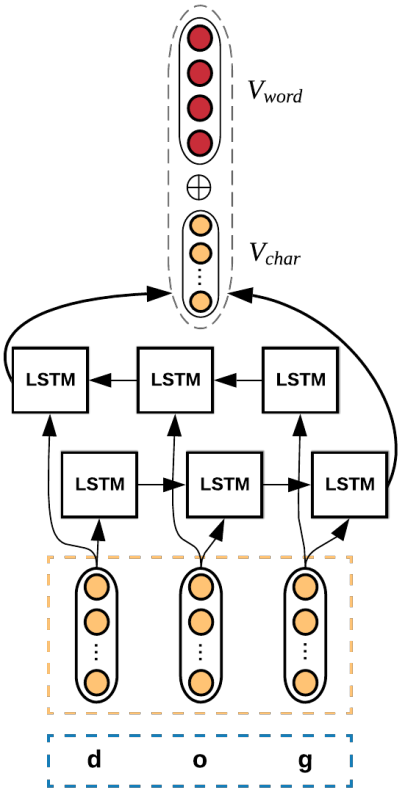


Figure 1: Joint word-character embedding mechanism with bi-LSTM network.

4 Experimental setup

In this section we describe the technical aspects of our project. We tune each of our models using hyperparameter tuning and therefore maximize the efficiency of the learning process. Therefore, hyperparameters such as learning rate, batch size or epochs vary from one model to another.

To measure the quality of the proposed models, we will look at precision, recall and F1 measure, three good indicators for multi-class classification. The different models are implemented with the Keras API using TensorFlow as backend.

We use early stopping during training to prevent overfitting, as well as dropout on all layers

of our enhanced models as described in Srivastava et al. (2014), and weight regularization on activated layers. We save 10% of the data for testing. During training, 15% of the remaining data is used as a validation set. Since we are working in a multi-class context, we systematically use an output layer with three nodes activated by a softmax function, and minimize a categorical cross-entropy loss function.

4.1 Baselines

We implement three shallow baselines, all of which are not thoroughly preprocessed (simply tokenized) and using the GloVe Twitter (200 dimensions) pre-trained embeddings: a dense model, a CNN model with one convolution-pooling block without fully connected layers, and an RNN model with one recurrent layer (64 LSTM units).

4.2 Convolutional network

Convolutional neural networks have turned out to be a significant enhancement in NLP tasks in recent years. First described in LeCun et al. (1998) and proved to be efficient for image analysis (two-dimensional), they can also be used with text (one-dimensional) in order to detect different patterns.

Figure 2 describes a convolution-pooling block. A given filter (of size s) is slid over successive regions of s words. Each element of the region matrix M_r is multiplied by the corresponding element in the filter matrix M_f. The resulting element e is the sum of all products. The feature map is composed of all successive elements e to which a bias is added and an activation function is applied. The shape of the feature map is described as follows:

$$S_{map} = (n - s + 1) \times 1$$

After a full convolution, we perform a max pooling step that only keeps the highest value from the feature map.

We experiment on different combinations of components (type of word embedding, character embeddings...), and try to keep the network shallow: given the low amount of data, increasing the number of layers increases the number of parameters and can therefore hurt the model. We therefore apply only one convolution-pooling block followed by a dense layer and the output layer. The models were tuned empirically. We use a learning rate within the interval $[0.001; 0.0001]$, mini-batches of size 64, a dropout rate of 0.5 and regularization $L2 = 0.001$. Finally, we train the model over 30 to 40 epochs.

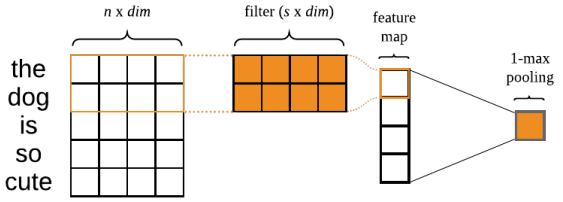


Figure 2: Convolution-pooling block with a single filter and a single filter iteration.

4.3 Attention-based recurrent network

We train a recurrent neural network using 256 LSTM units in a single unidirectional layer, and integrate an attention mechanism (Yang et al., 2016) on top of it that computes a context vector at each run.

We hope that the model will manage to focus on the most informative words in the tweets, leaving the less useful words out during weights update. Like in all our other models, we do not learn the word embedding parameters but only the deeper layers' parameters. We experiment on GloVe and FastText embeddings, both with all the features described before: extensive preprocessing, character embeddings, and class weights.

Here again, models were tuned empirically. We set $lr = 0.003$ when using GloVe Twitter and $lr = 0.0003$ for FastText. We use larger mini-batches (256) for time efficiency reasons. Dropout is set to 0.75 and L2 regularization to 0.03. We train our models over 30 epochs.

5 Results and discussion

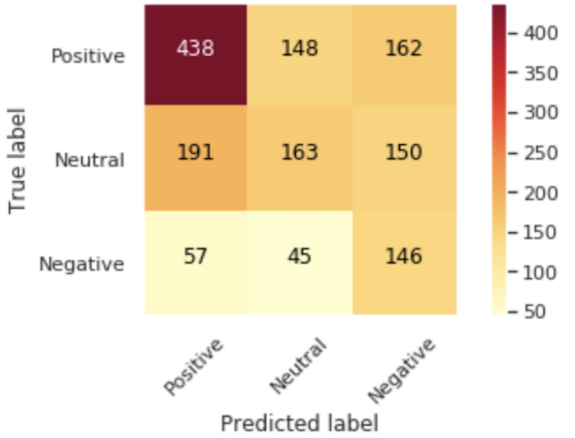
All the results measured with precision, recall and F-measure are averaged on five different runs of each models to avoid exceptions and ensure classification consistence. Results are shown in Table 2, and are separated in two parts, using GloVe embeddings on the one side and FastText embeddings on the other. The reason for this is concerning the legitimacy of the Glove multilingual representations in this particular code-mixed context.

5.1 Impact of words embeddings

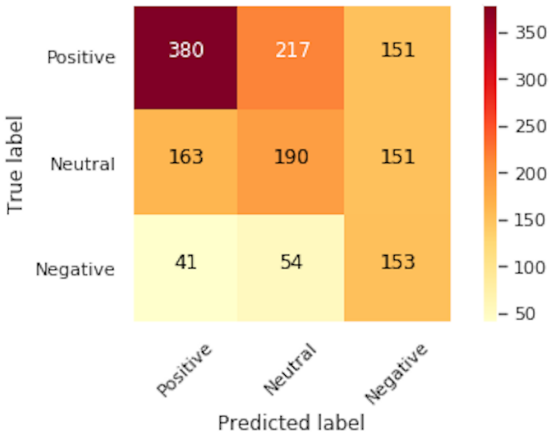
We believe that the rather small gap in the different metrics between augmented models using GloVe and those using FastText (about 1 to 2%) should be more significant with a larger and more evenly

	Model	Precision	Recall	F1
GloVe	Dense	0.46	0.502	0.401
	CNN	0.464	0.501	0.457
	LSTM	0.416	0.521	0.421
	CNN + PREP	0.507	0.493	0.489
	CNN + PREP + charEmbedding	0.521	0.507	0.507
	LSTM + PREP + attention + charEmbedding	0.52	0.486	0.492
fastText	CNN + PREP	0.528	0.503	0.507
	CNN + PREP + charEmbedding	**0.531**	0.507	**0.510**
	CNN + PREP + charEmbedding − class weights	0.521	**0.547**	0.494
	LSTM + PREP + attention + charEmbedding	0.507	0.47	0.469

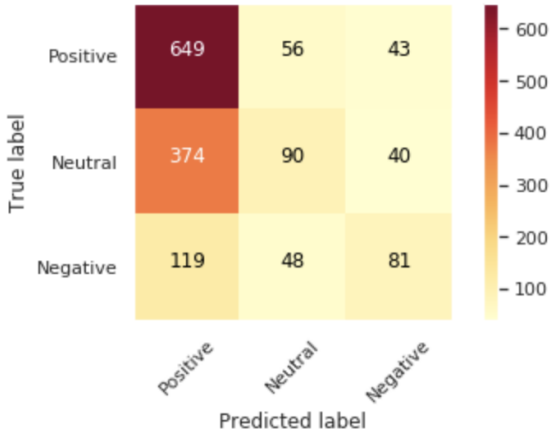
Table 2: Results for all baselines and augmented models averaged on five runs.



(a) Convolutional network



(b) Recurrent network



(c) Convolutional network without weights

Figure 3: Confusion matrices for two augmented models (line 8 and 6 in Table 2, respectively) using extensive preprocessing, character embeddings and class weights.

represented data set. The fact that the GloVe embeddings are not aligned between languages should, in theory, result in worse results than reported here. More specifically, our enhanced LSTM model trained with GloVe yields better results than the same model trained with fastText (0.492 F1 against 0.469), but our enhanced CNN model yields better results when trained with FastText (0.510 F1 against 0.507).

These variations might yet simply be due to the

specificities of each type of model. Nevertheless, we believe that aligned FastText embeddings are more legitimate to be used in this particular case. We also acknowledge that the averaging of bilingual word vectors is only a temporary solution to embedding concatenation, and we leave to future work the improvement of that method.

The main issue we encounter here concerns the model where we do not add class weights to the model's loss function during training time. Indeed, and as we will see later in figure 3, this model does not perform well because of unbalanced class during training. The only difference with our best model is the absence of class weight, which results in a 1% decrease in precision. On the other side, recall increases by 4% because of the the positive class which is overly represented and predicted during testing. This model is therefore not reliable and shows that we either need balanced data or to add class weights when using unbalanced data.

5.2 Quantitative analysis

Figure 3 shows the confusion matrices for our two best models using a convolutional network with FastText embedding and all components (a), and a recurrent network using GloVe and all components (b). The test set used here is composed of 1500 sentences (10%), preprocessed in the exact same way as the training sentences. Both matrices look rather similar, with a strong prediction tendency towards the positive class.

(a) shows that 686 sentences were classified as positive with a 63.8% precision, therefore achieving the best results. The neutral class achieves a 45.8% precision, and the negative class yields the worst results with a 31.9% precision. Nevertheless, it is interesting to note that the negative class achieves the best recall (58.9%), which means that the model is less often wrong when classifying sentences as negative than for other classes. The same goes for the recall of the negative class in (b) which yields the highest result (61.7%). We believe that this is mostly (but not only) due to the class weights that act against the under-representation of that class.

(b) shows that our recurrent network with all components has more trouble to correctly differentiate positive and neutral patterns, with only 50.8% recall for the positive class, against 58.6% in (a). The precision of the positive class is yet comparable between models (63.8% in (a) against 65.1% in (b)), which shows a generally good performance when classifying sentences as positive. Finally, we note that the precision of the neutral class is much lower in (b) (41.2%) than in (a) (45.8%), which confirms that this model, although achieving correct results, struggles to generalize on neutral and positive cases. Perhaps the reason for that is that the attention mechanism implemented in (b) tends to focus on the most informative parts of each sentence, with intuitively does not make much sense when classifying neutral sentences, in which words are rarely polarized.

(c) shows an imbalance in predictions towards the positive class. This is most likely due to an over-representation of the positive class in the training set, which influences the model and pushes it to predict most samples as positive. This gives us a recall of 86.8% for the positive class but 19.4% for the neutral class and 32.7% for the negative class. This fits with the results obtained in table 2 since the metrics are computed using a weighted average on classes and we mostly have positive instances in the test set, which results in a high overall recall.

5.3 Qualitative analysis

Table 3 presents a few concrete examples taken from the test data set. We selected, for each class, one example and one counter-example to try and figure out what types of patterns promoted a rather good or bad classification. All example outcomes were predicted with our best performing model (line 8 in Table 2), which uses a convolution with 64 filters of size 3.

For the positive examples, we can tell with some degree of certainty that the model picked up most of the information in the words "mega cute" in the first sentence and "cabron" in the second. These words are strongly polarized, but here the model did not seem to have captured the entire meaning of the second sentence, which was actually dealing with falling in love. Perhaps an LSTM-based model with attention mechanism, which are known to work slightly better in that context where the sense is not what it seems, would have made a better guess.

The neutral examples are among the hardest to classify, as discussed before, because instead of picking sharp patterns of polarized sequences, we ask our CNN model to look for rather flat sen-

Sentence	Prediction	Correct tag
"Ok enserio stickers Snapchat estan mega cute"	Positive	Positive
"Enamorarse mutuo debe pa ' l cabron ."	Negative	
"Dile babyyy"	Neutral	Neutral
"4th fuego blue dream day"	Positive	
"office .. .. Senda mierda"	Negative	Negative
"no idea ☹ "	Neutral	

Table 3: Examples of correctly and wrongly classified tweets from the test data set, using our best model.

tences without emotion. We can imagine that the word "fuego" has a strong positive meaning in a Twitter informal context ("fuego" or "fire" often refers to something exciting), which might have decided the positive classification.

Finally, it is easy to figure that negative classification relies on sharp patterns, often containing insults, swears words, capitalized words and negative emojis/emoticons. The model manages to correctly predict the first sentence, surely mostly due to the word "mierda" that is contained is many other code-mixed or even monolingual Spanish sentences. The last sentence in Table 3 is slightly harder to analyze for our CNN model, as the text alone ("no idea") would almost certainly be classified as neutral by any human annotator, but we hoped that the use of character embeddings would correct the absence of such emojis in the GloVe and FastText representations, which it did not for that particular instance.

6 Conclusion and future work

In this paper we investigated and implemented some of the tools necessary for multi-sentiment analysis of code-mixed Spanish-English tweets. We have focused on different aspects, including extensive preprocessing, the choice in pre-trained word embeddings, the addition of LSTM-based character embeddings, and class weights to counter class over/under-representation. We have trained several CNN and RNN models and experimented with the various network components.

Our best model is a convolutional model that shows a 10.9% improvement in F-measure over the worst baseline and 5.3% improvement over the worst convolution-based baseline model. We manage to reduce by 24.29% the amount of out-of-vocabulary words in our noisy data, and partly counter the gap in representation between positive, neutral and negative classes. We managed to implement character-based embeddings that allow a better representation of each word, therefore improving performances by about 1% compared to models that do not use them. We prefer aligned multilingual word representations from FastText to those unaligned proposed by GloVe, which again improves the results by about 0.03%, on the best models from each embedding experiment set.

In future work, it would be interesting to focus on using the word-level language annotations and other code-mixing oriented features, that we could include in our models. It would also be useful to gather more data to enhance the learning process, and extend our preprocessing step to lower the amount of OOV words, therefore allowing a better representation of sentiment polarity in word sequences. Finally, the alignment of multilingual word embeddings seems essential to that particular task, and it would be interesting to either further develop the method used here, or use in-development tools such as ConceptNet Numberbatch[3] that use aligned multilingual embeddings.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Cicero Dos Santos and Maira Gatti. 2014. Deep convolutional neural networks for sentiment analysis of short texts. In *Proceedings of COLING 2014, the 25th International Conference on*

[3] https://github.com/commonsense/conceptnet-numberbatch

Computational Linguistics: Technical Papers, pages 69–78.

Sukanya Dutta, Tista Saha, Somnath Banerjee, and Sudip Kumar Naskar. 2015. Text normalization in code-mixed social media text. In *2015 IEEE 2nd International Conference on Recent Trends in Information Systems (ReTIS)*, pages 378–382. IEEE.

Souvick Ghosh, Satanu Ghosh, and Dipankar Das. 2017. Sentiment identification in code-mixed social media text.

Alec Go, Richa Bhayani, and Lei Huang. 2009. Twitter sentiment classification using distant supervision. *CS224N Project Report, Stanford*, 1(12):2009.

Aditya Joshi, Ameya Prabhu, Manish Shrivastava, and Vasudeva Varma. 2016. Towards sub-word level compositions for sentiment analysis of Hindi-English code mixed text. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 2482–2491, Osaka, Japan. The COLING 2016 Organizing Committee.

Armand Joulin, Piotr Bojanowski, Tomas Mikolov, Hervé Jégou, and Edouard Grave. 2018. Loss in translation: Learning bilingual word mapping with a retrieval criterion. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

Vishal Kharde and Sheetal Sonawane. 2016. Sentiment analysis of twitter data: A survey of techniques. *International Journal of Computer Applications*, 139:5–15.

Efthymios Kouloumpis, Theresa Wilson, and Johanna Moore. 2011. Twitter sentiment analysis: The good the bad and the omg! In *Fifth International AAAI conference on weblogs and social media*.

Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Pruthwik Mishra, Prathyusha Danda, and Pranav Dhakras. 2018. Code-mixed sentiment analysis using machine learning and neural network approaches.

Giovanni Molina, Fahad AlGhamdi, Mahmoud Ghoneim, Abdelati Hawwari, Nicolas Rey-Villamizar, Mona Diab, and Thamar Solorio. 2016. Overview for the second shared task on language identification in code-switched data. In *Proceedings of the Second Workshop on Computational Approaches to Code Switching*, pages 40–49, Austin, Texas. Association for Computational Linguistics.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

Zhongqing Wang, Sophia Lee, Shoushan Li, and Guodong Zhou. 2015. Emotion detection in code-switching texts via bilingual and sentimental information. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 763–768.

Zhongqing Wang, Yue Zhang, Sophia Lee, Shoushan Li, and Guodong Zhou. 2016. A bilingual attention network for code-switched emotion prediction. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 1624–1634.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1480–1489.

Xinjie Zhou, Xiaojun Wan, and Jianguo Xiao. 2016. Attention-based LSTM network for cross-lingual sentiment classification. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pages 247–256.