Classifying emotions in text using Machine Learning

Stefan Kandić
Faculty of Technical Sciences
Novi Sad, Serbia
stefan.kandic@yahoo.com

***Abstract*—*Machine Learning* techniques have proven to be a powerful tool for different language processing tasks, especially sentiment analysis. Encouraged by these results, I propose a solution to automatically detect emotion from text, using a dataset of 450 thousand social media posts and comments belonging to 6 classes (*joy*, *sadness*, *love*, *surprise*, *fear* and *anger*).**

This paper describes more traditional *Natural Language Processing* (NLP) and modern *Deep Learning* (DL) techniques comparing their advantages, disadvantages and finally performance on the task at hand. Traditional models include popular classifiers *Naive Bayes* and *Support Vector Machine* (SVM). DL model is based on the *Recurrent Neural Network* (RNN), or more specifically *Long Short Term Memory* network (LSTM).

The deep learning model shows a performance of 94%, which is considered a state-of-the-art result for sentiment analysis tasks. This was achieved despite a much simpler architecture compared to best-ranking models due to hardware limitations. NLP models display a slightly inferior performance of 89%.

***Index Terms*—emotion classification, machine learning, natural language processing, deep learning, sentiment analysis**

I. Introduction

In the past, text classification was mostly used for labeling text as positive or negative, but given enough data we can extract more nuanced and wider-ranged information, such as emotion. In the era of social media, it has never been easier to share opinions and feelings on just about anything, and being able to analyze thousands of posts to understand public perception about a topic of interest could have huge commercial value. Examples would be predicting the stock market prices [1] or presidential election results [2] by analyzing social media posts. It could also prove beneficial in automating the creation of audio books [3], which are widely used today.

The primary objective of this paper is to demonstrate and compare the use of different *Natural Language Processing* (NLP) and *Deep Learning* (DL) techniques in recognizing emotions in an informal text prone to having spelling errors. The set of emotions I will try to identify contains: *joy*, *sadness*, *love*, *surprise*, *fear* and *anger*. First, I will show how to represent text, and later explain and compare more traditional Natural Language Processing against state-of-the-art Deep Learning methods in solving the emotion classification problem.

This paper is organized as follows. Traditional NLP and Deep Learning approaches are presented in Section II and III, respectively. Experimental results are in Section IV, and the paper is concluded in Section V.

II. Traditional approach

Although overshadowed by their Deep Learning counterparts, traditional approach methods still have some applications, especially when there is only a limited amount of training data available. Unlike DL models, they rely on handcrafted features created using encoders. Those features are then provided to a statistical learning model which first learns patterns in the data, and is then able to make assumptions on previously unseen data.

Subsection II-A shows how traditional methods represent text, and subsection II-B introduces models used in the experiment.

A. Text representation

In this approach, the text is represented as the bag (multiset) of words, disregarding grammar and word order. Also, since Machine Learning algorithms take numbers as inputs, we will need to convert the text into a numerical representation. Usually, there are two steps in this process, *tokenization* and *vectorization*. In the following text, I will explain each of those.

1) Tokenization: usually includes:

- Dividing text into a sequence of words
- Removing punctuation, links, etc.
- Removing *stop words* — words that appear frequently but don't add any actual meaning to the text ("a", "the")

By doing this, we will get more robust data, and improve the generalization of the relationship between the text and the labels. It will also generate a set of all unique tokens, the vocabulary (Fig. 1).

i	0	love	1	cats	2	dogs	3

Fig. 1. Example vocabulary

I mentioned that the traditional approach does not understand the order of tokens, but nothing is stopping us from using tokens consisting of multiple words or *n-grams*. N-gram is simply a sequence of n words. Tokens with $n = 1$ are called *unigrams*, with $n = 2$ bigrams and so on. For example, for the phrase *"don't live in New York"*, bigrams would be: *don't live*, *live in*, *in New*, *New York*.

The vocabulary created with the combination of unigrams and higher n-grams would be a useful tool for capturing phrases consisting of multiple words. It would be especially helpful with negations (*don't live*), because it is easier to understand that the sentiment should be inverted, whereas just having tokens *don't* and *live* does not mean that they appeared next to each other. Also, tokens *New* and *York* have a considerably different meaning than *New York*.

2) Vectorization: After we have successfully converted our text into a sequence of words, we need to convert those sequences into numerical vectors.

Using a technique called *one-hot encoding*, we can encode a sentence into a single vector, by indicating the presence or absence of a token in the sentence (Fig. 2). By doing this we will lose the order of words in the sentence. Also, since vocabulary usually contains thousands of words, these vectors will be sparse and thus memory inefficient.

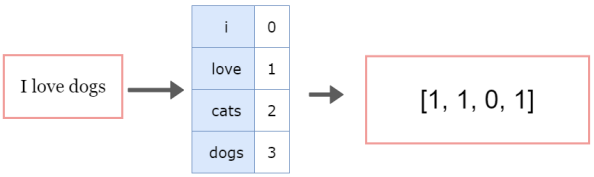

Fig. 2. Encoding in a vector

Note that values in one-hot vectors don't have to be binary, there are different ways of calculating them. One simple encoder is called *count encoder*. For each token in our vocabulary we count how many times it was found in the input sentence. It is based on the notion that having multiple occurrences of the same word in a sentence speaks to a higher importance of that word compared to others.

A more popular encoder is *term frequency-inverse document frequency* (tf-idf). Instead of ones and zeroes, values in the vector will be calculated as the product of the term frequency and the inverse document frequency:

$$tfidf(t,d) = tf(t,d) \cdot idf(t,d). \tag{1}$$

Here the $tf(t,d)$ is the number of times a term t occurs in a document d , and $idf(t,d)$ is the inverse document frequency and can be calculated as follows:

$$idf(t,d) = \log \frac{n_d}{1 + df(d,t)}, \tag{2}$$

where n_d is the total number of documents (in our case sentences), and $df(d,t)$ is the number of documents that contain the term t. The idea behind term frequency is that we want to add greater value to words that frequently appear in the document because they probably represent key words. On the other hand, the goal of the inverse document frequency is to diminish the significance of the words that frequently appear in all documents, because they usually don't contain useful or discriminatory information.

B. Models

There are multiple learning algorithms to choose from, but when it comes to solving document classification tasks most commonly used ones are *Naive Bayes* (NB) and *Support Vector Machines* (SVM) [4] [5].

1) Naive Bayes: Naive Bayes is a straightforward algorithm based on Bayes theorem. Its goal is to find an emotion with the highest probability for the input based on the words found in previously seen samples.

For an input sentence s and the set of all emotions E, we predict its emotion as:

$$\underset{e \in E}{\operatorname{argmax}} P(s|e) \cdot P(e). \tag{3}$$

Basically, we choose an emotion with the biggest value of the product of the conditional probability of the sentence given the emotion $P(s|e)$, and the probability of the emotion $P(e)$. Probability of the sentence is calculated by multiplying conditional probabilities of each word w in the sentence:

$$P(w_1, ..., w_n|e) = P(w_1|c) \cdot ... \cdot P(w_n|e), \tag{4}$$

where the probability of word w given the emotion e $P(w|e)$ is calculated as the number of occurrences of that word in all sentences related to emotion e (forming vocabulary V_e), divided by the total number of words in them:

$$P(w|e) = \frac{|w|}{\sum\limits_{w' \in V_e} |w'|}. \tag{5}$$

Similarly, the probability of an emotion $P(e)$ is determined by the fraction of sentences that belong to emotion e:

$$P(e) = \frac{|e|}{\sum\limits_{e' \in E} |e'|}. \tag{6}$$

2) SVM: Support Vector Machine's objective is to maximize the margin between the data samples. The margin is defined as the distance between the decision line and the training samples that are closest to this line (Fig. 3), which are the so-called support vectors [7].

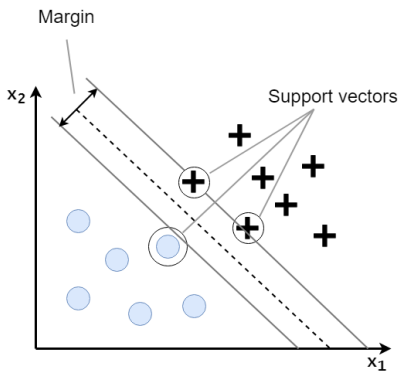

Fig. 3. Linear Support Vector Machine

III. Deep Learning approach

In traditional models we had to manually extract feature vectors from sentences. In deep learning we don't need hand-crafted features, since they are learned from scratch during the training process. Also we were not capable of handling the order of input samples due to not having a memory of the past seen samples. However, by using *Recurrent Neural Networks* (RNN), which are capable of remembering past information, we should be able to understand the structure of sentences much better, since sentiment is heavily impacted by the order of words and not just by words themselves.

Subsection III-A shows how deep learning methods represent text, and subsection III-C explains DL model's architecture used in the experiment.

A. Text Representatation

Words have meanings associated with them. As a result, we can represent them in a vector space where the location and distance between words indicate how similar they are semantically (Fig. 4).

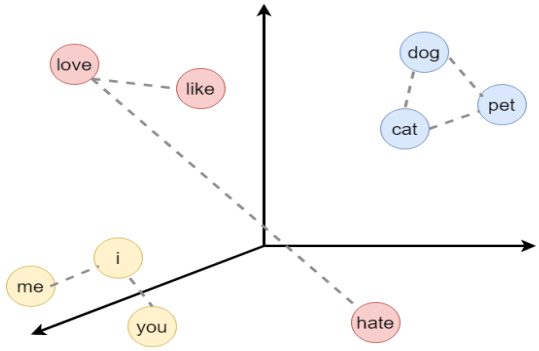

Fig. 4. Simplified semantic space

Usually, we have such an embedding layer as the first layer in our neural network. This layer learns to map words from the vocabulary into word embedding vectors during the training process, such that each word gets mapped to a vector of real values representing that word in the semantic space [6]. The output of this layer will be a matrix of size $m \times n$ (Fig. 5), where m is the number of words in a given input sequence, and n is the dimension of the embedding layer (semantic space).

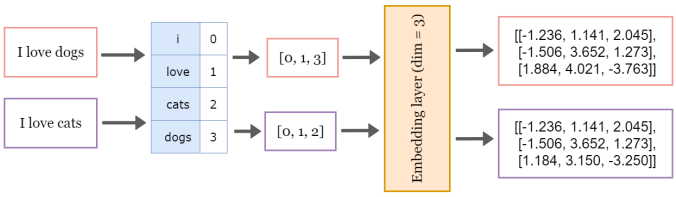

Fig. 5. Embedding layer

B. Understanding the structure of RNN

Let's start by introducing the architecture of an RNN. The following figure 6 is a comparison between the standard feedforward neural network and an RNN [7]:

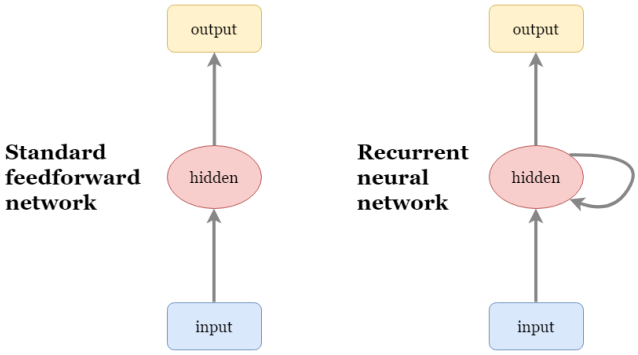

Fig. 6. Feedforward vs Recurrent neural network

As we can see, in a standard feedforward network, information flows from the input to the hidden layer(s), and then to the output layer. On the other hand, in a recurrent network, the hidden layer gets its input from both the input layer and the hidden layer from the previous time step. This kind of information flow allows the network to have a memory of past events.

So, a recurrent neural network can be thought of as multiple networks, each passing a message to a successor allowing "information loop" (Fig. 7).

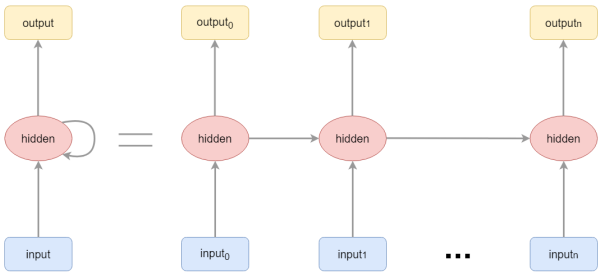

Fig. 7. Recurrent neural network loop

In this paper, I used a special type of RNN called *Long Short Term Memory* [8] network or LSTM. A number of best-ranking models in sentiment analysis use LSTM layers in their architecture and they are considered state-of-the-art [1].

C. Deep Network Architecture

There is no clear cut way to design an architecture of a deep neural network since there are countless possible options. Comparing different architectures is quite difficult, because it

[1] http://nlpprogress.com/english/sentiment_analysis.html

heavily depends on the task at hand, and would take a long time.

Due to hardware limitations, I had to build a network with as few layers as possible. Thus I used a 6-layer network, consisting of an input embedding layer followed by two LSTM layers. As LSTM reads a sentence, gradually its state gets more and more feature-rich at each time step. It causes the end of the sentence to become more feature-rich than the beginning, and by having two of them run in the opposite direction we get features from both ends.

Next up, there is a dense output layer, using *Rectified Linear Unit* (ReLU) [9] as its activation function. It is defined as:

$$f(x) = max(0, x), \tag{7}$$

which just clips all negative values to zero. ReLU's nonlinearity allows us to capture complex relationship in the data.

It is followed by a dropout layer [10], whereby randomly dropping 50% of all units in the network each epoch[2] we try to prevent overfitting. Dropout will introduce redundancy in the network which will then increase its robustness, since we won't be able to rely only on certain neurons in the network to make predictions. Alternatively, we can look at it as creating an ensemble of different models all trained together to correctly predict the label.

And finally, using a softmax output layer, we normalize the output and predict a probability for each of the six emotions.

IV. Results

This section will contain different experiments used and their results. Subsection IV-A includes data that was used. Subsection IV-B shows models that are compared. Subsection IV-C describes metrics used to evaluate solutions, and finally IV-D contains experimental results.

A. Data

Dataset used to train and test the machine learning models in this paper is an "Emotion Classification" dataset from *kaggle* [3]. It consists of more than 450k sentences, all of which have been manually classified into one of six emotions: *joy*, *sadness*, *anger*, *fear*, *surprise* and *love*. These sentences have been extracted from different news articles, interviews, discussion forums and social media. Therefore, the dataset accurately represents most of our online communication.

However, by looking at the distribution (Fig. 8) we can see that the data is somewhat imbalanced, since *anger*, *fear*, *love* and *surprise* represent a considerably smaller portion of the data compared to *joy* and *sadness*. This could potentially cause lower performance of our model for under-represented emotions. It could also favor more popular emotions *joy* and *sadness* with inputs that are somewhat neutral in emotion. This problem could be solved by adding more samples of less-represented emotions and creating another label which would indicate a lack of emotion, but doing this would require a lot of extra grunt work and is out of the scope of this paper.

[2]one pass through the entire training data

[3]https://github.com/ssttefann/EmotionClassification/tree/master/data/dataset

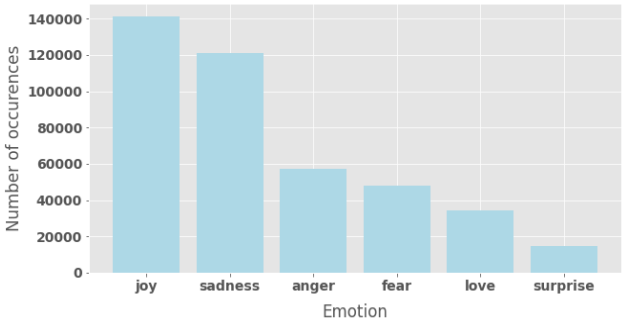

Fig. 8. Emotion distribution

B. Models

In this paper I have compared the following models:

- Naive Bayes with count encoder
- SVM with tf-idf encoder
- SVM with bigrams and tf-idf encoder
- Deep Learning model explained in subsection III-C

C. Metrics

To better understand performance of one model, we must first create a *confusion matrix* (Table I). Each element in this matrix C_{ij} represents the number of observations from group i predicted to be in group j.

TABLE I
CONFUSION MATRIX

True emotions	Predicted emotions					
	joy	***sadness***	***anger***	***fear***	***love***	***surprise***
joy	C_{11}	C_{12}	C_{13}	C_{14}	C_{15}	C_{16}
sadness	C_{21}	C_{22}	C_{23}	C_{24}	C_{25}	C_{26}
anger	C_{31}	C_{32}	C_{33}	C_{34}	C_{35}	C_{36}
fear	C_{41}	C_{42}	C_{43}	C_{44}	C_{45}	C_{46}
love	C_{51}	C_{52}	C_{53}	C_{54}	C_{55}	C_{56}
surprise	C_{61}	C_{62}	C_{63}	C_{64}	C_{65}	C_{66}

Now, the most common metric for evaluating classification performance is accuracy, which is simply the number of correct predictions divided by the number of total predictions. However, it can be misleading sometimes. If we wanted to detect spam emails, working with data containing 9500 non-spam, and 500 spam emails, predicting that every email is not spam would lead to 95% accuracy, even though our model does not actually do its job. That is why it is better to use *precision* and *recall*.

Precision of emotion e (indexed by i) is a fraction of all sentences classified as e which are correct. It is calculated by dividing an element on the main diagonal of the matrix with the sum of the corresponding column:

$$precision = \frac{C_{ii}}{\sum_j C_{ji}}. \tag{8}$$

On the other hand, recall is a fraction of all sentences of true emotion e (indexed by i) which are classified by the model as e. It is calculated by dividing an element on the main diagonal of the matrix with the sum of the corresponding row:

$$recall = \frac{C_{ii}}{\sum_j C_{ij}}. \tag{9}$$

If we calculate these values for that spam detection example, we would get 0 for both precision and recall which more accurately reflects the performance of the model.

Finally, since both of these values are equally important for this task, we can combine them into one using their harmonic mean, and get something called F_1 *score*:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}. \tag{10}$$

Therefore, the final measure of a model's performance will be its average F_1 score across all six emotions.

D. Experimental results

Data was split into train/test sets with 90/10 split, each containing 375k and 40k samples, respectively. In Table II, you can see the performance of all models on the test data.

TABLE II
RESULTS

	Metrics		
	Lowest precision	***Lowest recall***	***Avg F_1 score***
Naive Bayes	0.81 (love)	0.37 (surprise)	0.86
SVM	0.70 (surprise)	0.69 (surprise)	0.89
SVN with Bigrams	0.66 (surprise)	0.66 (surprise)	0.88
Deep Learning	0.86 (surprise)	0.71 (love)	0.94

First of all, we can see that even the lowest F_1 score of 0.86 is still considered pretty good for such a complicated problem. It seems that all traditional models have very similar results. However, Naive Bayes classifier is pretty heavily biased against *surprise*, correctly assigning barely above one-third of the sentences, while other models show more balanced scores across different emotions. Interestingly, using bigrams did not help SVM's performance. I suspect that this is because 450k sentences were enough for unigrams to accurately capture the language model.

While the Deep learning model took significantly longer to train, it has proven its state-of-the-art status, and its performance could only increase with more training and parameter tuning. Also, it seems that my suspicion that models might not perform as well with less-represented data is confirmed, since all the lowest metrics come from the two emotions with the least number of data samples (*love* and *surprise*).

V. CONCLUSION

This paper demonstrates that even the complex concepts such as language and emotion, which although may seem inherently human, can be successfully modeled using machine learning. Whether you make online videos or run for president, having a tool to help you understand your audience better could solve crucial problems you might not even know you have.

Traditional methods have shown surprisingly high results of 89%, but I think it would be interesting to see how they would perform on a smaller fraction of the data. Also, NLP models were trained within minutes, while DL model took a couple of hours to finish its training. But still, Deep Learning has confirmed its state-of-the-art status in the field, yielding a result of 94% which is comparable to much more complex models trained on larger text corpora.

I also intend to continue exploring other techniques that could improve this system, such as fine-tuning pre-trained language representation models like BERT [11], which has been trained at *Google* on an enormous amount of data and shows incredible results in all language processing tasks.

REFERENCES

[1] Nguyen, Thien Hai, Kiyoaki Shirai, and Julien Velcin. "Sentiment analysis on social media for stock movement prediction." Expert Systems with Applications 42.24 (2015): 9603-9611.

[2] Ibrahim, Mochamad, et al. "Buzzer detection and sentiment analysis for predicting presidential election results in a twitter nation." 2015 IEEE international conference on data mining workshop (ICDMW). IEEE, 2015.

[3] Alm, Cecilia Ovesdotter, Dan Roth, and Richard Sproat. "Emotions from text: machine learning for text-based emotion prediction." Proceedings of the conference on human language technology and empirical methods in natural language processing. Association for Computational Linguistics, 2005.

[4] Aman, Saima, and Stan Szpakowicz. "Identifying expressions of emotion in text." International Conference on Text, Speech and Dialogue. Springer, Berlin, Heidelberg, 2007.

[5] Colas, Fabrice, and Pavel Brazdil. "Comparison of SVM and some older classification algorithms in text classification tasks." IFIP International Conference on Artificial Intelligence in Theory and Practice. Springer, Boston, MA, 2006.

[6] "Text Classification", Google. Retrieved March 10, 2020 from https://developers.google.com/machine-learning/guides/text-classification

[7] Sebastian Raschka. 2015. Python Machine Learning. Packt Publishing (pp. 79-81, 568-572).

[8] Sundermeyer, Martin, Ralf Schlüter, and Hermann Ney. "LSTM neural networks for language modeling. " Thirteenth annual conference of the international speech communication association. 2012.

[9] Nair, Vinod, and Geoffrey E. Hinton. "Rectified linear units improve restricted boltzmann machines." Proceedings of the 27th international conference on machine learning (ICML-10). 2010.

[10] Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." The journal of machine learning research 15.1 (2014): 1929-1958.

[11] Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).