COMP3222 Machine Learning Technologies Coursework

Riccardo Andronache
ra2u18

1 Abstract

The information flow related to crisis/disasters should give credit to the user generated content on social media. To leverage the social media data effectively, it is crucial to filter out the noisy information from the big corpus researchers usually work with. Nowadays, information spreads very quickly on Twitter. This is a popular social networking service on which members create, post and interact with each other with messages known as **tweets**. This platform serves as a mean for individuals around the world, to express their feelings and knowledge about different ideas or subjects.

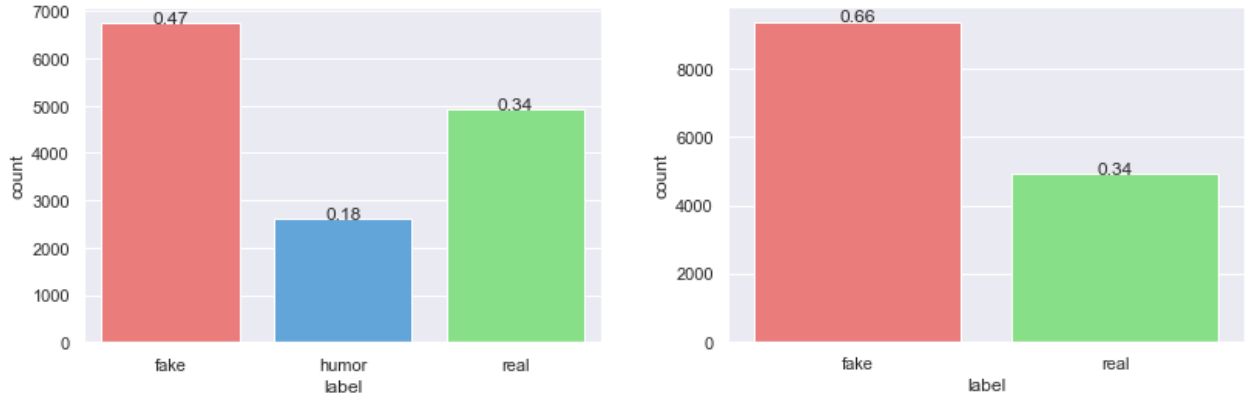
Moreover, since Twitter users post events in real time, various people interested in data analysis such as consumers and marketers have done sentiment analysis on tweets to gather insights for their specific goals. In this report, we will try to conduct sentiment analysis on tweets. The idea is to classify a tweet as being fake or real given the structure of the content text. We used simple machine learning classifiers at first: Logistic Regression, Embedding + double GRUs and then, on the third iteration, we scaled to BERT transfer learning. The customized BERT architecture is trained to compare with the simple models aforementioned. Results show that the customized BERT almost attain the best results. We could not delve deeper into optimizing BERT, because of low hardware resources (missed training bert on the pretrained bert-large-uncased).

2 Data description and analysis

The data provided by the University of Southampton comes in the form of comma separated values with the information: tweets and their corresponding label. Frankly, there are multiple columns, namely **tweetId**, **userId**, **imageId**, **username**, **timestamp**. For the purposes of this coursework they were dropped and the only data characteristic feature used is the text itself.

This will indeed jiggle down our final f-score, but it simplifies the problem for us.

After we load the data in pandas, we start analyzing it using a rigorous pipeline of steps. From now on every plot that contains green data refers to data with real label, whereas red data refers to fake labels. Firstly, let's look into the class distribution, see if the data is equally distributed along the unique classes.

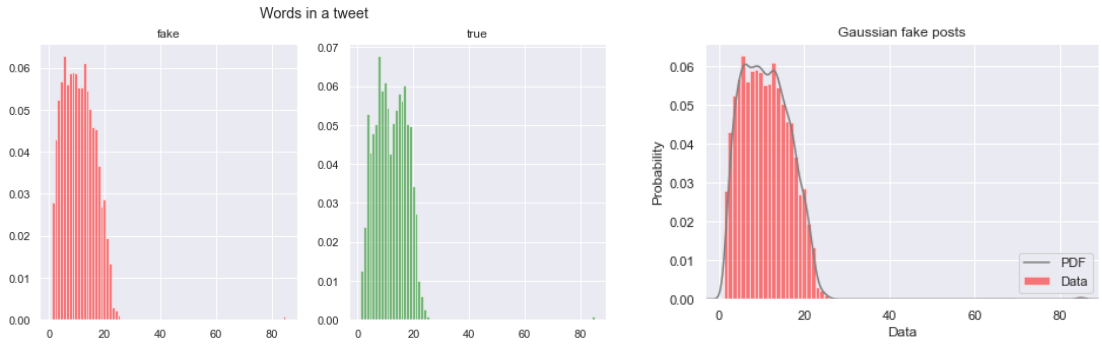


a) b)

Figure 1: *Data distribution of unique classes a): with humor ; b) replaced humor with fake*

Communities that deliver humorous content became popular lately, and the tweets that come from the content creators should not be considered real. They take events, in most of the cases non-real events, and create stories behind that are usually credible if not taken with a grain of salt. In the first iteration we tested the models on this ensemble of three classes. The results of the multi-class classification were not desirable, so for simplicy we considered all humorous tweets fake and used binary classification instead.

Another useful thing to do as a preprocessing step is to strip the outliers in the data. In our case an outlier could represent a very long tweet. **What does a long tweet mean?** We considered a tweet longer than 85 words to be "very long". If we strip the tweets larger than 85 words the distribution of the word length in a tweet almost follow a Gaussian Distribution.



a) b)

Figure 2: *a): Average words in tweet (fake, real) ; b) Gaussian pdf of average words in fake data*

Another signal that points towards fake events is punctuation and how it is used. We tried to plot the most common punctuation signs to get insight on how they are used and in which context. Is the punctuation of a fake tweet really different from that of a real one?

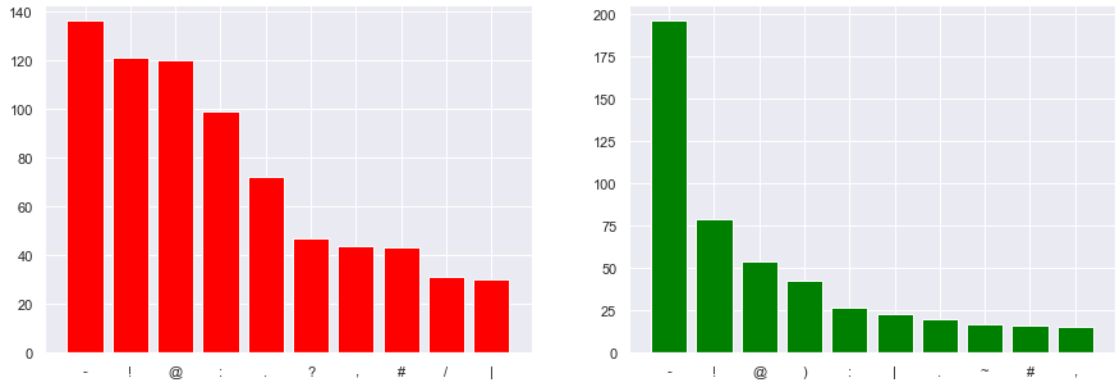


Figure 3: *Distribution of punctuation signs across fake and real data*

We all know that Twitter uses `@handle` to mention users. This is a handy way to attract the eye of the public and gather followers if done the *right way*. From Figure 3 we can see that most of the fake tweets use this feature to gain spotlight on the gigantic social media service. The exclamation mark is another indicative of the presence of a fake tweet. It may indicate astonishment or surprise. It is occasionally placed mid-sentence with a function similar to a comma, for dramatic effect.

What we have shown so far is just an insight of the data that has to be preprocessed. Raw tweets from Twitter generally result in a noisy data-set and this is due to the casual nature of people's usage of social media. They have certain characteristics such as retweets, user mentions, hashtags, emoticons (known as application features) which have to be carefully extracted into machine learning features - ideal for our final goal. Therefore, the noisy data-set has to be translated into a normalized, easy to read input for various classifiers. We have applied an extensive number of pre-processing steps to reduce its size and give it meaning; they can be summarized as follows.

- **Filter out non-english tweets**. This step was cumbersome, because there were some subtleties we found about only later on. The library used for this job is called `langdetect` and for each tweet it returns the language expressed as a two letter string, for example: `langdetect("Hello world")` outputs `-> en`, which stands for english. Using this library on raw tweets was not wise, because of the presence of hashtags. Even though half of the content of a random tweet was written in native spanish (let's say), the library detected english, thus keeping the respective tweet inside the normalized data frame. That was noisy data. In order to solve that issue, we have stripped the hashtags, emojis and links of the tweets and run the language detection process again. This indeed gave us a more accurate vector of languages that we then used as a mask for our dataframe.

- **URLs, User Mentions and Retweet**: Users often share hyperlinks to other evidence to enhance the credibility of their post. If we considered every single distinct URL as a feature, that would have led to a very sparse feature matrix. Therefore, we kept it simple and replaced the URLs with the token [URL] using the regex pattern `('https?://S+')` . Every user associated with a twitter account has a `handle`, that uniquely identifies them. To mention someone and get their attention, people use `@username` . This application feature allows them to gather attention from certified users, corporations in order to boost their number of followers. The repetitive abuse of this system could easily be one strategy to deceive

others with their "knowledge" into specific subjects. Therefore, we used this as a feature, but to avoid sparse matrices we replaced every @username with the token [USER_MENTION]. Lastly, retweets are tweets which have already been sent by others and shared further by other users. It begins with the letters RT and in our case it doesn't represent a useful feature, so we remove it.

- **Emoticons and emojis**: To easily express emotion or overall sentiment, users use a number of different emoticons or emojis in their tweets. With the help of some mappings found on GitHub[1] (emoticon, emoji -> sentiment) we matched every emoticon into its respective meaning. We wanted the classifier to be flexible in its final decisions, so instead of mapping emojis into EMO_POS or EMO_NEG, we mapped them into actual legit representation, for example statue_of_liberty for an emoji representing that.

- **Hashtags**: Another application feature used by users to mention trending topics or places around the world. We replaced them with the words after the hash in order to get a broader vocabulary.

3 Algorithm design

Machine learning is faced with one major hurdle, its algorithms usually deal with numbers that are later adjusted to fit the goal of the problem. Since we only have pre-processed text at our disposal, we should use something that accurately describes that text into numbers.

A simple and effective model for thinking about text documents in machine learning is called the BOW model. It throws away all of the order information and focuses on the occurrences of words in a document. This can be done by assigning each word a unique number, then any document can be encoded as a fixed-length vector with the length of the vocabulary of known words. Throughout this project we used the schemes described below.

3.1 Feature Extraction

These schemes are a statistical measure that evaluate how relevant a word is in a collection of documents. It enables us to associate each word in a document with a number that represents how relevant it is in that document.

- **TF-IDF**: In a large text corpus, some words will be very present (e.g. "the", "a", "is" in English) hence carrying very little meaningful information about the actual contents of the document. In order to re-weight the count features into floating point values suitable for usage by a classifier it is very common to use the tf–idf transform. If we were to feed the direct count data directly to a classifier those very frequent terms would shadow the frequencies of rarer yet more interesting terms. In our case, we removed the stop-words so this scheme is not going to influence the results that much.

- **CountVectorizer**: This method provides a simple way to both tokenize collections of text documents and build a vocabulary of known words and to encode new documents using that vocabulary. The representation of the encoded vector has the length of the entire vocabulary at each index we get the count for the number of times that word appeared in the document. Because the vectors will contain a lot of zeros, we call them sparse [2].

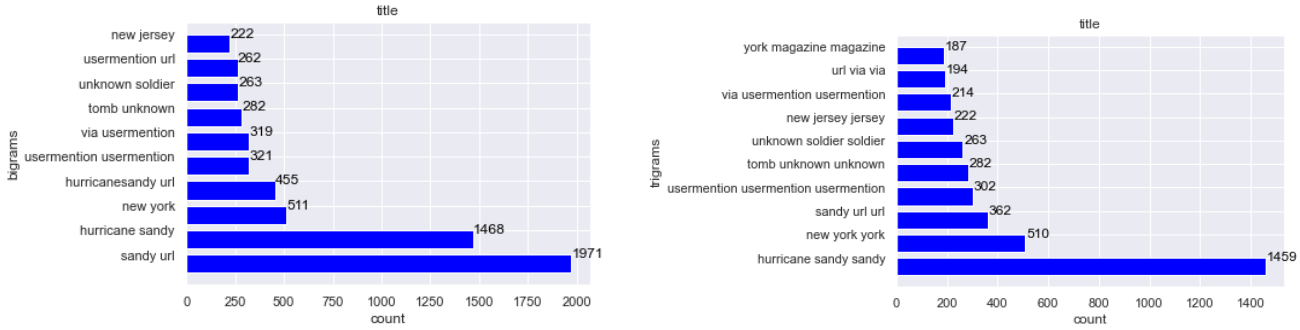
[1] https://github.com/NeelShah18/emot/blob/master/emot/emo_unicode.py

[2] Sparsity is advantageous in terms of computational efficiency. Unlike operations with full matrices, operations with sparse matrices do not perform unnecessary low-level arithmetic, which can lead to dramatic improvements in execution time for programs working with large amounts of sparse data.

The count vectorizer can tokenize tweets with respect to specific n-gram ranges. The size of the vocabulary fit on the train corpus considering unigrams, bigrams and trigrams is 61038. Which means the size of the sparse matrix will be (nb_tweets_train, 61038).

Unigrams are the simplest and most commonly used features for text classification. We extract single words from the training dataset and create a frequency distribution of these words. However, these are not enough to capture for example user satisfaction "very cool" or negation "not cool", which in this case expresses positive sentiment and negative sentiment respectively.

Bigrams are word pairs in the dataset which occur in succession in the corpus, they are a good way to model negation. Most of them are noisy, but since our dataset is not very large we use all of them in the vocabulary. Lastly, **trigrams** are good to capture phrases like "very very sad" which emphasize on the emotions the user has at the time of posting the tweet. Also, they can spot the presence of fake tweets that exploit the user mention feature of twitter "@username @username @username".



a) b)

Figure 4: *a): Top most bigrams and their count ; b) Top most trigrams and their count*

3.2 Logistic Regression, first iteration

In the first iteration we used Logistic Regression from `sklearn.linear_model` package. We left the penalty default of l2 and C equal to 1, the best values predicted by the grid search. We used sparse representation for classification and ran the model using both n-grams and word frequency feature types. We found that n-grams outperform the word frequency scheme. We also compared this model on the unbalanced data-set containing 66% fake posts and 34% real posts against the balanced data-set (under-sampling)

- Model scheme: **TF-IDF**, Data-set type: **unbalanced**.
  Log-loss of 0.341 and f-score of 0.806 on validation set, f-score of 0.2 on test set.
- Model scheme: **TF-IDF**, Data-set type: **balanced**. Log-loss of 0.359 and f-score of 0.904 on validation set, f-score of 0.43 on test set.
- Model scheme: **n-grams(1-3)**, Data-set type: **unbalanced**.
  Log-loss of 0.238 and f-score of 0.857 on validation set, f-score of 0.34 on test set.
- Model scheme: **n-grams(1-3)**, Data-set type: **balanced**.
  Log-loss of 0.221 and f-score of 0.908 on validation set, f-score of 0.51 on test set.

Having tested on all four possibilities the highest score was given by the n-gram(1-3) scheme together with the balanced data-set which was the result of under sampling the training data-set to have the same value count for the unique labels **fake** and **real**.

3.3 Double GRUs, second iteration

Recurrent neural network [4] have the ability to take variable length of text sequence but they are extremely tricky to learn. Hence new types of RNN were deployed like LSTM and GRU. GRU are much simpler in structure and probably more practical than LSTM. We found that GRUs were effective in the task of sentiment analysis because of their ability to remember long time dependencies.

In a quest to further improve accuracy and f score, we developed a sequential model comprising multiple layers. In this model, we use embeddings with dimension 50 to be representatives of each word and seed it with GloVe word vectors provided by the StanfordNLP group. The word embeddings are pretrained by StandfordNLP on different corpuses, but fine-tuned during model training to capture the context of data we are working with.

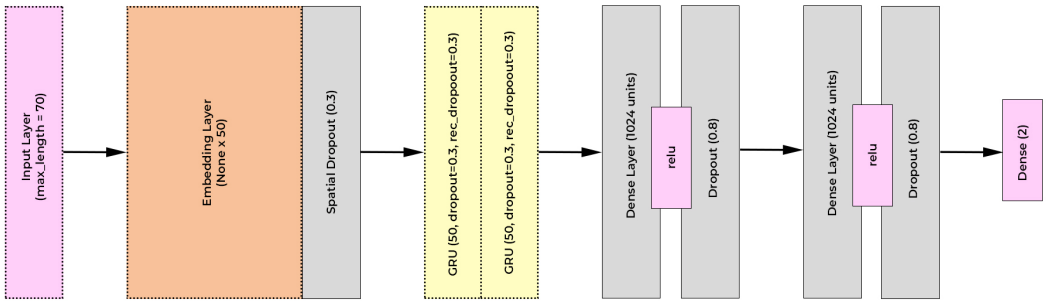


Figure 5: *Double GRUs model pipeline*

- **Non-random embedding initialization**: When using this model, we've tested between different pre-trained embedding feature sizes (50, 100, 200). The one that performed the best is `nb_feature=50` and the idea behind its success is that the model over-fitted later than the other two. Since the data-set used is small in dimension, having a large feature dimension encouraged the model to over-fit the training data and create a unique representation for each word in its given context. To spot this we have added a callback to monitor the validation loss with respect to the training loss

```
earlystop = EarlyStopping(monitor='val_loss', min_delta=0, patience=3, verbose=0, mode='auto')
```

The embedding layer is followed by a Spatial Dropout (0.3) [2] layer that will possibly drop some dimensions on every single embedding. The intuition behind this is to mask some words and learn to take correct decisions without those words, in other words to avoid over-fitting the embedding of those selected words. The output of the dropout layers are then connected to fully-connected Dense layers with `relu` activation functions, followed by `softmax` in the end. As usual, grid search pointed us to use Adam optimizer and categorical cross entropy for our loss.

Evaluation This model performed unexpectedly well with a **log-loss** of 0.403 and **f-score** of 0.821 on the validation set and an **f-score** of 0.6452 on the test set. The epochs run on the unbalanced data-set and we have chosen that data-set in order to preserve the amount of data we would have lost by doing under sampling. The total number of epochs desired were 100, but the early stopping condition stalled at 60, because the model started over-fitting.

3.4 BERT transfer learning [1], third iteration

Even though the f-score achieved in the second iteration was not bad, we wanted to see if we could achieve better evaluation metrics. In this iteration, BERT model is built based on the pytorch-pretrained-bert repository [3]. The customized BERT model is built upon bert-base-uncased model, which has 12 transformer layers, 12 self-attention heads, and with a hidden size 768. We wanted to go even further and try out bert-large-uncased, but we could not fit in memory even after rigorous optimization steps [see coments in train_bert.py].

Going back to analysis of the data... Even though this might seem annoying, another careful analysis on the tweets is required. Approaching the problem from another angle makes our the previous pre-processed data unreliable for the kind of input BERT models are expecting. For every tweet we need to append a [CLS] token in front and compute the length of the tokenized version of the tweet with respect to the config we choose for BERT: it can be either `bert-base-uncased` or `bert-large-uncased`.

We do keep the base pre-processing step, i.e texts are lowered, but we completely remove the retweet, user mention and URLs symbols. Texts with length less than 3 and greater than 85 are thrown away and duplicates are filtered out. No lemmatization or spell-checking is performed and no punctuation mark is removed since pre-trained embeddings are always used. No stop-words are removed for fluency purposes. After this data manipulation is done we end up with smaller datasets. For example, if the original test data-set contained 3755 candidates, after preprocessing for BERT we ended up with 1671 good candidates.

[3] https://github.com/huggingface/transformers

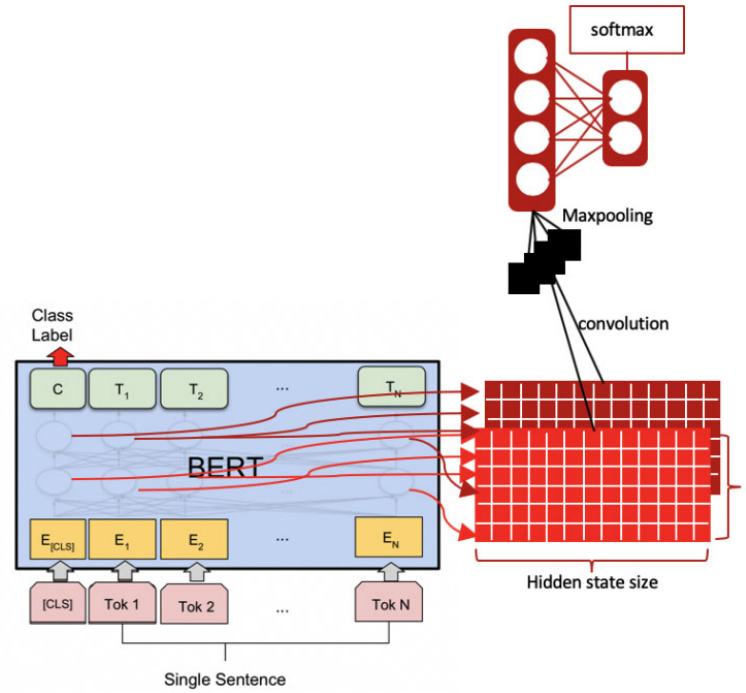


Figure 6: *Customized BERT + convolution (BERT+CNN)*

The model we implemented is **Customized BERT + convolution (BERT+CNN)** [3]. It utilizes hidden states from all the layers of BERT and for each layer a convolution is perform with 16 filters of size (3, hidden size of BERT). The output channels are concatenated together. We do a max-pooling to squeeze the convolution output into a 1D vector, which is then linked to a fully connected Dense layer to perform softmax.

Experimental details: Desired batch size is 32. Initial learning rate is set to 0.001 for non-BERT parameters and 0.00002 for BERT parameters as suggested by [https://arxiv.org/abs/1810.04805]. If the model doesn't get better after a patience of 5 (5 epochs), the learning rate decays by 50%. The maximum number of learning rate decay trials is 5 before we begin the process of earlystopping. We used Adam optimizer and cross entropy as our loss function.

Evaluation: The model performed quite well given the fact that we conducted a shallow preprocessing of the data. It achieved an **accuracy** of 0.788 and **f-score** of 0.615 on the test set.

References

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

[2] Yarin Gal and Zoubin Ghahramani. "A theoretically grounded application of dropout in recurrent neural networks". In: *Advances in neural information processing systems* 29 (2016), pp. 1019–1027.

[3] Guoqin Ma. "Tweets Classification with BERT in the Field of Disaster Management". In: ().

[4] Xingyou Wang, Weijie Jiang, and Zhiyong Luo. “Combination of convolutional and recurrent neural network for sentiment analysis of short texts”. In: *Proceedings of COLING 2016, the 26th international conference on computational linguistics: Technical papers*. 2016, pp. 2428–2437.