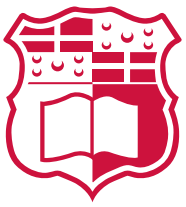

L-Università ta' Malta

LIN3012 - Assignment

Project 4: Multilingual Emoji Prediction

Daniel Magro

February 2019

Introduction

Since around the 90s [1], Emojis have become a core component of modern, casual computer mediated communication (CMC) [2]. Emojis are "punctuation-based renditions of facial expressions, objects and symbols, e.g., :-) or :-P)" or more recently, "pictographic" icons or symbols [2].

The aim of this work is to solve the problem of predicting what tweet an individual may use given a body of text, particularly Tweets.

Research suggests that what emoji an individual uses are linked to their personality traits [2].

With such a solution, if applied to, say, a customer service chat bot, it may make users feel as if they are communicating with a real individual with emotions, rather than a bot.

Furthermore, perhaps it is not so far-fetched to say that by correctly predicting an individual's emojis, elements of their personality can be inferred.

It is also useful to be able to understand what purpose tweets serve in text, especially in social media posts, to better understand the sentiment of that text, as tweets are often used in place of words [2], and may hold relevant information.

Literature Review

Barbieri et al. discuss the implementation details of the top ranking submissions. The submissions range from classical approaches such as SVMs and

Random Forests, to more modern approaches such as Neural Network (NN) based solutions [3].

The top ranking solution for both English and Spanish was 'Tübingen-Oslo'. This solution made use of an "SVM classifier with bag-of-n-grams features (both characters and words)" [3][4]. All other solutions that beat the baseline for Spanish used a similar, classical solution.

The runner-up solution was 'NTUA-SLP', that made use of a BiDirectional LSTM with attention over a pre-trained embedding layer [3][5]. 'EmoNLP' also used a BiDirectional LSTM, however this time over character and word n-grams rather than an embedding layer [3].

As has been discussed, the majority of top ranking solutions have used 'classical' approaches such as SVMs and Random Forests. None of the NN based solutions beat the Base Line for Spanish. In another similar task, Barbieri et al. use a BiDirectional LSTM, with a dataset of 40 million tweets, and obtain very good results, beating human annotators [6].

All this is good motivation to attempt to improve the results obtained using Neural Networks, which are a more contemporary, and often more effective, approach to most NLP problems.

Data

The data for this task was downloaded from the original Shared Task Page, `https://competitions.codalab.org/competitions/17344#learn_the_details-data` [3].
The downloaded zip file contained three directories, 'train', 'trial' and 'test'.

The 'trial' directory contains a .text file of 50,000 English (US) tweets, along with a .labels file of 50,000 labels, and a .text file of 10,000 Spanish (ES) tweets, again with an accompanying .labels file with 10,000 labels. These (tweet, label) pairs are meant to be used as a validation/development set, i.e. to determine whether the model is overfitting, and to fine tune hyper parameters.

Similarly, the 'test' directory contains 50,000 English (US) and 10,000 Spanish (ES) (tweet, label) pairs, to be used solely for evaluation.

The 'train' directory does not contain any tweets or labels, as these need to be downloaded through Twitter's API due to Twitter Restrictions.
To do this, a Twitter developer account was created on `https://developer.twitter.com/en.html`. Upon creation of a developer account, users are provided with API keys and Access tokens.
These are entered into the 'crawler.properties' file, which specifies to the crawler with which API keys and access tokens to download the tweets, which tweet IDs to download (US or ES tweets), and where to download the tweets to. Once the properties file was set, the crawler was run from a Linux

Command Line using the following command:

```
java -cp './twitter-crawler-0.4.jar:./lib/*' org.backingdata.twitter.crawler
.rest.TwitterRESTTweetIDlistCrawler crawler.properties
```

This starts downloading the tweets specified in the 'train_us/es_semeval18.ids' file (as set in the 'crawler.properties' file) in a .json format. The same command is run again for the es tweets after changing 'crawler.properties' to specify the es tweets .ids file.

To convert the downloaded tweets from a .json format to a .text file and an associated .labels file, i.e. the same format as the trial and test tweets, the following commands were run from a Python 2.7 environment:

```
python emoji_extractor_semeval18.py ./crawler\data\us\tweet_by_ID_28_1_2019__06_28_21
.txt us
python emoji_extractor_semeval18.py ./crawler\data\es\tweet_by_ID_04_2_2019__03_18_24
.txt es
```

Out of 500,000 US training tweet IDs, 454,020 were successfully downloaded and converted.

Out of 100,000 ES training tweet IDs, 91,050 were successfully downloaded and converted.

The US data is split as follows:

454,020 training tweets : 50,000 validation tweets : 50,000 test tweets

The ES data is split as follows:

91,050 training tweets : 10,000 validation tweets : 10,000 test tweets

= 82% training : 9% validation : 9% test

How this data was preprocessed (cleaned, indexed etc.) is discussed in the

following ‘Methodology’ section.

Methodology

The developed program first defines a list of hyper parameters, these describe the following:

- 0 - whether to remove stop words
- 1 - whether to stem words
- 2 - the size of the embedding vector
- 3 - whether to use a bidirectional RNN over the embedding
- 4 - the vector size representing the input sequence
- 5 - whether to use a SimpleRNN, LSTM or GRU
- 6 - whether to use a second RNN after the first
- 7 - the vector size of the second RNN
- 8 - rate of dropout to use - float between 0 and 1
- 9 - number of epochs to train for
- 10 - batch size to use while training

These hyper parameters are listed at the very beginning of the program, such that they can be easily edited from one place and can be referred to from anywhere in the script to determine how certain things will be done. How each hyper parameter affects the model is described below.

First the training set is loaded. The training set text file has an individual

tweet on each line. The training set is thus stored as a List of tweets. Each tweet is then stored as a List of tokens or 'words', where each token is a string. The individual tokens in a tweet are obtained by splitting a tweet by white spaces. So the training data is stored as a List of Lists of Strings.
The labels of the training set are also loaded. These are also stored as a label on each line, where each label is stored as a number between 0 and 19. Each label corresponds to the tweet in the same position in the training text file. The labels are read and stored as a List of Labels, so the data structure is a List of Ints.

The trial (validation) data is loaded in the same manner, and stored in identical data structures.

A function 'clean_data' is defined, which given the data sets will return a cleaned version of the dataset. This function will:
Add an 'EDGE' token at the beginning of each tweet. If hyper parameter 3 (use_bidirectional) is set to true, add an 'EDGE' token to the end of the tweet too. Change each token to lower case. If the 'remove_stopwords' hyper parameter is set to True, remove stop words from the tweet (eg. 'the', 'a', 'is' etc.). If the 'stem_tokens' hyper parameter is set to True, tokens will be stemmed using the Snowball Stemmer for the appropriate language.

A function 'get_vocab' is defined, which given the cleaned training data, will return a vocabulary, i.e. a list of words recognised by the model (with some minimum frequency of occurrence), plus some keywords (PAD, EDGE and UNKNOWN).

A dictionary 'token2index' is initialised which maps any given token to an index. A 'defaultdict' is used so that when looking tokens which are not in the key set (vocab), instead of throwing an error, the dictionary returns the index of the UNKNOWN token, which is 2.

A function 'index_data' is defined, which given a cleaned data set of tweets, will return the same data set of tweets, but with indexes instead of tokens. Thus, a dataset of tweets is converted from a List of Lists of Strings (tokens) to a List of Lists of Ints (indexes).

Finally, one last data function is defined, 'pad_data'. This function converts a List of indexed tweets into a **numpy array** of indexes. Since not all tweets are of the same length (i.e. do not have the same number of tokens), they need to be padded. A numpy array is created with as many rows as there are tweets, and as many columns as there are tokens in the longest tweet. When a tweet has less tokens than the width of the array, the remaining cells are filled with 0, the index for the 'PAD' token.
It is important for data sets to be stored as numpy arrays as this makes them compatible with keras.

All tweet data sets are cleaned, indexed and padded before they are used with the model.

The function 'build_neural_net' builds the Neural Network based on the hyper parameters described above.
The first layer is the input layer, this has as many nodes as there are tokens in the vocab.

This is connected to the embedding layer, which has as many nodes as is specified by the hyper parameter *embedding_vector_size*. This layer builds a representation of a word's meaning.

The second layer is a Recursive layer. This layer can be a Simple RNN, an LSTM or a GRU, depending on the value of the hyper parameter 'use_SRNN_LSTM_GRU'. This layer goes over the input from start to end, or possibly also from end to start (in both directions), depending on the value of 'use_bidirectional'. The number of nodes in this layer will be as defined by the hyper parameter 'input_vector_representation_size'. This layer also uses dropout at the rate defined by the hyper parameter 'dropout'.

The hyper parameter 'use_second_RNN' defines whether another RNN is used over the intermediate output of the first RNN. If this hyper parameter is set to True, the first RNN will also 'return_sequences', meaning that along with its output, it will also "return the hidden state output for each input time step" [7].

The last layer is a softmax layer. This layer will contain one neuron per possible label/classification (20 for US, 19 for ES). Each neuron will output the model's confidence that the correct label is that neuron's corresponding label.

Once the model is built, it is compiled using the 'adam' optimiser and 'sparse_categorical_crossentropy' as a loss function.

The model is then fit to the cleaned, indexed and padded training data for the number of epochs specified in hyper parameter 9, with a batch size as defined in hyper parameter 10 and the trial data as validation data.

The loss over the training data and validation data is plotted, such that overfitting can be detected, as well to get an indication on how well the model is generalising.

Finally, once the model is trained, the cleaned, indexed and padded test data is passed to the model in feed forward mode, and the output is recorded in a List of Lists of floats (Each element in the outer list is the softmax output for a test instance, and each float in the inner List is a neuron's output for a particular label.)
The output is then converted to a List of Ints, by choosing which label in the list of floats obtained the highest score.
Once the final output is obtained, this is passed to the provided Evaluation Script for evaluation of the model's performance. The workings of this script are discussed in the 'Results' section.

In order to further optimise the values of the hyper parameters used, the Python package scikit-optimize (skopt) is used. After the accepted values for each hyper parameter are defined, Random Forest is used to explore how different combinations of hyper parameters affect the performance of the model.

Results

The official evaluation script provided for the task was used to evaluate the performance of a trained model.

Since there is a skew in the distribution of data, i.e. some emojis appear much more frequently than others, the script computes the **macro average** F1 Score (official scoring metric), precision, recall and accuracy. In other words, a confusion matrix is built for each label, and the results for each metric across all classes are averaged with equal weighting [3].

The performance of the trained models were compared to those of other participants, which can be seen in Figure 1.

ENGLISH					**SPANISH**				
Team	F1	Prec.	Recall	Acc.	Team	F1	Prec.	Recall	Acc.
Tübingen-Oslo	**35.99**	36.55	36.22	47.09	Tübingen-Oslo	**22.36**	**23.49**	**22.80**	**37.27**
NTUA-SLP	35.36	34.53	**38.00**	44.74	Hatching Chick	18.73	20.66	19.16	37.23
hgsgnlp	34.02	35	33.57	45.55	UMDuluth-CS8761	18.18	19.02	18.6	34.83
EmoNLP	33.67	39.43	33.7	**47.46**	TAJJEB	17.08	18.99	20.36	25.13
ECNU	33.35	35.17	33.11	46.3	Duluth UROP	16.75	17.11	18.1	28.51
UMDuluth-CS8761	31.83	**39.80**	31.37	45.73	BASELINE	16.72	16.84	17.52	31.63
BASELINE	30.98	30.34	33	42.56	Nova	16.7	17.2	17.07	26.50
THU_NGN	30.25	31.85	29.81	42.18	ECNU	16.41	16.91	16.48	30.82
TAJJEB	30.13	29.91	33.02	38.09	MMU - Computing	16.34	17.83	16.4	28.92
EmojiIt	29.5	35.17	29.91	39.21	PickleTeam!	15.86	17.57	16.76	29.70

Figure 1: Top ranking participant scores for the official task [3]

The trained model for English (US) obtained an F1 score of 31.708. This is a respectable score given that it surpassed the baseline, which is itself a good score.

The trained model for Spanish (ES) obtained an F1 score of 16.91. This is a very satisfactory score. Not only does it come in fifth, but it is also the only NN based approach to beat the base line for Spanish.

Barbieri et al. note that the majority of high ranking solutions for one language, perform poorly in the other [3]. This solution has shown to obtain

good results for both languages. It may well be due to the minimal preprocessing that is carried out, as suggested by [3].

When looking at the performance of the models in a qualitative manner, most incorrectly classified instances are still classified as very similar emojis, that convey the same sentiment. Two examples of this are Figures 2 and 3 for English and Spanish respectively.
Very few incorrect labels are far off. Two instances where the label is more far off can be seen in Figures 4 and 5.

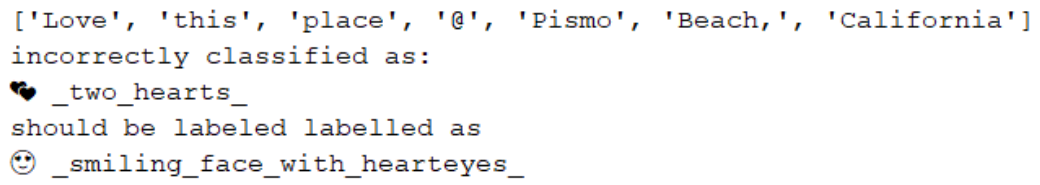

Figure 2: Example of an incorrect label, but close guess in English

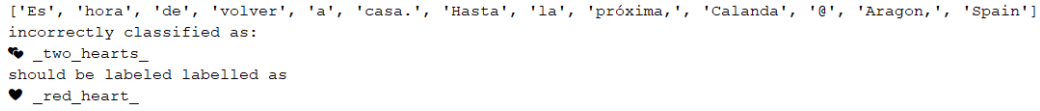

Figure 3: Example of an incorrect label, but close guess in Spanish

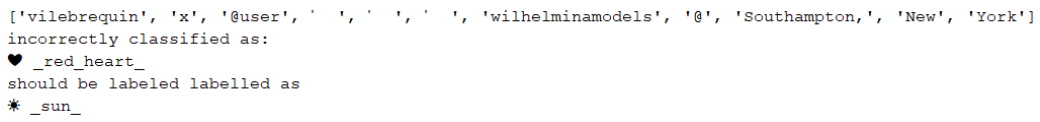

Figure 4: Example of an incorrect label in English

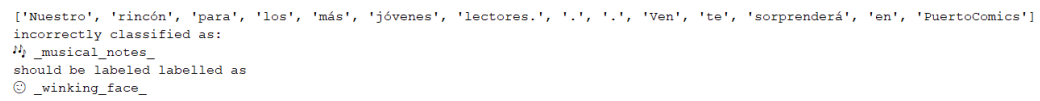

Figure 5: Example of an incorrect label in Spanish

Discussion and Conclusions

Even though with the resources and time available, proper hyper parameter optimisation could not be carried out (Around 200 hyper parameter combinations were tested over 3 days on one laptop), most initial predictions on which values would produce the best score were correct.

Models generally scored better for this task without removal of stop words and stemming, more broadly, with minimal text preprocessing. This is confirmed by [3].

A BiDirectional RNN works much better than a single direction/pass RNN. This is also confirmed by [6].

A LSTM RNN works better for this task than a simple RNN or a GRU. This was again confirmed by [6].

It would be interesting to observe how this model could be improved by using pre-trained embedding layers, instead of training one from scratch, as suggested by Baziotis et al. [5].

References

[1] C. Bonnington, "Emoji," *Wired*, vol. 21, p. 60, 05 2013.

[2] D. Marengo, F. Giannotta, and M. Settanni, "Assessing personality using emoji: An exploratory study," *Personality and Individual Differences*, vol. 112, pp. 74 – 78, 2017.

[3] F. Barbieri, J. Camacho-Collados, F. Ronzano, L. Espinosa-Anke, M. Ballesteros, V. Basile, V. Patti, and H. Saggion, "SemEval-2018 Task 2: Multilingual Emoji Prediction," in *Proceedings of the 12th International Workshop on Semantic Evaluation (SemEval-2018)*, (New Orleans, LA, United States), Association for Computational Linguistics, 2018.

[4] C. Coltekin and T. Rama, "Tubingen-oslo at semeval-2018 task 2: Svms perform better than rnns at emoji prediction," 04 2018.

[5] C. Baziotis, N. Athanasiou, G. Paraskevopoulos, N. Ellinas, A. Kolovou, and A. Potamianos, "Ntua-slp at semeval-2018 task 2: Predicting emojis using rnns with context-aware attention," 04 2018.

[6] F. Barbieri, M. Ballesteros, and H. Saggion, "Are emojis predictable?," *CoRR*, vol. abs/1702.07285, 2017.

[7] J. Brownlee, "Understand the difference between return sequences and return states for lstms in keras," Oct 2017.