INGEOTEC at IberEval 2018 Task HaHa: μTC and EvoMSA to Detect and Score Humor in Texts

José Ortiz-Bejar[1,3], Vladimir Salgado[3], Mario Graff[2,3], Daniela Moctezuma[3,4], Sabino Miranda-Jiménez[2,3], and Eric S. Tellez[2,3]

[1] Universidad Michoacana de San Nicolás de Hidalgo, México
jortiz@umich.mx
[2] CONACyT Consejo Nacional de Ciencia y Tecnología, Dirección de Cátedras, México
[3] INFOTEC Centro de Investigación e Innovación en Tecnologías de la Información y Comunicación, México
{vladimir.salgado,mario.graff,sabino.miranda,eric.tellez}@infotec.mx
[4] Centro de Investigación en Ciencias de Información Geoespacial A.C., México
daniela.moctezuma@centrogeo.edu.mx

Abstract. This paper describes our participation in *Humor Analysis based on Human Annotation (HAHA)* task on IberEval'2018. The classification task is tackled using our previous work on creating a multilingual sentiment analysis classifier (EvoMSA) and our generic text categorization and regression system (μTC) as a solution for the regression task.

Keywords: Sentiment Analysis · Text Categorization · Genetic Programming.

1 Introduction

The use of humor for communicating ideas is a human resource that can have a multitude of meanings and forms. An idea can be explicitly expressed to be fun, or it can be found to be humorous after a long conscientious reflexion. It is also possible to understand that something is funny because of happiness or sadness. Moreover, the humorous can be constructed based on truth or falseness, or it can be subtle or cynic. Finding humor in some situation is many times a consequence of personal and social experiences, language variations, culture, etc.

Learning to detect humor through a machine learning supervised approach based on labeled examples of what is a joke or not, is pretty hard, due to the complexities above mentioned, i.e., not even humans have a clear consensus of what is humorous or its *degree of fun*. However, based on an extensive enough knowledge database, a proper model of the text, and a learning algorithm, the identification process can be tackled more or less effectively.

To find solutions about this area, IberEval-2018 forum ran a task named Humor Analysis based on Human Annotation (HAHA) where a set of human-labeled messages from Twitter are provided to train and test algorithms for humor identification (classification) or ranking (regression). More detailed, each text is labeled as humorous or not humorous; a score of the humor-intensity is also given to define a rank problem.

In this paper, our solution to solve this problem is described. This paper is organized as follows, in Section an agnostic approach to tackle humor detection is explained, in Section 2 the task is described. Proposed solution is presented in Section 3. All the results and experimental methodology are discussed in Section , and finally Section 5 concludes.

1.1 An generic approach to humor detection

Let us introduce some necessary notation before we dive into the main discussion. Let T be the set of all texts, and t_i will refer to some $t_i \in T$. Let Θ be the set of labels for each θ_i and is defined over $\{0,1\}$, such that $\theta_i = 0$ implies that t_i is not humorous expression, while $\theta_j = 1$ entails that t_j is humorous. The set of real values in $[0,5]$ is named as Y, and each $y_i \in Y$ is the average funniness for each $t_i \in T$. Finally, let X be a vector space related to the text T.

Figure 1 illustrates our generic supervised model for humor classification and regression. The process starts with the set T and its associated Θ; then, the idea is to create a vector space X that will be used to train a classifier.

The vector space is created, firstly, normalizing and transforming the text; secondly, the processed text is tokenized using multiple schemes like word n-grams, character q-grams, and skip-grams; and, this bag of tokens is vectorized through a weighting scheme. Finally, T is transformed into the vector space to train a classifier, along with labels Θ.

A similar process is necessary to create a regressor, using Y instead of Θ. The model's quality depends on the entire pipeline. The entire process is documented in [11].

Figure 1 describes the work-flow of the training process. The prediction process is almost the same, that is, the unknown text t_q has to be transformed into a vector x_q by using the same preprocessing, tokenization, and term weighting steps, such that the classifier observes the new vector in the same space of the training set. Again, the procedure is quite similar for regression analysis.

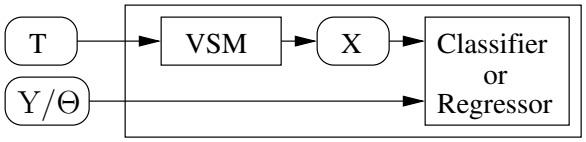

Fig. 1. The generic diagram for our humor classification and regression system.

There are available many tools to model texts, for example, *gensim* [9], *nltk* [1], *fastText* [6]. The main drawback for these approaches is that all of them have a set of parameters that should be tunned and wrong parameters selection may lead to generate features (vectors) of low quality, resulting this in a poor performance for classification/regression tasks. As exploring all possible parameters combination is prohibitive and finding a good parameter set of values is highly dependent of the problem, one simple approach is to perform a search over parameter space and use the parameters with the best performance over a cross-validation simulation for a particular dataset.

2 Task Description

Humor Analysis based on Human Annotation (HAHA) asks for systems that classify tweets, in the Spanish language, as humorous or not. Also, it asks for systems that determine (rank) how funny the tweets are. Those two tasks are described by *HAHA* organizers as follows:

Humor detection: determining if a tweet is a joke or not (intended humor by the author or not). The results of this task will be measured using F-measure for the humorous category and accuracy. F-measure is the primary measure for this task.

Funniness score prediction: predicting a funniness score value (average stars) for a tweet in a 5-star ranking, supposing it is a joke. The results of this task will be measured using root-mean-squared error (RMSE).

The first task can be solved as a classification problem, while the second one can be tackled as a regression problem. The following sections describe μTC and *fastText* and how we use these tools for our solution.

In this task, the training set provided consists of a corpus of 20000 crowd-annotated tweets, as described in [2], divided into 16000 tweets for training and 4000 tweets for the test dataset. Multiple annotators evaluated each tweet, and each annotation consists of the class (humorous or not) and the intensity (number of stars 0-5). The final label is determined using a voting scheme. Table 2 shows an example of the content of the provided dataset.

Table 1. Humorous tweet example

Text	La semana pasada mi hijo hizo un triple salto mortal desde 20 metros de altura. Es trapecista? - Era :(
Is humorous	True
Average stars	3.25

For task one *text is humorous* corresponds to the set of labels Θ, while *average starts* is used as Y.

3 Systems Description

Our best solutions are mainly based on following algorithms; μTC, a set of well-known classifiers from scikit-learn [8] (Naive Bayes, Support Vector Machine and NearestCentroid), several regressors also from scikit-learn (Kernel Ridge, Ridge, Ada Boost, Decision trees and ElasticNet), B4MSA and EvoDAG [4], but also we explored the use of *fastText* as classifier. In the following Sections, we describe several approaches in more detail.

3.1 μTC

μTC [11] is a minimalistic and powerful library that generates text models maximizing a performance measurement. It manages the entire pipeline of a text classifier, as specified in §1.1. Under the hood, μTC uses a Support Vector Machine with a linear kernel

as the classifier. The core idea behind μTC is to define a parameter space describing a massive number of text-classifiers. The problem is posed as a combinatorial problem, and an efficient set of meta-heuristics are used to find very competitive solutions.

3.2 EvoDAG and EvoMSA

Evolving Directed Acyclic Graph (EvoDAG) is a steady-state Genetic Programming system with tournament selection [3, 4]. The main characteristic of EvoDAG is that the genetic operation is performed at the root. EvoDAG was inspired by the geometric semantic crossover proposed in [7]. EvoMSA uses an EvoDAG classifier to perform text classification, this approach is robust in problems with unbalanced classes.

3.3 B4MSA

The baseline algorithm for multilingual sentiment analysis (B4MSA) [10] is a sentiment classifier for informal text such as Twitter messages. The design is similar to μTC, but the internal problem is solved differently, along with the use of specific features for sentiment analysis and some language-dependent capabilities.

3.4 FastText

FastText [5] is a library for text classification and word vector representation. It transforms text into continuous vectors that can later be used on any language related task. FastText represents sentences with a weighted bag of words, and each word is represented as a bag of character n-gram to create text vectors. This representation is based on the skip-gram model [6] which take into account subword information and sharing information across classes through a hidden representation. Also, it employs a hierarchical softmax classifier that takes advantage of the unbalanced distribution of the classes to speed up computation.

As μTC, we optimized many of the parameters of *fastText* along with the applied preprocessing functions. We used random search over a state space for this purpose.

4 Experiments and results

We tested multiple approaches by using the tools above described. The experimental setup consisted of using the set T of 16000 tweets human annotated by the task organizers. Firstly, T was split in training (T_t) and validation (T_v) sets following a 80-20 proportion. Organizer provided data in *CSV* format, so we need to convert them to the native formats of μTC and EvoMSA; Appendix A detail system usage and required formats.

4.1 Classification task

Our first intent was to use μTC and FastText with its default parameters. Further improvements were achieved by optimizing FastText parameters and using a Naive Bayes classifier with μTC. We used fastText with default parameters, nevertheless

after optimizing the learning rate (lr), vector dimension (dim), size of word n-grams (wordNgrams), window size (ws), and number of epochs (epoch), we obtained better performance. The default values for these parameters were $lr = 0.1$, $dim = 100$, $wordNgrams = 1$, $ws = 3$ and $epoch = 5$ and the ones found by optimization process were $lr = 0.2$, $dim = 300$, $wordNgrams = 5$, $ws = 5$, $epoch = 35$. However, the best results were reached with EvoMSA. All results shown in Table 2 were the ones obtained over T_v.

Table 2. Performance of the different systems on the validation set.

System	Macro-F1	Macro-Recall	Accuracy	F1
EvoMSA	**0.8440**	**0.8454**	**0.8553**	**0.8021**
μTC NaiveBayes (macrof1)	0.8342	0.8268	0.8506	0.7821
μTC (LSVM)	0.8238	0.8237	0.8372	0.7751
FastText (optimized parameters)	0.7976	0.7840	0.8241	0.7244
FastText (default parameters)	0.6673	0.6604	0.7209	0.5337

Only results for EvoMSA and μTC with Naive Bayes were submitted to the contest score system (best result). Table 3 shows a summary of the performance of the top three participants as well as the baselines set by the organizes. The best result corresponds to the model generated with EvoMSA. It is relevant to mention that F1 score was used to rank the participants.

Table 3. Performance on the test set

Team	Accuracy	Precision	Recall	F1
INGEOTEC	0.8452	0.7796	**0.8157**	**0.7972**
UO_UPV	**0.8455**	0.8158	0.7567	0.7851
ELiRF-UPV	0.8367	0.8046	0.7426	0.7724
baseline	0.4915	0.3645	0.4886	0.4175
baseline	0.6595	**0.9392**	0.0932	0.1695

4.2 Regression

For the second subtask, we tested all the regression algorithms in scikit-learn which support sparse vectors over the space vector X generated by μTC. The regressors were trained with X_t and validated at X_v. As the average stars for the test set were unknown and the organizers state that *...for task 2, it is important that all rows have a predicted score. The scoring algorithm will check the ones that were appropriate for evaluation*; it was necessary to guess how RMSE scores were calculated. Three scenarios were assumed: Firstly, RMSE was calculated over **all** predicted scores (using all validation set); secondly, score value was calculated only for the samples labeled (**real**) as humorous; thirdly, RMSE was calculated for tweets **predicted** as humorous. Table 4 shows the scores for the validation set T_v where the more stable over the three consider cases were Ridge regressors.

As Kernel Ridge regression exhibited best average RMSE, it was used to predict average stars scores over the test set. Table 5 shows the contest's results.

Table 4. Regressors performance over validation set

Regressor	RMSE(all)	RMSE(real)	RMSE(predicted)	Average
Kernel Ridge	**0.8255**	**0.9865**	0.9816	**0.9312**
Ridge	0.8331	0.9941	0.9747	0.9339
Random Forest	0.8406	0.9947	1.0813	0.9722
Ada Boost	1.2181	0.9899	**0.8449**	1.0176
Decision Tree	1.1487	1.3421	1.3822	1.2910
ElasticNet	1.1049	1.4555	1.5220	1.3608
SGD	1.0597	1.4936	1.6100	1.3878

Table 5. Performance on the test set

Team	RMSE
INGEOTEC	**0.9784**
baseline	1.1419
UO_UPV	1.5919

5 Conclusions

This paper describes the performance of the INGEOTEC team at HAHA'18, to the best of our knowledge, the first humor analysis in the Spanish language (Mexican-region). Our approach consists of well-tuned μTC and EvoMSA models to perform both classification and regression tasks. Moreover, we include an appendix as a guide to replicate our results.

References

1. Bird, S., Klein, E., Loper, E.: Natural language processing with Python: analyzing text with the natural language toolkit. " O'Reilly Media, Inc." (2009)
2. Castro, S., Chiruzzo, L., Rosá, A., Garat, D., Moncecchi, G.: A crowd-annotated spanish corpus for humor analysis. In: Proceedings of SocialNLP 2018, The 6th International Workshop on Natural Language Processing for Social Media (2018)
3. Graff, M., Tellez, E., Escalante, H., Miranda-Jiménez, S.: Semantic genetic programming for sentiment analysis, vol. 663 (2017). https://doi.org/10.1007/978-3-319-44003-3_2
4. Graff, M., Tellez, E., Miranda-Jiménez, S., Escalante, H.: EvoDAG: A semantic Genetic Programming Python library. In: 2016 IEEE International Autumn Meeting on Power, Electronics and Computing, ROPEC 2016 (2017). https://doi.org/10.1109/ROPEC.2016.7830633
5. Joulin, A., Grave, E., Bojanowski, P., Mikolov, T.: Bag of tricks for efficient text classification. In: Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers. pp. 427–431. Association for Computational Linguistics (April 2017)
6. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in neural information processing systems. pp. 3111–3119 (2013)
7. Moraglio, A., Krawiec, K., Johnson, C.G.: Geometric semantic genetic programming. In: International Conference on Parallel Problem Solving from Nature. pp. 21–31. Springer (2012)

8. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
9. Řehůřek, R., Sojka, P.: Software Framework for Topic Modelling with Large Corpora. In: Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks. pp. 45–50. ELRA, Valletta, Malta (May 2010), http://is.muni.cz/publication/884893/en
10. Tellez, E.S., Miranda-Jiménez, S., Graff, M., Moctezuma, D., Suárez, R.R., Siordia, O.S.: A simple approach to multilingual polarity classification in Twitter. Pattern Recognition Letters **94**, 68–74 (2017). https://doi.org/10.1016/j.patrec.2017.05.024
11. Tellez, E., Moctezuma, D., Miranda-Jiménez, S., Graff, M.: An automated text categorization framework based on hyperparameter optimization. Knowledge-Based Systems (2018). https://doi.org/10.1016/j.knosys.2018.03.003

A μTC and EvoMSA quick start guide

As we have mentioned earlier, our systems are publicly available, developed in Python and to facilitate their use there is a command line interface (CLI).

The format used for the datasets, e.g., training set, is json per line, e.g., { "klass" = 0, "text": "good life" } where *klass* contains the label and *text* is the text to be classified.

A.1 EvoMSA

EvoMSA can be installed from different sources; however, the most accessible path to install it is using conda with the following command:

```
conda install -c ingeotec evomsa
```

The first step in EvoMSA is to create the model, this is achieved with the following command:

```
EvoMSA-train -n2 -o evomsa.model train.json
```

where *-n2* indicates to two cores, *-o* specifies the model's name, and train.json contains the training set.

Once the model is created, it can be used to predict unseen instances with the following command:

```
EvoMSA-predict -n1 -o out.json -m evomsa.model test.json
```

where *-n1* indicates to use one core, if it is omitted then the number of cores used in training is used instead, *-o* specifies the output file, m is the model, and *test.json* contains the file to be predicted.

A.2 μTC

μTC is intentionally simple, so only a small number of features where implemented. The number of dependencies is limited and fulfilled by almost any Scientific Python distributions, e.g., Anaconda.

MicroTC can be easily installed in almost scientific python distribution.

```
git clone https://github.com/INGEOTEC/microTC.git
cd microTC
python setup.py install --user
```

It supposes that `git` is installed in the system. If it is not available, it can be installed using `apt-get`, `yum`, or downloading the latest version directly from the repository.

For any given text classification task, μTC will try to find the best text model from all possible models as defined in the configuration space.

```
microTC-params -k3  -Smacrof1 -s24 -n24 train-haha.json -o vsm.params
```

these parameters means:

- `train-haha.json` is database for HAHA as one json-dictionary per line with text and klass keywords
- -k3: three folds
- -s24: specifies that the parameter space should be sampled in 24 points and then get the best among them, i.e. the sample size of an internal random search.
- -n24: let us specify the number of processes to be launch.
- -o: `vsm.params` specifies the file to store the configurations found by the parameter selection process, in best first order.
- -S or –score: the name of the fitness function.
- -H: indicates that a hill climbing search will be performed over the best result found by random search.

These parameters have default values, such as no arguments are needed. The interested reader is referred to the μTC page https://github.com/INGEOTEC/microtc.

Once a set of parameters is found the dataset `train-haha.json`, and the parameters in `vsm.params` can be used to train a model and save it in mtc.model using the following command:

```
microtc-train -o mtc.model -m vsm.params train-haha.json
```

the resulting model can be tested (i.e., `test-haha.model`) in a new test set. That is, we can ask the classifier to label some database as follows:

```
microtc-predict -m mtc.model -o test-predicted.json test-haha.json
```

Finally, the prediction performance is computed with the microtc-perf command.

```
microtc-perf gold.json test-predicted.json
```

This will show a number of scores in the screen.

```
{
  "accuracy": 0.850625,
  "f1_0": 0.8863528292914883,
  "f1_1": 0.7821330902461258,
  "macrof1": 0.834242959768807,
  "macrof1accuracy": 0.7096279176533414,
  "macrorecall": 0.826785580669612,
  "microf1": 0.850625,
  "quadratic_weighted_kappa": 0.6690373825911413
}
```