NLP for political polarity classification from tweets

Nisim Jonatan Hurst Tarrab

Abstract

Tweets are a common way for political candidates to pronounce themselves about some topic. However, the use of jargon, open ended language and lack of context turn most of the direct classification methods useless. However, cues of the political and twitter realm can be used to extract features that can improve sentiment analysis substantially. In this work we will follow mainly the approach proposed by [14] of using machine learning classifiers but also will in some way give a limited amount of importance to the gramatical structures. We first extract common sense features, that hereby are called *special tokens*. Then, we merge those features with bigrams of special tokens that probably take precedence in each of the attitudes. Finally, word2vec attributes are used to compute sentence orientation regardless of word order. Word2vec features can preserve syntactical and semantical relationships between words. The main contribution of this work is to be a proof-of-concept for extracting syntactic grammatical and semantic features on the domain of actual Mexican policy that improve automatic tweet attitude tagging. A simple special tokens bigram counts and wrod2vec features have shown an improvement over bag-of-word of y on average and an improvement of 0.07 and 0.20 respectively.

1. Introduction and problem understanding

Tweets are a common way for political candidates to express their opinions about current affairs. Since the arrival of the Web 2.0 microblogging platforms have become political instruments and reveal political attitudes of political candidates all over the world (except for those places in which government controls internet access like China). An example of previous work done on the pollical milieu can be found on [17], [3] and [6].

A group of Mexican anthropological researchers (Marta Bárbara Ochman et al., 2016) took the task of making a taxonomy just for classifying political attitudes that can be identified on tweets. Their hypothesis is that those attitudes are correlated with future campaign proposals. The taxonomy developed was the following:

1. Proactive: Tweets under this category are aimed to generate information about their personal virtues, their proposed program and their party's current efforts.
2. Reactive: Seek to neutralize their adversaries' derisions and any infamous depiction on the media.
3. Aggressive: Emphasize negative traits of their enemies or defame their opponents.
4. Vote winner: Demagogue speech aimed at winning a political advantage.

No one can gainsay that labeling each of these tweets is drudgery. Thus, a model that can classify those tweets with minimal human intervention is highly desirable.

However, this task isn't easy. Tweets are very limited in the following ways [10]:

- Data sparsity.

- Changing nature of languages due to trending topics.
- Political candidates usually make use of jargon and informal language.
- Lack of context from the text. The text field is limited to 40 characters.
- Short irregular forms may be used.

Clearly, if we try to use a machine learning classifier over the raw tweets headfirst, the results would be disappointing. So, on the one had we have these data highly skewed and sparse with the problems just mentioned. On the other hand, we have all these mature machine learning algorithms dating back to the 50's. We need to find good instance representation that our models can use uniformly, expresses latent attributes present in the data and finally reduce noise produced by redundant features that just doesn't add any value. But how?.

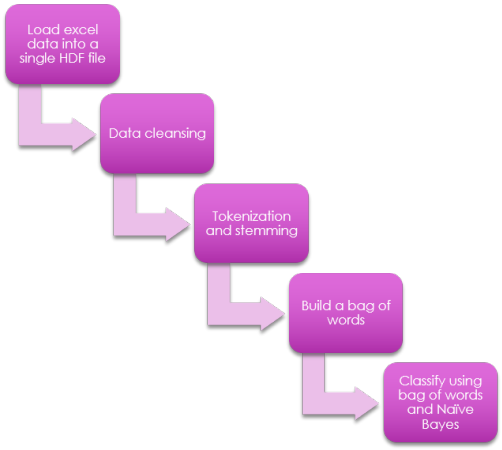


Figure 1: Text preprocessing pipeline

Well, firstly we should use some common preprocessing steps that might help semantic and grammar decomposition or somehow incorporate domain knowledge into our search space. These techniques are show in Section 4. Secondly, deep learning techniques using CNN have prove to be useful in many realms, from heuristics design to computer vision. NLP is not an exception. Here we can used them to extract those latent features we mentioned earlier. A common technique that has been infallible is word2vec. Since its conception in [11], this method has been capable of both preserving semantic and syntactic representation and can be used reliably to categorize a bag-of-words when data is sparse.
The initial hypothesis of the projects is:

1. A set of features represented in a word2vec vector representation of the tweet can leverage the power of an already trained word2vec model and gives a Naïve Bayes classifier a very low generalization error [1].
2. A more diverse set of features can increase accuracy [15]. Thus, a minimum representation of a grammatical structure, i.e. a bigram count of *special tokens* are added to the resulting set of features. This bigram count increases the classifiers accuracy.

[1] Although there exists an Spanish corpus it is not focused on political jargon and each party has its own jargon

3. Normalized features achieve better results and can be selected more easily because they are scale invariant. Thus, the vectors corresponding to tweets with different lengths are weighted. However, tweet length will be added to the features in representation of energetic grammatical structures.

Results show that the ROC AUC curve increases significantly, from 0.65 (on average for the four political attitudes) using only bag of words to 0.72 using only bigram features, to 0.85 using word2vec features and finally to 0.857 using all the features.

In the final model we will see that the top 5 most important features are for all the four labels coming from V2W.

2. Previous works

[13] places Sentiment Analysis (SA) within the área of Natural Language Processing (NLP) and can be defined as the computational treatment of opinions, feelings, and subjectivity in text. This article mentions that early history places 2001 as the milestone at which a widespread awareness began to arise around sentiment analysis, with beliefs systems as forerunners. One of the factors behind this land rush was the availability of datasets for machine learning on the World Wide Web.

[10] brings to the table two of the firsts approaches for the research community to tackle the problem of SA. [18] proposes the use of linguistic analysis. This kind of approach can be thought as supervised because it relies on previous domain knowledge, e.g. Chomsky grammatical structures. At the other end we have [14], which proposes the use of classical machine learning techniques. Contrary to the approach taken by Turney, here we rely more on achieving a high accuracy using an ensemble of different techniques, commonly ignoring grammatical structures as in the case of a simplification using bag-of-words representation.

The bag-of-words representations gets its name form a passage from linguist Zellig Harris (1954), "language is not merely a bag of words but a tool with particular properties.". [12] suggest we think of the model as "putting the words of the training corpus" in a bag and then selecting one word at a time. Then, the notion of order is lost, but we end up with a binary vector that we can neatly use in our machine learning classifiers.

Now, let's go back to a more recent 5-year horizon. As aforementioned there is an extreme impairment over context in which we are just fettered to a 140 characters text context. Furthermore, tweets usually don't have representative and syntactically consistent words. [8] proposes a sentiment grade for each distinct notion in the post using an ontology instead of evaluating it as a whole. The authors use a *Formal Concept Analysis* (FCA) algorithm proposed by [5] in which applies a user-driven-step-by-step methodology for creating domain models, i.e. it creates an ontology specific for the bulk of tweets to classified. Tweets were classified on a rank per topic. They used a tool called OntoGen in which a semi-supervised approach was possible.

Through the lens of our work, topic and ontologies could prove useful when considering political parties, allies, government institutions, commercial and foreign institutions. However, these ontologies must be built mostly by human annotations, a cost we cannot afford in this study.

The approach taken by [16] was a bit different. The paper measures how to word of mouth (WOM) affect movies sales, negatively or positively. There were four tweet categories very similar to the ones we are measuring: intention tweets, positive tweets, neutral tweets and negative tweets. *Intention tweets* are very similar to our *vote winner* category because an intention to win votes can be achieved either by aggressive of proactive tweets. The authors decided to use two well-known classical machine learning classifiers: Naïve Bayes and Support Vector Machines. This approach is similar

the one proposed by [14] in which we harness the efficiency of classical machine learning algorithms by using meaningful instance representations.

In the work of [9] many approaches for feature extraction are mentioned. Namely, extracting frequent terms while measuring compactness, association rule mining to find syntax rules, ontologies, hyponyms (more general) and meronyms. However, most of the methods mentioned in the introduction use unigrams, ngrams and part-of-speech (POS) [10]. The next section will explain our approach.

3. Proposed algorithm

We propose an ensemble of features that can be better representations than the bag-of-word alone. Although *word2vec* preserves semantic and syntactical relationships, it does not preserve grammatical structures. This drawback can be compensated by just using bigram structures of special tokens in which the order is still maintained. In the following listings we present pseudocode to achieve each piece of the final vector representation that will be used by a Naïve Bayes classifier (vs the classical bag-of-words representation).

Algorithm 1 ExtractFeatures -Extraction upper process

Input: N - a tweet, $tokenList$ - list of tokens that should be used

Output: FS - a set of features

1: **procedure** ExtractFeatures(N)
2: $FS \leftarrow \varnothing$
3: $BOW \leftarrow$ ExtractBOW($N.Text$) ▷ Bag-of-Words extraction
4: $W2V \leftarrow$ ExtractW2V ($N.Text$) ▷ Word2Vec extraction
5: $BIG \leftarrow$ ExtractBIG ($N.Text$, $tokenList$) ▷ Bi-gram extraction
6: $FS \leftarrow BOW \cup W2V \cup BIG$ ▷ Union-all
7: $FS \leftarrow$ Normalize(FS)
8: **return** FS
9: **end procedure**

Algorithm 2 ExtractBOW -Extract BOW representation

Input: N - a tweet, $wordList$ - twitter word list

Output: BOW – a binary vector of words

procedure ExtractBOW(N)
    $BOW \leftarrow$ zeros(1,len($wordList$))
    $index \leftarrow 0$
    for each $i \in wordList$ **do**
        if $i\ in N.Text$ **then**
            $BOW[index] \leftarrow 1$
        end if
        $index++$
    end for
    return BOW
end procedure

Algorithm 3 ExtractW2V -Word2Vec feature extraction

Input: N - a tweet, $w2v\_model$ - a neural network pre-trained model that maximizes conditional probability of context given a word

Output: W2V - a set of features

procedure ExtractW2V(N, $w2v\_model$)
    $W2V \leftarrow zeros(1, len(N.Words))$
    $index \leftarrow 0$
    for each $i \in N.Words$ **do**
        $W2V[index] \leftarrow$ ApplyW2V($w2v\_model$, i)
        $index++$
    end for
    $W2V \leftarrow avg(W2V, axis = 1)$ ▷ Calculate the sentence vector by averaging the vector of their words
    return W2V
end procedure

Algorithm 4 ExtractBIG -Bigram count vector

Input: N - a tweet, *tokenList* - list of tokens that should be used

Output: BIG – bigram counts

procedure ExtractBIG(N)
    $bigram\_list \leftarrow$ GenerateBIG ($tokenList$)
    $sentence\_bigram\_list \leftarrow$ GenerateBIG ($N.Words$)
    $BIG \leftarrow zeros(1, len(bigram\_list))$
    $index \leftarrow 0$
    for each $i \in bigram\_list$ **do**
        if $i \in sentence\_bigram\_list$ **then**
            $BIG[index] \leftarrow 1$
        end if
        $index++$
    end for
    return BIG
end procedure

Algorithm 5 GenerateBIG

Input: *tokenList* - list of tokens that should be used

Output: BIG - a set of features

```
1:  procedure GenerateBIG(tokenList)
2:      global specialTokens                ▷ Globally defined special part of speech and punctuation
3:      SpecialBigrams ← CombinationsOf2(specialTokens)
4:      BIG ← zeros(1, len(SpecialBigrams))
5:      for each i ∈ tokenList do
6:          if not (i ∈ specialPOS) then tokenList.remove(i)
7:          end if
8:      end for
9:      for i = 0 to len(tokenList) − 1 do
10:         for j = i + 1 to len(tokenList) do
11:             bigram ← new Bigram(tokenList[i], tokenList[j])
12:             pos ← Find(SpecialBigrams, bigram)
13:             BIG[pos] + +
14:         end for
15:     end for
16:     return BIG
17: end procedure
```

Special tokens to use on the Bigram generator are:

Token	Purported Attitude
ellipsis	Reactive
exclamation	Aggressive, Proactive
hashtag	Proactive, Vote Winner
mention	Proactive, Reactive
name	Aggressive, Vote Winner
neg_emoticon	Aggressive
pos_emoticon	Proactive, Vote Winner
question	Proactive
quoted	Vote Winner
uppercase	Aggressive
url	Proactive
colon	Vote Winner, Proactive
semicolon	Aggressive
comma	Vote Winner

These tokens will also be counted individually (1-gram). In contrast to other approaches found on the internet [1], the bigrams will be counted instead of just asserting their presence.
We can see that the algorithms are quite simple. This owns to the fact that we are relying more on the pretrained neural networks models that came with the gensim python library for word2vec representations. Those packages already came pretrained on [2].

4. Experimental setup

Our dataset is small. From over 51,453 samples extracted from the provided Excel files, we just have labels for only 7,594 of them. Due to the skewness we have decided to make a stratified sample set consisting of $\frac{3}{5}$ of the data for training and $\frac{2}{5}$ thirds for testing, i.e. 4,500 and 3,094 respectively.
Ground truth consists of a rank given for each of the four categories in each tweet. These ranks go from "0" to "9" and can be considered as ordinal values. As far as this study is concerned, no correlation exists between these 4 target classes. Thus, we will treat each category as a separate classification problem.
Some tweets present an homogeneous structure (having only one class dominate over the others) while other tweets are more ambiguous. The following figure shows the 4 target classes distribution in a binary way, "0" equals "not present" and "not 0" equals "present":

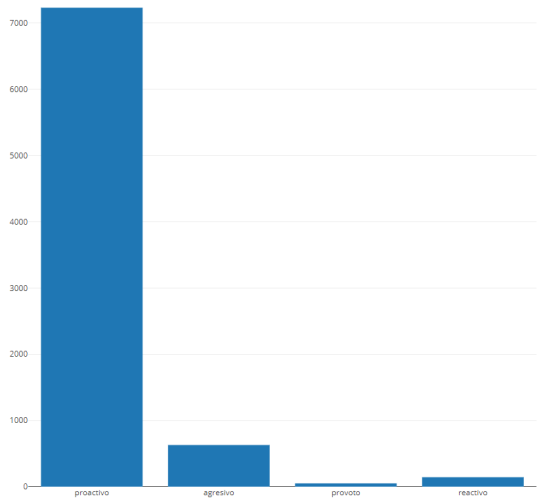


Figure 2: Data is heavily skewed towards proactive attitude. The approach given in section 3 would help us ameliorate this problem.

In most of the provided files there were three human annotators. However, through a thorough database perusal it was found that only Martha had valid annotations. So, let's do a variance analysis over each of the target classes at least for Martha.

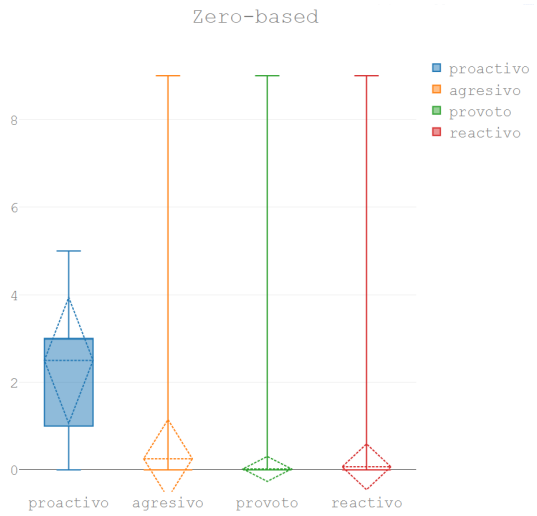


Figure 3: Overall category rankings provided by Martha

Just as expected, Martha is mostly filling the proactive field without further care for any of the other categories. The rest of the categories have a mean close to zero. However, a comparable variance analysis can be done filtering out the times when those categories weren't used and taking into account only the tweets which have a value greater than zero.

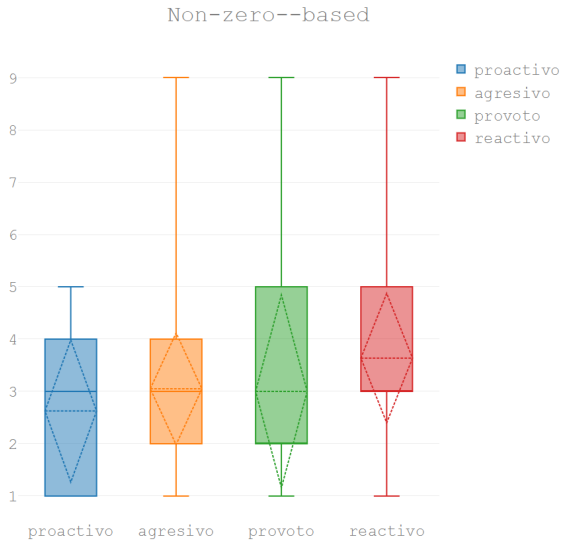


Figure 4: W hen Martha decides to use some category ($rank \neq 0$), which numbers she uses more frequently

Here are some observations we can derive from the graph:

- While the most used category is proactive, Martha is still quite reserved in the scores she gave on this category, reaching a peak at 5.
- The vote-winner category has a similar interquartile range but reaches a more radical peak at 9.

- The reactive and aggressive categories swing wildly but their means are between 3 and 4.

Now let's explore some the most common words that can serve of basis for a bag-of-words representation. We will remove the stop words because they are too common (low inverse document frequency) and do not add value to the contrast we are trying to discover.

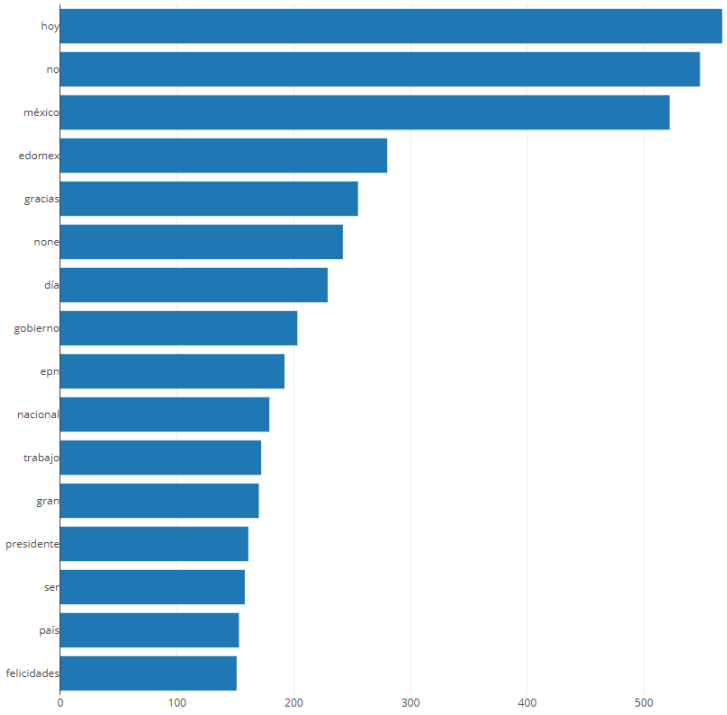


Figure 5: Bar chart showing the most frequent used word

However, this word list still contains political parties and names. Let´s remove them and split by category usage.

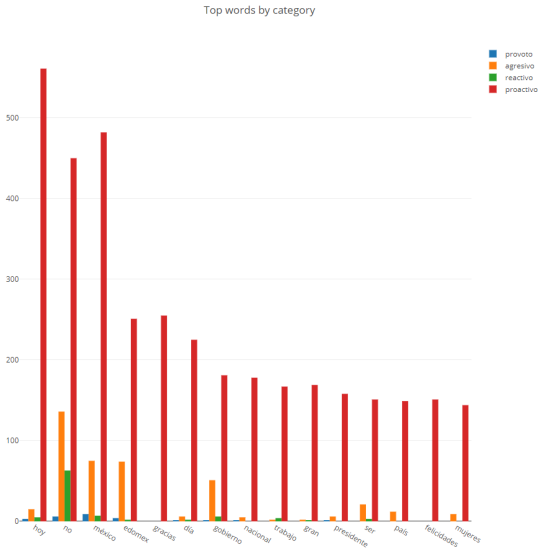


Figure 6: Most used words per category

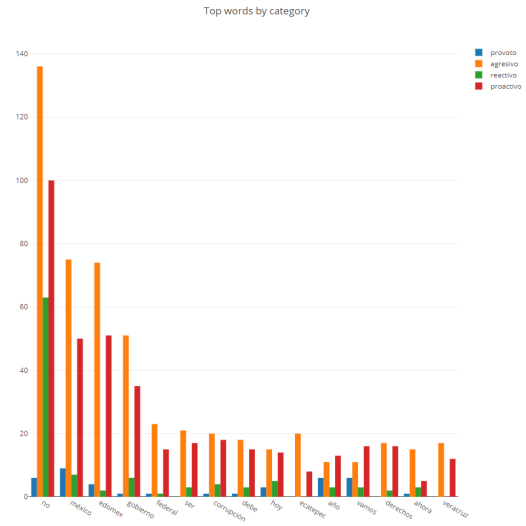


Figure 7: Word usage per category taking into account that at least that word was used in other category from proactive

Owning to the fact that we added the word “no” to the exception list is the most common word.

A ROC curve will be used to measure accuracy selecting different set of features, once using a Naïve Bayes classifier and once using Random Forest. Naïve Bayes classifiers works well when using prior binary knowledge of which words are present in a sentence, even using a bag-of-words and is probable it will get more uniform results. To complete our test Random Forest will exploit the features more uniformly and probably will get a significant improvement.

It is impossible to measure the statistical significance of the improvement because we are only using one database. According to [4] we need at least $na * k$ where a is a number between 2 and 8 and k is the number of classifiers we are testing.

Wilcoxon test, which deals for just two classifiers as in our case, also relies on this number of separate databases. Finally, we are going to measure feature importance in the accuracy of each classifier.

5. Results and discussion

From a source code view (PyCharm project provided), the process taken to generate the results where:

1. In the *cleansing* python package run the files in the following order:
    a. **Load_tweets.py**. Centralizes the information contained in the provided Excels files into a one hdf5 and then a pickle file called final.pickle.
    b. **Tockenize_tweets.py**. Generates the column *tokenized_text* which is the text array in which has cleansed text for the BOW extraction.
    c. **Generate_tokenized_text_noparties.py**. Removes stopwords and political party names from the *tokenized_text* column.
    d. **Categories_boxplot.py**. Visualize Martha preference treating the classes as if they were numerical.
2. The *testing* package has some files we can use at this point. Those are:
    a. **Wordlist_histogram.py**. Plot that shows the most frequent words that will be pprobably important for our classifier.
    b. **Wordlist_category_histogram.py**. Help us visualize the problem in terms of most frequent words distribution over the 4 attitudes.
    c. **Box_plots.py**. Visualize Martha preference contrasting attribute usage by the presence of zero.
3. Here comes our contribution. In the *features* packages there are the following files:
    a. **ExtractBOW.py**. Binary bag-of-words vector extraction. All the extracted features are saved in Excel and pickle for latter usage.
    b. **SpecialTockens.py**. This file contain the function that we are going to evaluate in order to compute the special tokens ngram model.
    c. **ExtractBIG.py**. Common sense features are extracted first by counting special tokens. Then bigrams are extracted and counted to preserve grammatical structures that are obviated by the W2V representation.
    d. **ExtractW2V.py**. Spanish word2vec vector representation are extracted for each word and then and an average overall tweet vector is calculated.
    e. **ExtractFeatures.py**. Consolidate all the features into one file.
    f. **Save.py**. Save the features to an Excel and a Pickle file for rapid usage.
4. Having generated all the features, lets go back to our *testing* package:
    a. **Test_classifier.py** . Contains base functions for calculating the *ROC AUC* for multiclass environment, as well as **precision**, **recall**, **accuracy** and **f1-scores** that can be appreciated indirectly on the confusion matrices. For the sake of conciseness, the only metric presented on the report was the *ROC AUC*. A cross validation function is provided.
    b. **Confusion_matrix.py**. Two graph type were generated, the general one presented on Section 6 and the confusion matrices that will be explained in due course.
    c. **Feature_importance.py**. Finally this file help us visualize what are the most relevant top 20 features for the *W2V+BIG+BOW RandomForest*.

Having explored the data we saw there is a tendency for proactive classification. Thus, we should expect having low recall values for the aggressive, vote-winner and reactive classes. Two classifiers will be tested, a Naïve Bayes classifier that adapts very well to binary features and a Random-Forest one that can use features more uniformly. Both are natively multiclass. Random Forest was trained with 100 trees.

In this section we evaluate each of the feature sets individually taking into account also the class there are trying to predict.

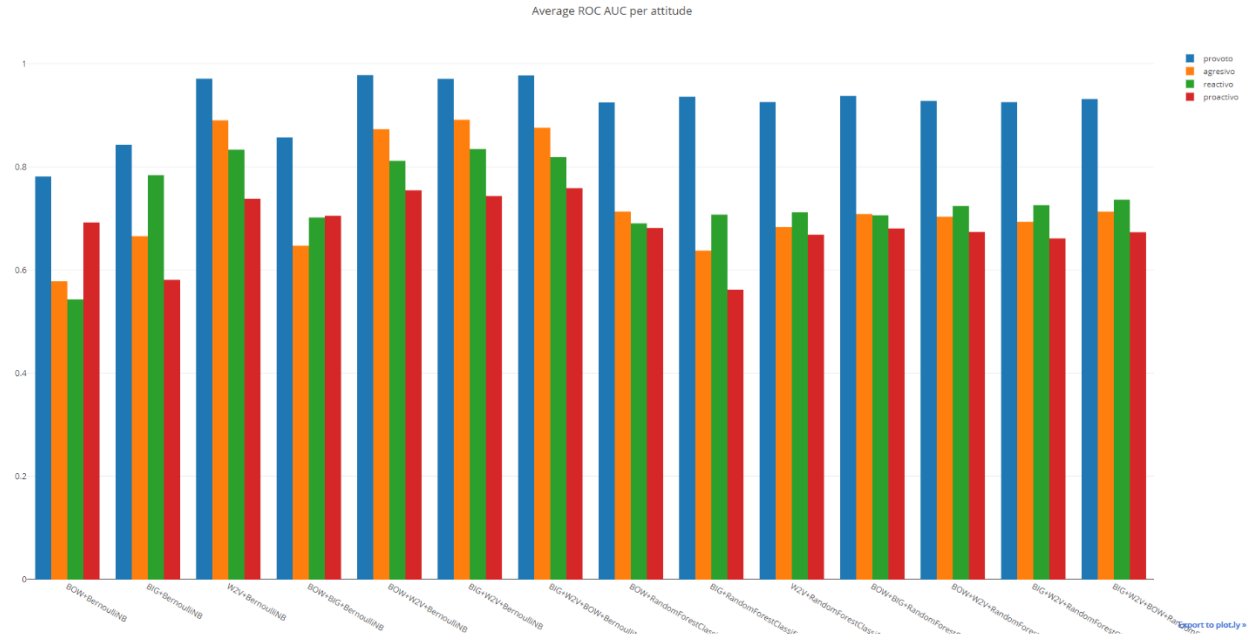
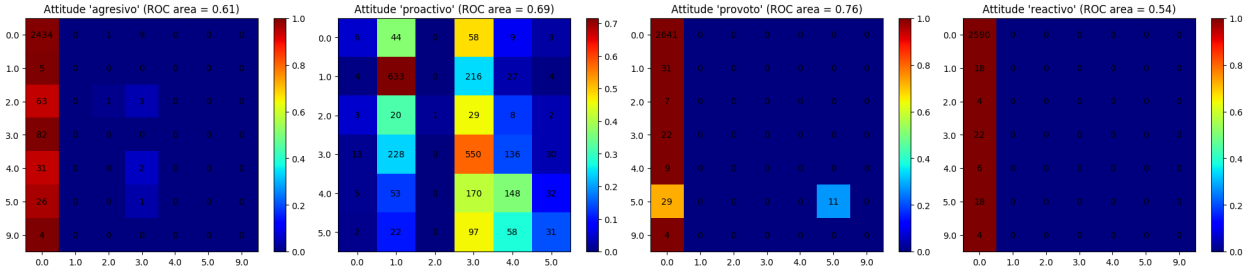
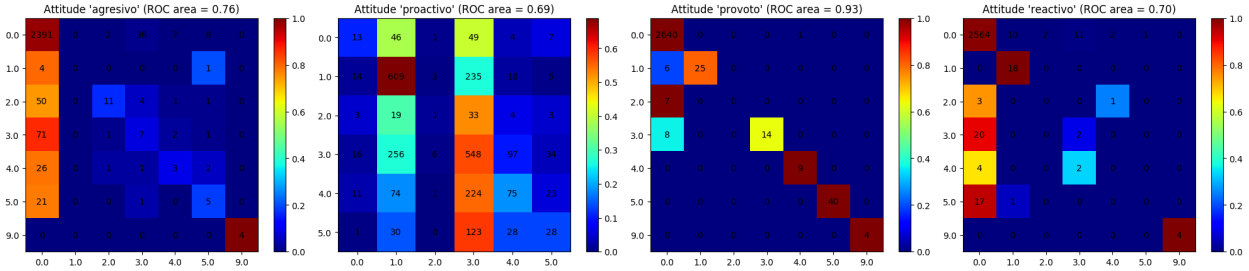


Figure 8: Results obtained expressed as the macro average ROC area using a one vs rest classifier (Weka approach)

General ROC areas (one vs rest) demonstrate that BIG+W2V using Naïve Bayes has a slightly better accuracy than the rest of the features. BOW features actually worsened the classifier accuracy and so do using Random Forest. However, literature recommends using confusion matrices for measuring performance on multiclass classifiers. Let's derive some conclusions from looking them.

5.1. Bag-of-Words

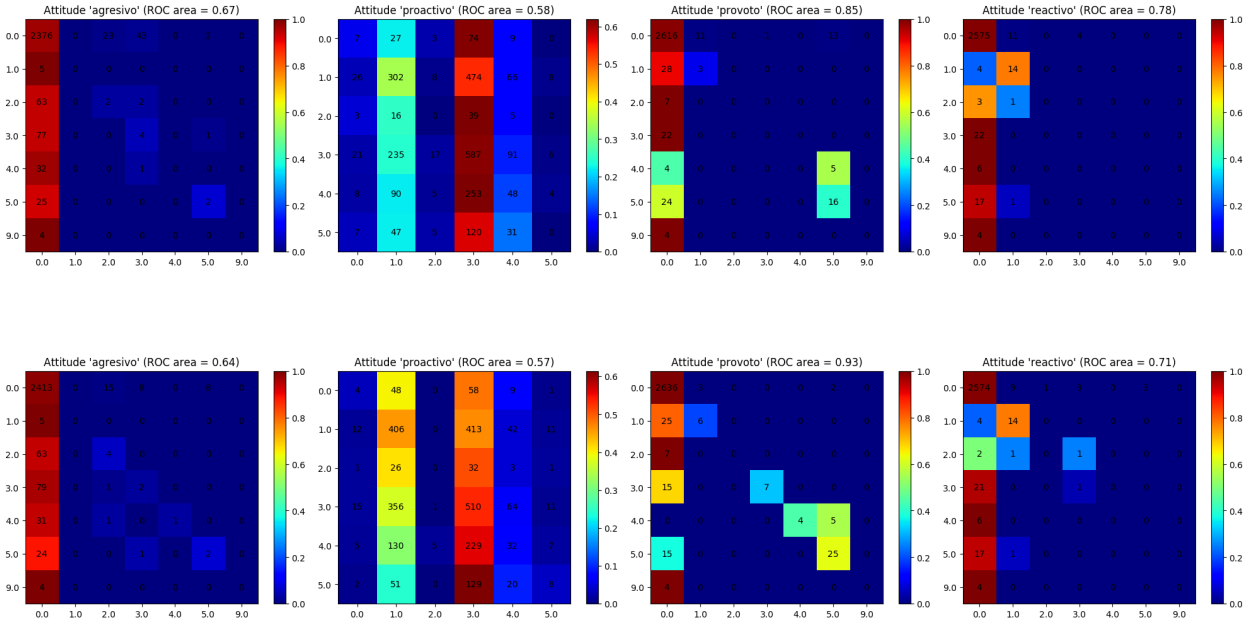




The first row of the confusion matrix heatmap show the BernoulliNB classifier and the second one the RandomForestClassifier.

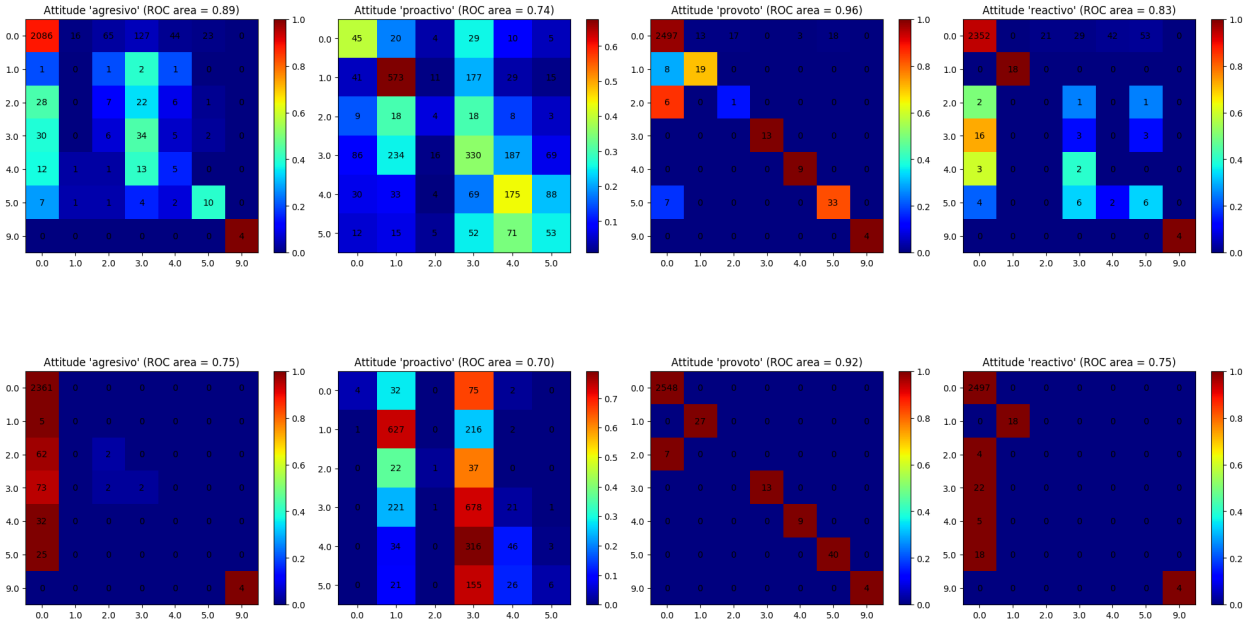
It is no surprise that bag-of-word has an good performance on the *Proactive* label. However, the actual data has a penchant for 3 so our classifier fails to pickup this static tendency. *Vote winner* category improves significantly when we use maybe owning to the fact that RandomForest uses the features better.

5.2. Bigrams



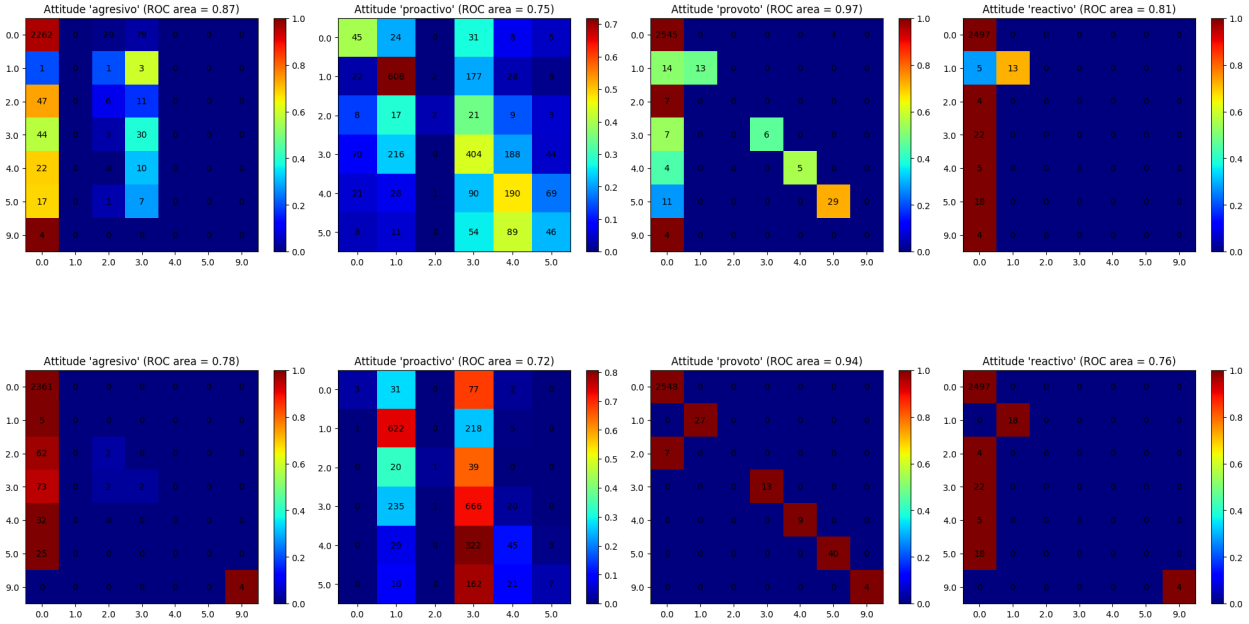
Compared to BOW, Bigrams improve the aggressive and the provote and reactive categories using *Naïve Bayes*. However, using *Random Forest* the behavior is a little worse.

5.3. *Word2Vec*

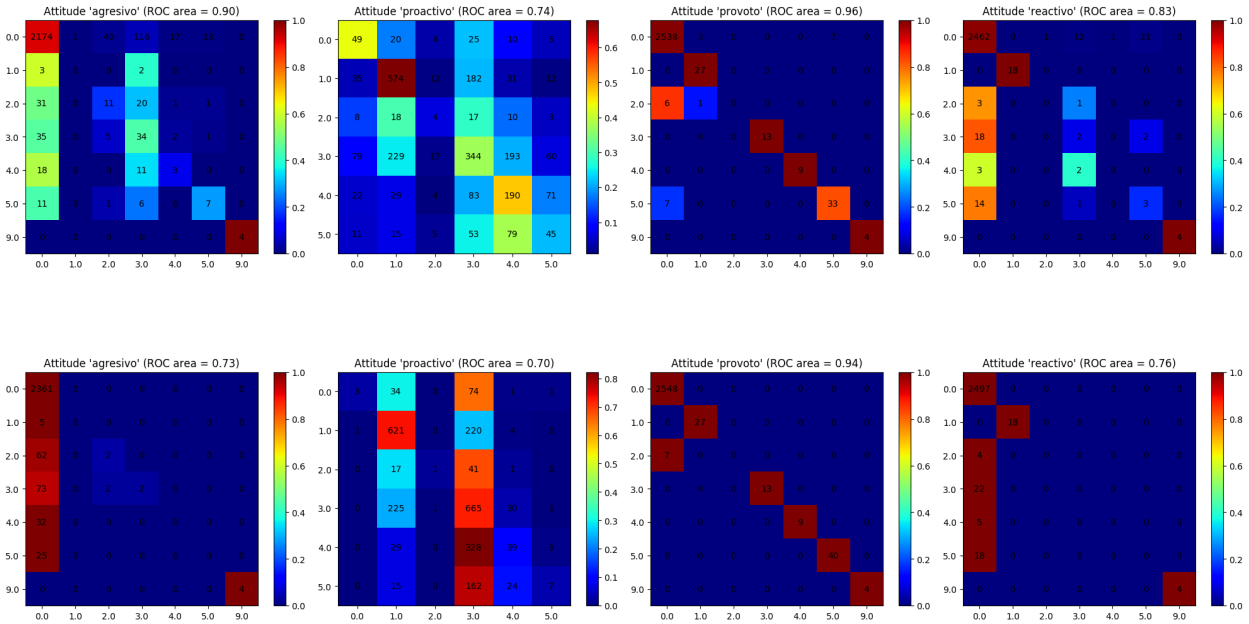


Word2vec features improve substantially both NB and RF. *Provoto* precision and recall levels also seem to improve.

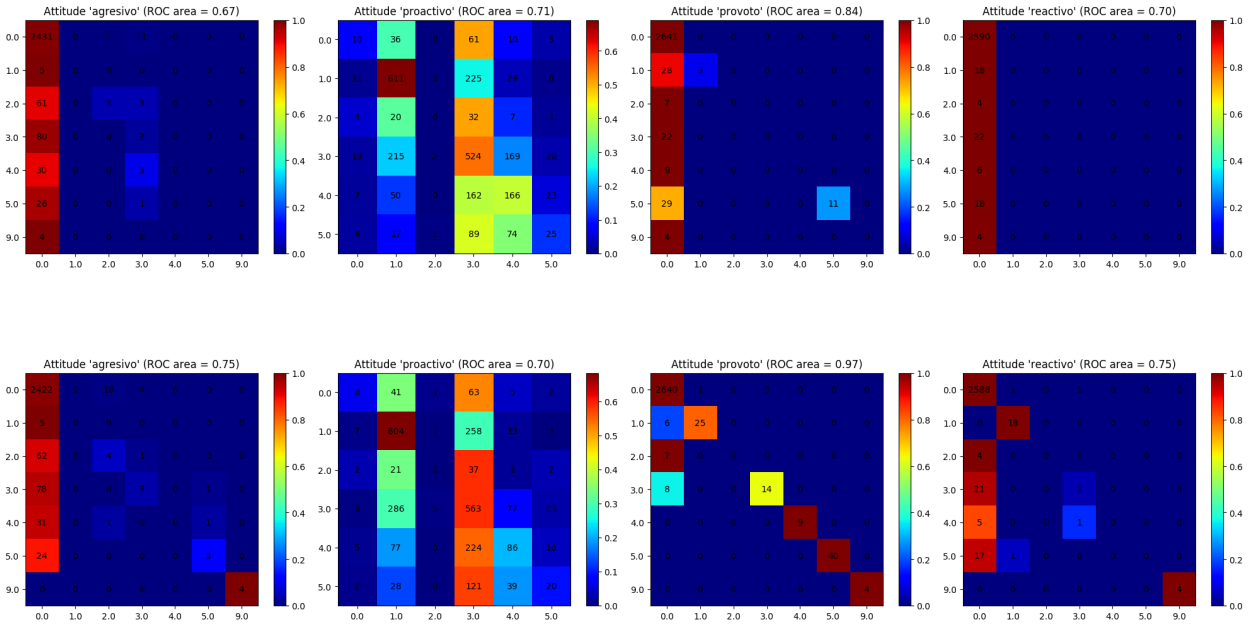
5.4. *Word2Vec + BOW*



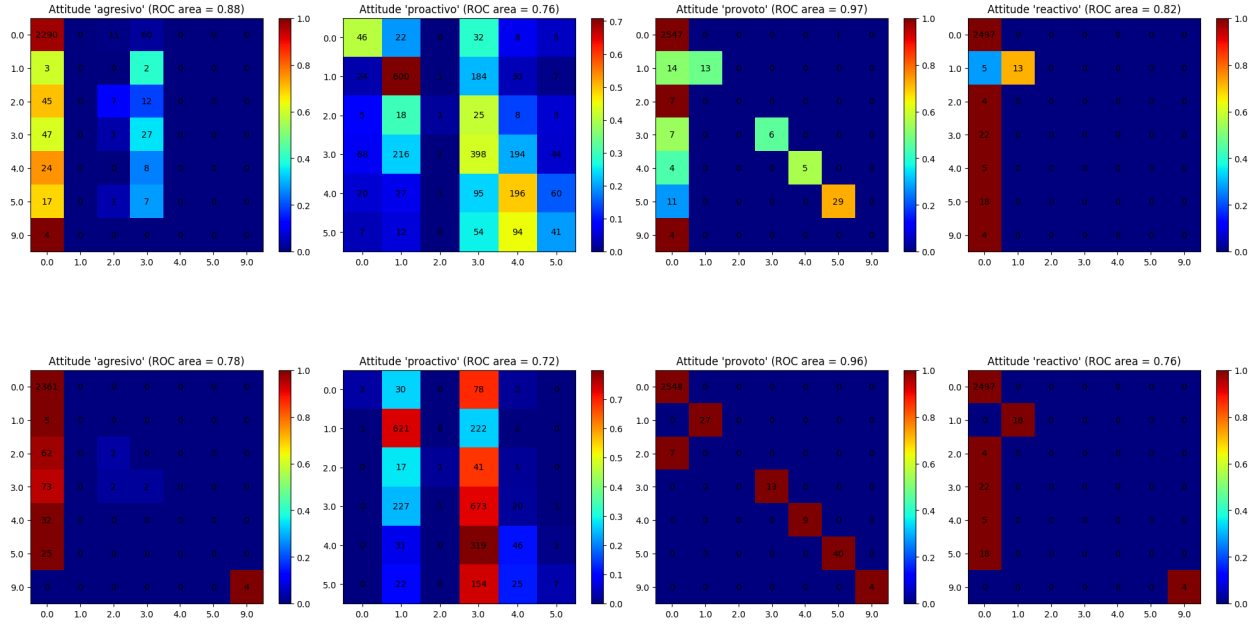
5.5. *Word2Vec + Bigrams*



5.6. *BOW + Bigrams*



Mixed features (BOW + W2V + BIG)



All the features combine generate a nearly perfect *Provoto* classification for *RandomForest*. However, that is as far as it gets, because the data is insufficient for learning other classes. The vertical lines indicate that the actual class is always static, the source of a big generalization error that could be ameliorated if we had more data.

6. Feature importance

We can see from the following figures that that most helpful features come from the word2vec classifier. Some come from BIG and a bit fewer from BOW.

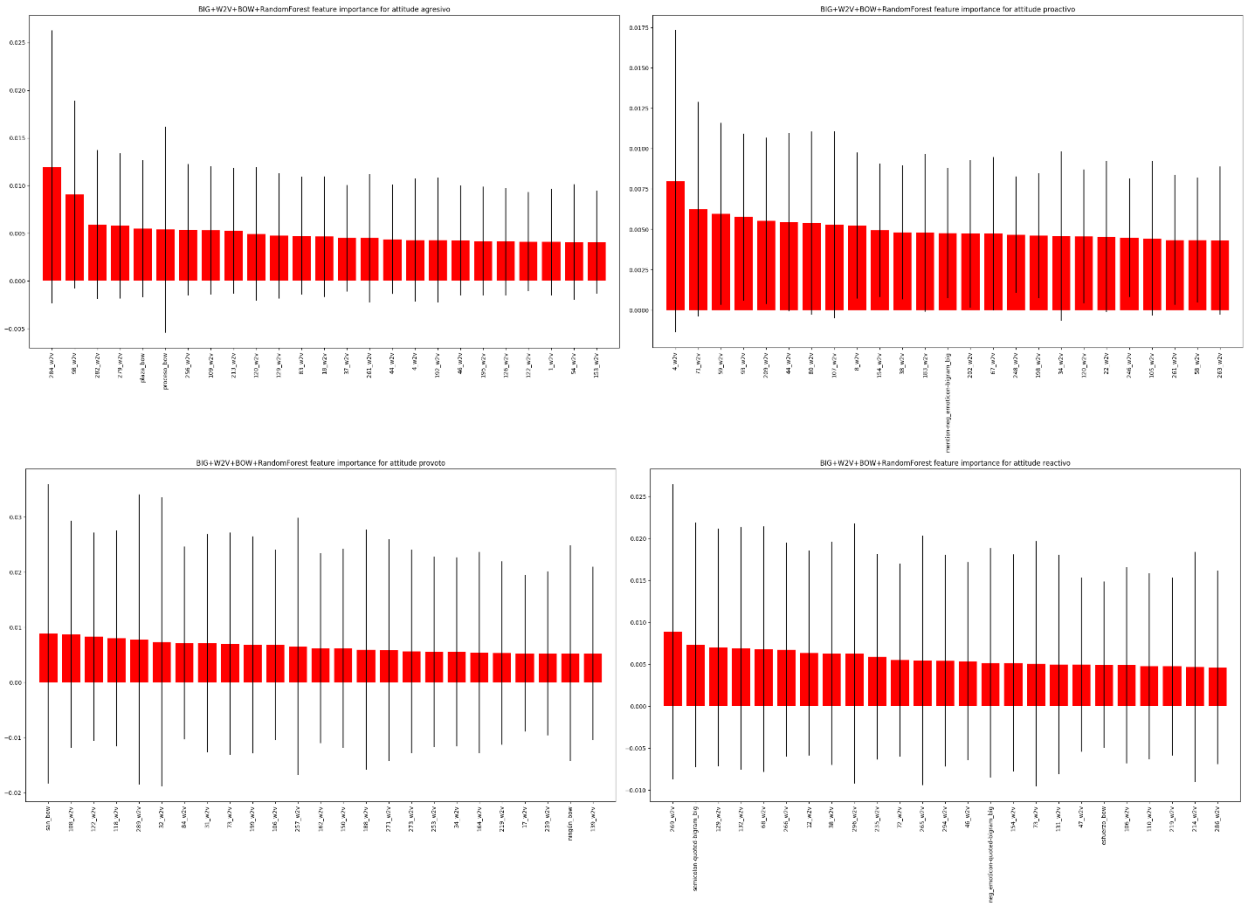


Figure 9: Feature importance

7. Conclusions

Features with which we feed a machine learning algorithm are very important. We just saw how by just adding bigram features improved accuracy and *ROC AUC* consistently. Feature work may generate more levels of bigrams and then made feature extraction by importance to improve the *recall* of the classifier.

Word2vec vector representation can help integrate the syntactic and semantic structures of any language and find similarities between sentences. Those similarities can then be used to train a more complex classifier.

The vertical lines in our results indicate that the actual class is static and that there is a big generalization error, so we probably we could do better with more data.

8. Reference work listing

An interesting study on political analysis of tweets was done by [7] in which a Spanish tweets were used in conjunction with word2vec for political elections prediction. They measure their results over the F1-score and using Support Vector Machines. On two distinct phases of 2242 and 16 million tweets, they achieved 60.5% and 85.09% respectively. A similar

approach of building n-grams was used. Compared to our results they achieved better results but that can be explained by the nature of their problem which was a single class problem.
Another starting point was the work published in [1]. However, in that work they don't use bigram generation nor bigram counting to feed the classifiers.

9. References

References

[1] , 2017. Sentiment analysis of tweets with python, nltk, word2vec & scikit-learn - marcin zabłocki blog.
URL http://zablo.net/blog/post/twitter-sentiment-analysis-python-scikit-word2vec-nltk-xgboost

[2] , 2017. Spanish billion word corpus and embeddings.
URL http://crscardellino.me/SBWCE/

[3] Bermingham, A., Smeaton, A. F., 2011. On using twitter to monitor political sentiment and predict election results. Psychology, 2–10.

[4] Derrac, J., García, S., Molina, D., Herrera, F., 2011. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm and Evolutionary Computation 1 (1), 3–18.
URL http://dx.doi.org/10.1016/j.swevo.2011.02.002

[5] Ganter, B., Wille, R., 1999. Formal concept analysis : mathematical foundations. Springer.
URL https://dl.acm.org/citation.cfm?id=550737

[6] Jungherr, A., Jürgens, P., Schoen, H., 2012. Why the pirate party won the german election of 2009 or the trouble with predictions: A response to tumasjan, a., sprenger, t. o., sander, p. g., & welpe, i. m. "predicting elections with twitter: What 140 characters reveal about political sentiment". Social Science Computer Review 30 (2), 229–234.
URL http://journals.sagepub.com/doi/10.1177/0894439311404119

[7] Khandelwal, A., Swami, S., Akhtar, S. S., Shrivastava, M., 2017. Classification of spanish election tweets (coset) 2017: Classifying tweets using character and word level features. CEUR Workshop Proceedings 1881, 49–54.

[8] Kontopoulos, E., Berberidis, C., Dergiades, T., Bassiliades, N., 2013. Ontology-based sentiment analysis of twitter posts. Expert Systems with Applications 40 (10), 4065–4074.
URL http://dx.doi.org/10.1016/j.eswa.2013.01.001

[9] Li, Y. M., Li, T. Y., 2013. Deriving market intelligence from microblogs. Decision Support Systems 55 (1), 206–217.
URL http://dx.doi.org/10.1016/j.dss.2013.01.023

[10] MARTÍNEZ-CÁMARA, E., MARTÍN-VALDIVIA, M. T., UREÑA-LÓPEZ, L. A., MONTEJO-RÁEZ, A. R., 2014. Sentiment analysis in twitter. Natural Language Engineering 20 (1), 1–28.
URL http://www.journals.cambridge.org/abstract_S1351324912000332

[11] Mikolov, T., Chen, K., Corrado, G., Dean, J., Jan 2013. Efficient estimation of word representations in vector space.
URL http://arxiv.org/abs/1301.3781

[12] Norvig, P., 2012. Artificial Intelligence: A Modern Approach.

[13] Pang, B., Lee, L., 2008. Opinion mining and sentiment analysis. Foundations and Trends® in Information Retrieval 2 (1–2), 1–135.
URL http://www.nowpublishers.com/article/Details/INR-011

[14] Pang, B., Lee, L., Vaithyanathan, S., 2002. Thumbs up? sentiment classification using machine learning techniques.
URL http://www.aclweb.org/anthology/W02-1011

[15] Ramirez-marquez, J., 2018. Some features speak loud, but together they all speak louder: A study on the correlation between classification error and feature usage in decision-tree classification ensembles. Engineering Applications of Artificial Intelligence 67, 270–282.
URL http://www.sciencedirect.com/science/article/pii/S0952197617302488

[16] Rui, H., Liu, Y., Whinston, A., 2013. Whose and what chatter matters? the effect of tweets on movie sales. Decision Support Systems 55 (4), 863–870.
URL http://dx.doi.org/10.1016/j.dss.2012.12.022

[17] Tumasjan, A., Sprenger, T., Sandner, P., Welpe, I., 2010. Predicting elections with twitter: What 140 characters reveal about political sentiment. Proceedings of the Fourth International AAAI Conference on Weblogs and Social Media, 178–185.
URL http://www.aaai.org/ocs/index.php/ICWSM/ICWSM10/paper/viewFile/1441/1852

[18] Turney, P. D., 2002. Thumbs up or thumbs down? semantic orientation applied to unsupervised classification of reviews.
URL http://www.aclweb.org/anthology/P02-1053.pdf