Task 2: Classification - Natural Language Processing

Machine Learning

Agustín Mora Acosta
Agustin.Mora1@alu.uclm.es
UCLM
Ciudad Real, Spain

Andrés González Díaz
andres.gonzalez8@alu.uclm.es
UCLM
Ciudad Real, Spain

ACM Reference Format:
Agustín Mora Acosta and Andrés González Díaz. 2020. Task 2: Classification - Natural Language Processing: Machine Learning. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 PROBLEM DESCRIPTION

The purpose of this work is to analyze a Twitter collection about offensive language and hatred.

Data used in this work comes from CrowdFlower, a company dedicated to acting as a mediator between companies and users who will carry out mini-jobs. These companies pay for certain tasks to be completed, and CrowdFlower is in charge of classifying them, establishing a payment for each one and then distributing them among its associates. In this case, the job was to classify tweets, indicating whether they were offensive language, hate speech or neither.

User generated content (UGC), such as the text in Twitter messages, is notoriously varied in content and composition, often containing ungrammatical sentence structures, non-standard words and domain-specific entities, this is why applying NLP techniques can be complex.

The collection contains the following fields:

- **count**: Number of CrowdFlower users who coded each tweet.
- **hate_speech**: Number of CrowdFlower users who judge the tweet to be hate speech.
- **offensive_language**: Number of CrowdFlower users who judge the tweet to be offensive.
- **neither**: Number of CrowdFlower users who judged the tweet to be neither offensive nor non-offensive.
- **class**: Class label for majority of CrowdFlower users. This class can take 3 different numerical values, depending on how it has been classified by CrowdFlower users. These classes are as follows:
  (0) hate speech
  (1) offensive language
  (2) neither
- **tweet**: Textual information to process.

Conference'17, July 2017, Washington, DC, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Our main objective is to apply techniques of text and language processing to discover useful and previously unknown "gems" of information in large text collections, like we present, and implement and facilitate Sentiment Analysis tasks using NLTK features and classifiers.

2 METHODS AND MATERIALS

For this tasks we will use **Natural Language Toolkit** (NLTK). Natural Language ToolKit (NLTK) is a comprehensive Python library for natural language processing and text analytics. NLTK is often used for rapid prototyping of text processing programs and can even be used in production applications. Not only does it provide convenient functions and wrappers that can be used as building blocks for common NLP tasks, it also provides raw and preprocessed versions of standard corpora used in NLP literature and courses.

2.1 Preprocessing & Transformation

First, we have applied some methods of preprocessing and data transfromation, we can separate them into *"Pattern detection"*, *"Replace of contractions, emoticones and others tokes"* , *"Spelling Corrector Declarations"*, *"lemmatize terms"* and *"other preprocessing methods"*.

2.1.1 ***Spelling Corrector Declarations.***

It's common for users to intentionally and unintendedly make spelling mistakes when writing tweets, that's why we have to declare some methods that will be useful to perform spelling correction over the tokens founds in dataset. Some of these methods have been favored by Jesus Serrano Guerrero and was been adapted using *pyspellchecker*, a python library that proposes another word that could be the searched word. For that we need a extensive library containing the words to compare. Spelling Corrector Colab

2.1.2 ***Pattern detection.***

At the time of preprocessing information and depending on the objective sought and the collection it is necessary to locate certain types of patterns that may provide us useful knowledge or maybe need to be removed. In this case we focus on retweets, censured words that appear like "&#12345", urls, hashtags, emojis, unseful chars, contractions or mentions. In order to carry out this task, we used *re library* to define regular expressions.

- user_re = '@[a-zA-Z0-9_]+:'
- mentions_re = '@[a-zA-Z0-9_]+'

Table 1: labeled_tweet.csv

count	hate_speech	offensive_language	neither	class	tweet
3	0	0	3	2	!!! RT @mayasolovely: As a woman you shouldn't...
3	0	3	0	1	!!!!! RT @mleew17: boy dats cold...tyga dwn ba...
3	0	3	0	1	!!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...
3	0	2	1	1	!!!!!!!!!!! RT @C_G_Anderson: @viva_based she lo...
6	0	6	0	1	!!!!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...

- enlaces_re = 'http\S+www\S+[A-Za-z0-9\/.:]* \.com(\.[A-Za-z]+)*'
- hashtag_re = '#[a-zA-Z0-9_]+'
- rt_re = '\!* RT'
- censured_re = '&#\d+'
- number_re = '\d+ '

To check and replace emoticons and contractions, we have used python dictionaries.The contractions dictionary was extracted from here, that was updated with *contractions*, a python library that has its own shrink dictionary. On the other hand, the dictionary used for the emoticons was built manually by us after looking at the data.

2.1.3 ***Replace/remove of contractions, emoticons and others elements.*** As already mentioned there are some elements that do not provide useful information or need to replace some examples is emojis like ":)" that is replace by "smile" or "you're" by "you are" in order to obtain useful information. That's why we have to tokenize the tweets, splitting word by word each tweet. This will help remove all capital letters, unseful chars and repeated following words.

2.1.4 ***Lemmatize terms.*** Then, we lemmatize all terms to obtain original word. Lemmatization is the process of grouping together the different inflected forms of a word so they can be analysed as a single item. Lemmatization is similar to stemming but it brings context to the words. So it links words with similar meaning to one word.

2.2 Vectorization

Once we have all the tweets preprocessed, we vectorized every tweet following different configurations. These configurations are as follows:

2.2.1 ***TFID.*** TFID or tf-idf is short term for term frequency-inverse document frequency. It is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is used as a weighting factor in information retrieval and text mining. The tf-idf value increases in proportion to the number of times a word appears in a document. It is offset by the frequency of the word in the corpus, that helps to adjust for some words which appear more frequently in general.

2.2.2 ***TFID + N-grams.*** To the above configuration we can introduce an extension, adding N-grams. N-grams is a contiguous sequence of N items from a given sample of text or speech. In our case, this items are the words of the tweet, and we have only considered the N-grams composed of 1, 2 and 3 words due to the great cost in memory and execution time involved in working with a greater range of N-grams.

2.2.3 ***TFID + N-grams + POS Tagging.*** After obtain every element from the vocabulary of the vectorizer, we can calculate the POS Tags for each of this elements of the vocabulary and build a dictionary which contains the POS Tag of each one of this elements. Using this dictionary, we add to the previous configuration columns that contains the number of occurrences of each POS tag in the tweet.

2.2.4 ***TFID + N-grams + POS Tagging + Other Features.*** Finally we can add to the configurations already seen, features that we have been calculating in previous steps, like the number of sentences, retweets, words, curse (censured words by Twitter), urls or hashtag. In addiction we add 2 extras columns "ngram_value", where if any n-gram detected in a tweet is found in the file "hatred n-gram dictionary", the corresponding associated weight should be included as a feature, otherwise 0, and "sentiments".

Sentiment Intensity Analyzer
Sentiment analysis has a wide appeal as providing information about the subjective dimension of texts. It can be regarded as a classification technique, either binary (polarity classification into positive/negative) or multi-class categorization (e.g. positive/neutral/negative). The methods required was given in NTLK Library.

2.3 Feature Selection

Before continuing to apply the supervised algorithms, we have to face a problem, the high dimensionality of the data. To face this problem, it has been decided to select the best characteristics among those that appear in the pre-processed tweets, discarding 70% of the characteristics used in the configuration.

For this task, the SelectKBest model was used, which can be found in the 'feature_selection' package of the Scikit Learn library. This model selects characteristics according to the score of each one of the characteristics with respect to a specific score. In this case, we have used chi2 as score to select the best features.

2.4 Classification Algorithms

Using the selected best features, let's use 2 classification algorithms to classify the tweets according to the 'class' field, which values means:

(0) hate speech
(1) offensive language
(2) neither

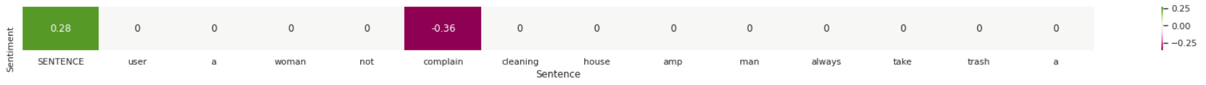

Figure 1: Sentiments Intensity from one tweet

For this purpose, we'll split our data, using 70% of the dataset for training and the remaining 30% for testing, tuning the different parameters by a cross validation.

- **SVM**
  Support vector machines (SVMs) are a particularly powerful and flexible class of supervised algorithms for both classification and regression. In this section, we will develop the intuition behind support vector machines and their use in classification problems.
  As part of our disussion of Bayesian classification, we learned a simple model describing the distribution of each underlying class, and used these generative models to probabilistically determine labels for new points. That was an example of generative classification; here we will consider instead discriminative classification: rather than modeling each class, we simply find a line or curve (in two dimensions) or manifold (in multiple dimensions) that divides the classes from each other.
- **NAIVE BAYES**
  Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable. The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of P(Xi|y).

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1, \ldots, x_n \mid y)}{P(x_1, \ldots, x_n)} \quad (1)$$

  In spite of their apparently over-simplified assumptions, naive Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering. They require a small amount of training data to estimate the necessary parameters.
  - GaussianNB
    GaussianNB implements the Gaussian Naive Bayes algorithm for classification. The likelihood of the features is assumed to be Gaussian:
  - MultinomialNB
    MultinomialNB implements the naive Bayes algorithm for multinomially distributed data, and is one of the two classic naive Bayes variants used in text classification.
  - ComplementNB
    ComplementNB implements the complement naive Bayes (CNB) algorithm. CNB is an adaptation of the standard multinomial naive Bayes (MNB) algorithm that is particularly suited for imbalanced data sets.
  - BernoulliNB
    BernoulliNB implements the naive Bayes training and classification algorithms for data that is distributed according to multivariate Bernoulli distributions; i.e., there may be multiple features but each one is assumed to be a binary-valued (Bernoulli, boolean) variable.

	precision	recall	f1-score	support
0	0.48	0.08	0.14	427
1	0.92	0.95	0.93	5747
2	0.76	0.89	0.82	1261
accuracy			0.89	7435
macro avg	0.72	0.64	0.63	7435
weighted avg	0.87	0.89	0.87	7435

Table 2: : SVM Results

3 EXPERIMENTS AND RESULTS

However, in practice, it is still difficult to analyze sentences correctly in term of syntax and semantics.Our experiments can be divided into two parts:

3.1 SVM

We have used Support Vector Machines as the first algorithm to classiffy tweets. In order to find the best parameters for this algorithm, we have used Grid Search with cross-validation, which will allow us to explore combinations of this parameters. After this Grid Search, we used the best estimator found to predict over test data. Using the classification report provided by Scikit Learn, we show the performance of our model at predicting in Table 2.

We also can see the confusion matrix in the Figure 2, which shows us the relation between predicted labels by the model and the true labels.

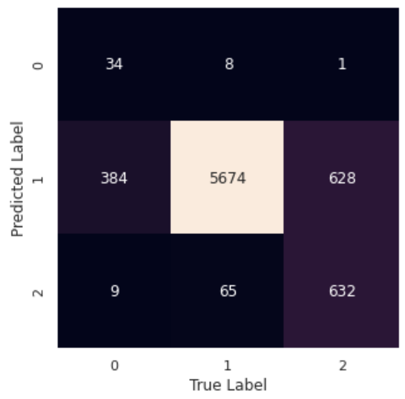

Figure 2: SVM: Confusion Matrix

As we can see in the matrix, the model tends to predict that the most of the tweets belong to the label 1 (Offensive language).

	precision	recall	f1-score	support
0	0.56	0.70	0.62	427
1	0.93	0.93	0.93	5747
2	0.85	0.77	0.81	1261
accuracy			0.89	7435
macro avg	0.78	0.64	0.80	7435
weighted avg	0.9	0.89	0.89	7435

Table 3: : Naive Bayes Bernoulli Results

Class 0, which is a minority in our data set, is where our model fails the most, predicting up to 384 tweets belonging to this class as offensive language. This result is normal, since even for any human being it can be difficult to differentiate between the term offensive language and hate speech.

3.2 NAIVE BAYES

SVM results aren't bad, but we are going to improve it. For this porpuse, we are going to use Naive Bayes models, which are a group of fast and simple classification algorithms that are often suitable for very high-dimensional datasets like ours.
As we already mencioned the appearance in the dataset of the classes is unbalanced, since there are many values "1" and few of "2" or "0".
That's the reason to use SMOTE (Synthetic Minority Oversampling Technique), it's synthesize elements for the minorities classes, in the vicinity of already existing elements. SMOTE algorithm works in 4 simple steps:

(1) Choose a minority class as the input vector
(2) Find its k nearest neighbors
(3) Choose one of these neighbors and place a synthetic point anywhere on the line joining the point under consideration and its chosen neighbor
(4) Repeat the steps until data is balanced

After all, we are ready to prove all of naive bayes. Using the classification report provided by Scikit Learn, we show the performance of our model at predicting in Table 3.
As we can see in the matrixes, the model follows class 1 tends, however it has greatly improved in minority classes.

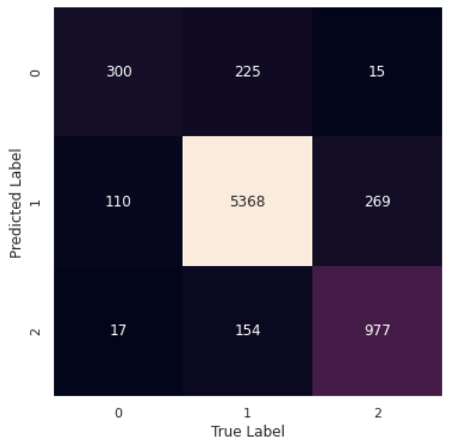

Figure 3: BernoulliNB: Confusion Matrix

Being the best model the ComplementNB where Class 1, which is a mayority in our original data set, is where our model fails the most, predicting up to 168 tweets as hate speech and 308 tweets as neither.

4 CONCLUSIONS

In conclusion, natural language processing is an arduous, time-consuming task due to the high dimensionality associated with it. Most of the process is focused on the pre-processing stage, having to process enough information to be able to extract useful information from the text. Although this is the first time we have worked with supervised algorithms using text as input, we think we have achieved quite decent results.

We believe that this work has been very useful for our future employment, since natural language processing is an increasingly important field of study, becoming of vital importance for some companies.

5 APPENDIX

You can find the code used to carry out this work in the following repository:

https://github.com/Ofeucor/Natural-Language-Processing-Machine-Learning

In addition to the code, you can also find the collection of tweets and intermediate data that will allow you to skip some steps like pre-processing.

In addition to our GitHub repository, you can view the notebooks we have used for this work online at the following link:

Natural Language Processing - Machine Learning