Data Science Lab: Process and methods

Politecnico di Torino

Project report

Student ID: s246601

Exam session: Winter 2020

First, I must mention that I have exploited multiple approaches and algorithms both for feature extraction and for building classification models; which will be explained in detail in their corresponding section. Consequently, the final selection of algorithms will be the combination of those who lead to a better model performance and higher score in the leaderboard.

1. Data exploration:

It can be seen that the data is unbalanced (consisting of 19532(68%) positive comments and 9222(32%) negative comments) which will affect the classification training process; meaning that the training will be biased towards predicting positives. As a solution, I used "Under-sampling and Bootstrap Aggregation" for building my model which is further explained in preprocessing section. Moreover, it is worth mentioning that in sentiments analysis we will face some challenges, which require attention.

One is **"Irony and Sarcasm"** where, Differences between literal and intended meaning (i.e. *irony*) and the more insulting or ridiculizing version of irony (i.e. *sarcasm*) usually change positive sentiment into negative whereas negative or neutral sentiment might be changed to positive. However, detecting irony or sarcasm takes a good deal of analysis of the context in which the texts are produced and, therefore, are really difficult to detect automatically.

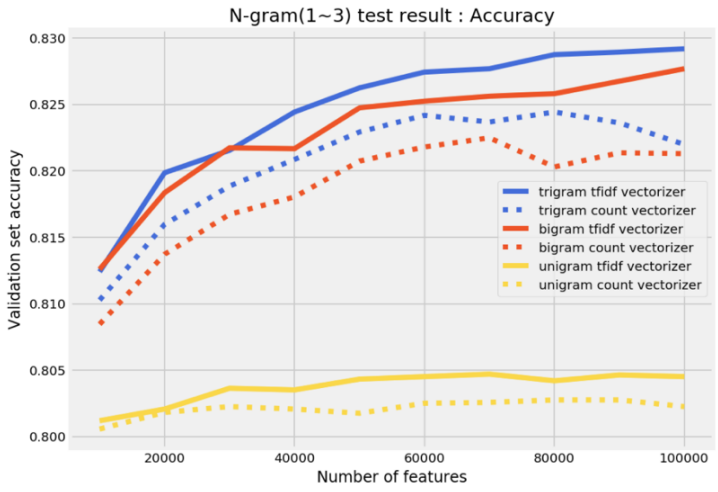
Second is "**Emojis**" where, there are two types of emojis according to Guibon et al.. Western emojis (e.g. :D) are encoded in only one character or in a combination of a couple of them whereas Eastern emojis (e.g. ¯ \ _ (ツ) _ / ¯) are a longer combination of characters of a vertical nature. Particularly in tweets, emojis play a role in the sentiment of texts. Sentiment analysis performed over tweets requires special attention to character-level as well as word-level.

In our case Emojis were transformed into their Unicode (token pattern: r'[^\s]+') to be presented in their own original figure throughout all devices. Exploiting and treating them like words could give us extra features for training our model; at first I thought it would may lead to better model. Yet, after implementation it led to a lower score. Based on several experiments I reached to a conclusion that these Emojis would not help improving the performance of the model. One reason could be since those new features are adding more dimensionality to our model it may not be beneficial.

2. Preprocessing:

- **Feature Extraction:**

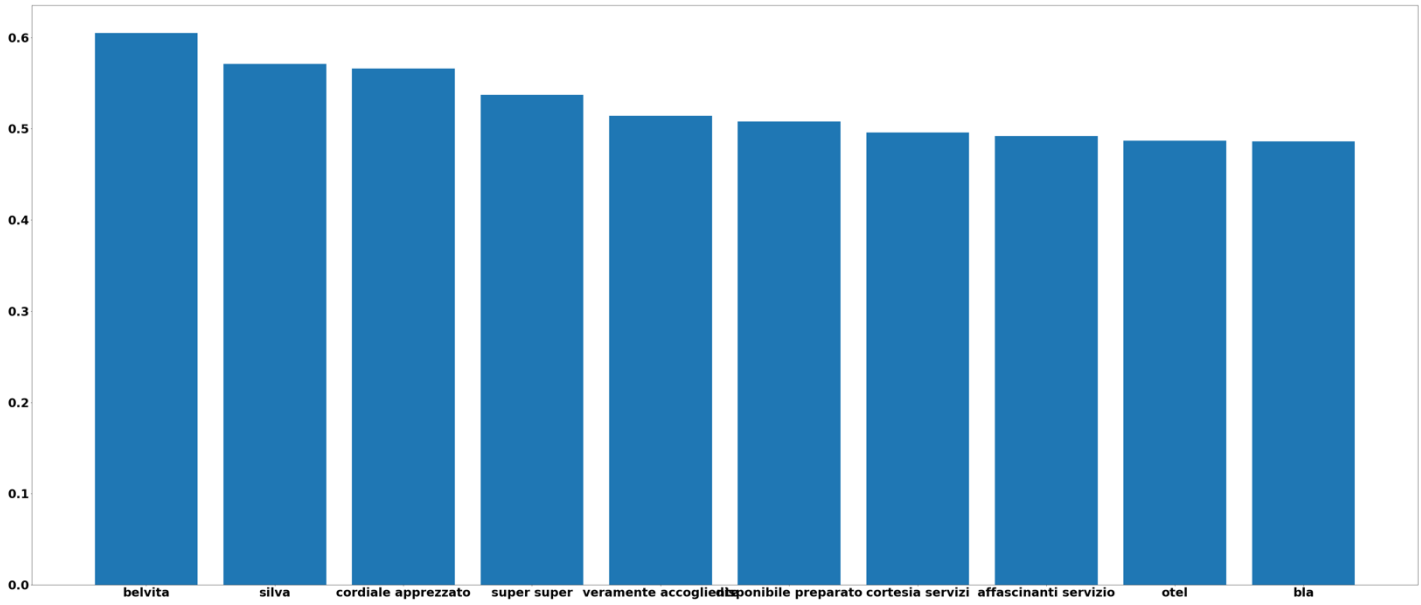
If we want to use text in machine learning algorithms, we'll have to convert them to a numerical representation. One of the methods is called bag-of-words approach. The bag of words model ignores grammar and order of words. Once we have a corpus (text data) then first, a list of vocabularies is created based on the entire corpus. Then each document or data entry is represented as numerical vectors based on the vocabulary built from the corpora.
For this there are two widely used algorithms are CountVectorizer and TF-IDF Vectorizer. Count Vectorizer is the most straightforward one, it counts the number of times a token shows up in the document and uses this value as its weight.
TF-IDF stands for "term frequency-inverse document frequency", meaning the weight assigned to each token not only depends on its frequency in a document but also how recurrent that term is in the entire corpora. The TF-IDF value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general.
In my case since the context of data are the same, TF-IDF does not show its significance. Yet, from below chart, we can see including **bigram and trigram** boost the model performance both in count vectorizer and TFIDF vectorizer. And for every case of unigram to trigram, TFIDF yields better results than count vectorizer. So, I selected TF-IDF as my feature extraction algorithm.



According to Wikipedia, “n-gram is a continuous sequence of n items from a given sequence of text or speech”. In other words, n-grams are simply all combinations of adjacent words or letters of length n that you can find in your source text.

if we set the “stopword” hyperparameter in TF-IDF, the input list is assumed to contain stop words, all of which will be removed from the resulting tokens. Because they will be considered as an outlier since they’re used more frequently and have no correlation to the labels.
By default, the TF-IDF only includes English language stopwords and since the data was in Italian language, in order to use stop words, I had to download a python package from pypi.org which included Italian stop words library and added it to the corresponding hyperparameter.

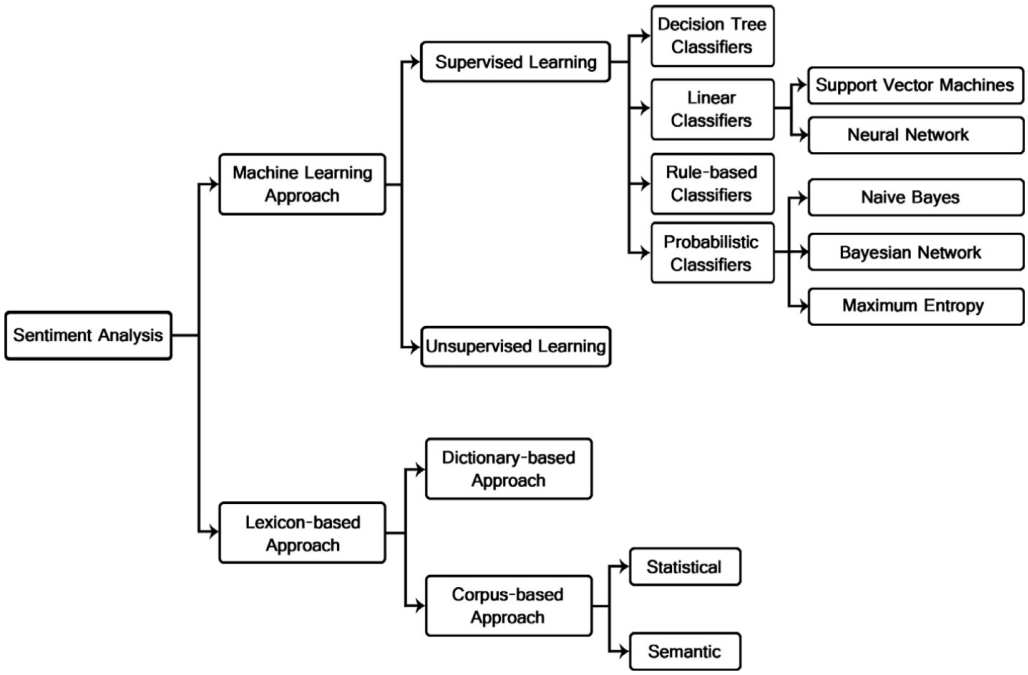
- **Keyword Extraction:**



to further analyze the dataset, I have sorted the TF-IDF vectors and extracted the 10 most frequent keywords represented above.

- **Data Cleaning:**
two different approaches were implemented. First, I removed “special characters” such as “\n” and made all the data to lower case. Second, TF-IDF itself cleans the data of Emojis, Strip_accents and stopwords.

3. Algorithm choice:



Considering what we learned in this course we must use Machine Learning Algorithms I tried to test different algorithms and select the best for the classification of our data.

- **Random forest:**
  the Random Forest classifier clearly has an upper hand with high accuracy and performance, simplicity in understanding, and improvement in results over a period of time. This makes the classifier best fit for situations like sentiment analysis. Though it **requires high training time and processing power** the improved accuracy due to aggregation of decision trees, more than makes up for other shortcomings.
  I used this first and without tuning its hyperparameters where it led to lower score result and as said since its tuning would require high training time I skipped to other approaches

- **Logistic Regression:**

  Logistic Regression is a 'Statistical Learning' technique categorized in 'Supervised' Machine Learning (ML) methods dedicated to 'Classification' tasks. using a linear regression equation to produce discrete binary outputs. And also, categorized in 'Discriminative Models' subgroup of ML methods like Support Vector Machines and Perceptron where all use linear equations as a building block and attempts to maximize the quality of output on a training set.
  This method will be very helpful for our classification task since we have only two classes and what Logistic Regression does is to simply draw a hyperplane between our point and separate them from each other

- **Support Vector Machine** (this case: Support Vector Classification **SVC**):
  he main principle of SVMs is to determine linear separators in the search space which can best separate the different classes. Text data are ideally suited for SVM classification because of the sparse nature of text, in which few features are irrelevant, but they tend to be correlated with one another and generally organized into linearly separable categories. SVM can construct a non-linear decision surface in the original feature space by mapping the data instances non-linearly to an inner product space where the classes can be separated linearly with a hyperplane.

- **Stochastic Gradient Descent (SGD):**
  SGD Classifier implements regularized linear models with Stochastic Gradient Descent. Stochastic gradient descent considers only 1 random point while changing weights unlike gradient descent which considers the whole training data. This implementation works with data represented as dense or sparse arrays of floating point values for the features. The model it fits can be controlled with the loss parameter; by default, it fits a linear support vector machine (SVM).

- After trying above mentioned approaches respectively. I saw that since I am using Linear Models for classification the score of the result cannot be improved more than acceptable threshold value of 0.968 which I achieved by implementing Logistic Regression with fine tuning which is described in the next section. Nevertheless, this must be taken under consideration that SVC is capable of applying non-linear model with tuning its kernel. However, use of RBF kernel did not improve the results. By fine tuning SGD Classifier (with Hinge loss it fits to SVM) the result improved to 0.970

4. Tuning and validation:

Logistic Regression Tuning:

1. **Regularization Parameter:**
   we use parameter C as our regularization parameter. Parameter $C = 1/\lambda$.
   Lambda (λ) controls the trade-off between allowing the model to increase its complexity as much as it wants with trying to keep it simple.
   Parameter C will work the other way around. For small values of C, we increase the regularization strength which will create simple models which underfit the data. For big values of C, we low the power of regularization which implies the model is allowed to increase its complexity, and therefore, overfit the data.
   In my case I tried many different ranges for C considering both wide and small steps to find the optimum value which would lead to the best score.

2. **Under-Sampling & Model Averaging (Voting):**
   Since our data unbalanced (30% negative and 70% positive) and unbalanced refers to highly unequal or uneven distribution of classes. We must use Undersampling technique to improve performance of our classifier. There are various methods of undersampling like Cluster Centroid based Majority Under-Sampling Technique (CCMUT). Yet, I implemented a custom bagging approach by selecting 9222 samples from the positive ones and training the model with these and all the negative sample; Repeating this by replacing the selected samples and resampling again.
   Following this we apply the Model Averaging technique. It does exactly what the name says, multiple models are trained on the same dataset, and during prediction, we take the average over multiple models. Thus, when we are predicting I extracted the class prediction probabilities to use them for averaging. The criteria for assigning the final labels is the average being above 0.5 for positive class and lower than 0.5 for negative class.
   Moreover, for selecting number of models to be averaged we started from 20 models and observed that it is improving our finals predictions so we increased the number to 300.

SGD Classifier Tuning:

To fine tune the representation and classifications together I created a pipeline of Vectorization and Classification. Then in order to find the best hyperparameters I have used GridSearch and Cross-Validation to perform an Exhaustive search over different ranges and combination of parameters. The steps of parameter tuning are as below:

Feature Extraction		Classifier Model		Score
Algorithm	Tuning	Algorithm	Tuning	
CountVectorizer	stop words	Random Forest	Estmator=100	0.886
CountVectorizer	stop words	Random Forest	Estmator=250	0.914
CountVectorizer	stop words, PCA	Random Forest	Estmator=250	0.788
CountVectorizer	stop words	Naïve Bayes		0.881
CountVectorizer	stop words	Logistic Resression		0.961
TF-IDF	stop words,ngram=(1,1)	Logistic Resression		0.964
TF-IDF	stop words,ngram=(2,5)	Logistic Resression		0.897
TF-IDF	stop words,MaxFeature=1000	Logistic Resression		0.947
TF-IDF	stop words,MaxFeature=50000	Logistic Resression		0.966
TF-IDF	stop words,ngram=(1,3)	Logistic Resression	C=12	0.96
TF-IDF	stop words,ngram=(1,3)	Logistic Resression	C=250,Undersampling,ModelAveraging	0.968
TF-IDF	stop words,ngram=(1,2),Max_df=0.8,useidf='true'	SGD with hinge loss	alpha=10e-5,class_weight=none,Max_iter=20	0.966
TF-IDF	stop words,ngram=(1,2),Max_df=0.75,useidf='true'	SGD with hinge loss	alpha=10e-5,class_weight=none,Max_iter=500	0.967
TF-IDF	stop words,ngram=(1,2),Max_df=0.75,useidf='true'	SGD with hinge loss	alpha=10e-5,classweigh=,Max_iter=500,{'neg': 2, 'pos': 1}	0.968
TF-IDF	stop words,ngram=(1,2),Max_df=0.75,useidf='true'	SGD with hinge loss	alpha=10e-5,classweigh=,Max_iter=500,{'neg': 3, 'pos': 1}	0.97

Highlighted changes created to model score and use of different Models

Conclusion and future work:

Given the results of numerous experiments that has been done, the quality of linear classification cannot be improved further; unless, we change the representation by acquiring bigger corpuses to extract better TF-IDF features and also other means of representation learning techniques such as Neural Networks (CNN or RNN) to extract discriminant features for better predictions. Doing so was considered out of scope for this course.

References:

- Stop word - Italian Library : https://pypi.org/project/stop-words/
- Guibon, Gaël, Magalie Ochs, and Patrice Bellot. "From emojis to sentiment analysis." 2016.
- Gupte, Amit, et al. "Comparative study of classification algorithms used in sentiment analysis." *International Journal of Computer Science and Information Technologies* 5.5 (2014): 6261-6264.
- Medhat, Walaa, Ahmed Hassan, and Hoda Korashy. "Sentiment analysis algorithms and applications: A survey." *Ain Shams engineering journal* 5.4 (2014): 1093-1113.