Machine Learning Nanodegree
NLP Capstone Project

Leonardo Cotti
July 15, 2017

Definition

Project Overview

Natural Language Processing (a.k.a. NLP) is definitely living a momentum. This was made possible thanks to many factors such as the use of complex models (with the necessary computation capacity to make them run) and the spread of big datasets. From the many applications of NLP, sentiment analysis is constantly increasing popularity year by year[1] and it is becoming widely used to make business decisions.

Unfortunately, many of the applications of NLP in sentiment analysis are conducted in English and it is very rare to find applications in other languages. This is mainly due to the lack of large and high-quality dataset in any languages but English. For this reason, I was inspired to conduct this project in order to create a good sentiment prediction model in my native language: Italian.

In this project, I first collected a dataset crawling reviews and associated rating (discrete numbers from 1 to 5) from Amazon.it, then I used scikit-learn on python to create a supervised model that was successfully able to predict sentiments to reviews.

Problem Statement

The objective of this project is to create a sentiment analysis model that can predict the rating of Amazon Italian reviews. To achieve the goal, I need to create a Supervised Model where I use as independent variables the text of the reviews and as dependent variable the rating. The tasks involved are the following:

1. Scrape Amazon.it to collect reviews and ratings
2. Clean the dataset
3. Train and tune a Supervised Classifier
4. Measure the performance of the Classifier
5. Compare the Classifier with another that uses an English dataset

The process of creation and analysis of a Classifier on an Italian dataset can be useful to confirm or not the feasibility and effectiveness of NLP techniques applied to a language that is not English.

[1] https://trends.google.it/trends/explore?q=sentiment%20analysis

Metrics

The first metric I use is a confusion matrix with the following structure:

		Predicted				
		1 star	2 stars	3 stars	4 stars	5 stars
Actual Class	1 star					
	2 stars					
	3 stars					
	4 stars					
	5 stars					

Then I compute the True Positive (TP), False Positive (FP) and False negative (FN) to compute precision and recall:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

I use precision because I want to know how accurate the positive predictions are and the recall to understand how many positive predictions I can predict.

In layman's terms, the recall is the ratio of number of events that a classifier is able to correctly recall from all the correct events. The recall can be a number between 0 and 1 that represents a percentage. For instance, if we have 10 correct events and we are able to recall just 6 of them, we will have a recall of 0.6 (or 60%)

On the other hand, the precision is the ratio of events correctly recalled to all events recalled. The precision is also a number between 0 and 1 that represents a percentage. For instance, if we have 10 events recalled and we correctly recall 8 of them, we will have a precision of 0.8 (or 80%).

Since I can only use one metric to tune the classifier, the ultimate metric is the F1-score: a single measure that can combine and weight equally precision and recall.

$$F_1\ score = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Apart from being a metric between 0 and 1 that capture different aspects of classifiers, the F1 Score is a metric that gives the same weight to precision and recall. This is very useful in our case where we want to test the general performance of the classifiers without giving more weight to a particular aspect of the general outcome.

Analysis

Data Exploration

The dataset has been created using the Scrapy library, implementing a two-step scraping approach.

First, I created a spider to look for amazon product ids in random categories. I let the spider run for about an hour and I collected around 1,000 unique ids. An amazon product id is a 10-character alphanumeric string (such as: B0069G5Z9K, B06Y45JZRV, B01MS14AAH) that can uniquely identify an amazon product.

Secondly, I created another spider that looked for the 1, 2, 3, 4 and 5 stars review using this groups of relative links for each product id found in the first scraper:

```
amazon.it/product-reviews/'PRODUCT_ID'/ref=cm_cr_arp_d_hist_5?ie=…filterByStar=five_star
amazon.it/product-reviews/'PRODUCT_ID'/ref=cm_cr_arp_d_hist_4?ie=…filterByStar=four_star
amazon.it/product-reviews/'PRODUCT_ID'/ref=cm_cr_arp_d_hist_3?ie=…filterByStar=three_star
amazon.it/product-reviews/'PRODUCT_ID'/ref=cm_cr_arp_d_hist_2?ie=…filterByStar=two_star
amazon.it/product-reviews/'PRODUCT_ID'/ref=cm_cr_arp_d_hist_1?ie=…filterByStar=one_star
```

Each page contained the top 10 reviews of its class. I let the scraper run for around 3 hours, so I could get a maximum of 50,000 reviews (1,000 product IDs * 5 classes * 10 reviews of each class). Because of repetitions and lack of reviews for some products, the final amount of unique reviews I collected was **11,147**.

A sample of product with 10 reviews in its 5-star class is this.
A sample of product with 6 reviews in its 2-star class is this.
A sample of product with 0 reviews in all its class is this.

Using all the 11,147 reviews collected I created JSON objects file as following:

```
{"rating": "1", "text": "Il cofanetto sara' anche bellino esteticamente,ma 290 euro??"}
{"rating": "2", "text": "Decisamente esagerato come prezzo!!!"}
{"rating": "5", "text": "Davvero un ottimo prodotto."}
```

Exploratory Visualization

Some visual representations can be of help to better understand the quality of the data collected.

Fig. 1 The following pie chart is showing the distribution of the dataset by rating. The number of 5-star and 4-star reviews account for more than 50% of the population, this is due to the inclination of users to leave 4-star rating or more to good products. 3-star rating or less is commonly used for bad experience, bad products, problems on delivery or any kind of issue with the product/service.

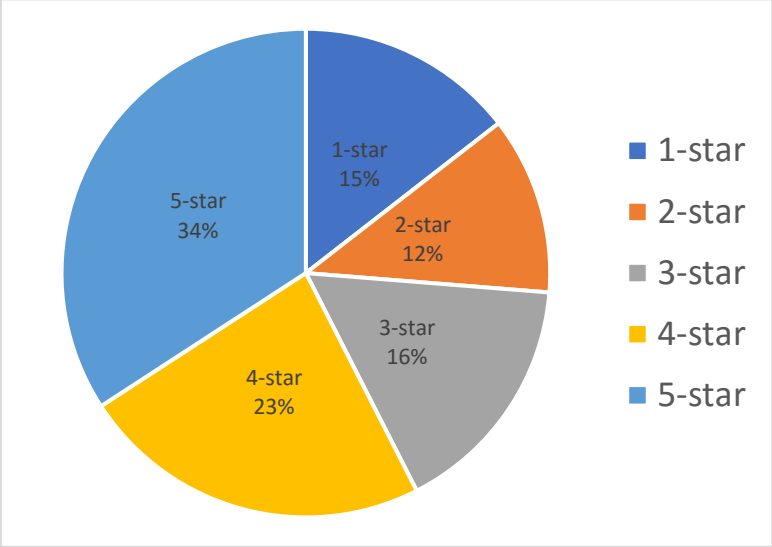
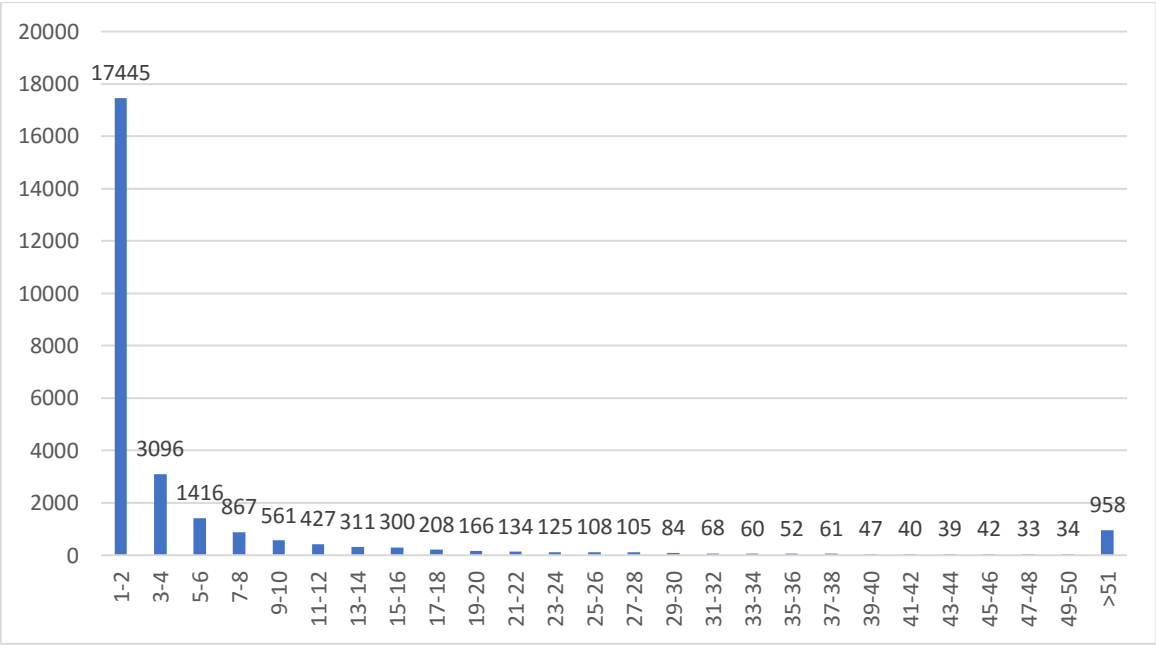


Fig. 2 The following histogram show on the x-axis the repetitions of words in the reviews and in the y-axis the number of words repeated x-axis time. It is possible to observe that the distribution is positive skewed, in fact the mean of repetition is 15 times while the mode is 1-2 repetitions. Because the number of words repeated 1-2 times is very high, it should be taken in consideration when preparing, cleaning and tuning the dataset to avoid noise. On the other hand, there are almost 1,000 words repeated at least 50 times, this is very good because these words are the ones that will probably be the feature with the highest contribution in the Classifier.



Algorithms and Techniques

In order to have a single process to transform and classify the data, I have implemented a sklearn Pipeline as following:

The **Count Vectorizer**[2] converts the text to a matrix of token counts and it could be optimized with the following parameters:

a) strip_accents: to remove or not accents in the text
b) ngram_range: indicate the range of an n-gram using a tuple, such as: (1,2), (1,3) or (1,4)

The n-gram is a contiguous sequence of n items from a given sequence of text. The count vectorizer creates an internal vocabulary to map the text. Using an n-gram range we could extend the vocabulary through combinations of the starting text. For instance, if we code the text: "This is a sample sentence" we could have the following dictionary using different n-grams:

Text	1-gram	2-gram	3-gram
This is a sample sentence	This is a sample sentence	This is is a a sample sample sentence	This is a is a sample a sample sentence

The n-gram is very useful to capture words that have a meaning combined such as phrasal verbs in English.

The **Tf Idf Transformer**[3] converts the matrix of token counts in a matrix of frequencies, instead of using the number of repetition of the words it can create a term frequency of words in the text. This is particularly useful because it scales down the impact of tokens that occur very frequently that are less informative than features that occur in a small fraction of the text. It can be optimized with the following parameters:

a) norm: to choose the kind of normalization (l1, l2 or none)[4]

The **Classifier**[5] is a Linear classifier (SVM, logistic regression, etc.) with stochastic gradient descent (SGD) training. It is a very powerful and flexible classifier with many parameters for tuning, such as:

a) loss: to change the loss function used (it allows to implement SVM, logistic regression or perceptron)
b) alpha: change the regularization constant

[2] Check the complete Count Vectorizer documentation here
[3] Check the complete Tf Idf Transformer documentation here
[4] Comparison between different normalizations here
[5] Check the complete SGD Classifier documentation here

The classifier implemented is the Stochastic Gradient Descent (a.k.a. SGD) because is able to encapsulate many loss Linear Classifier through the parameter "loss". The default loss function used is a Linear SVM that is a common used loss function in NLP. In any case, tuning the SGD Classifier through the parameter "loss" can be useful to test other loss functions and to understand if it is worth to replace the standard Linear SVM.

Since in our case we have 5 different classes, we want to have a model that can clearly select the class that is most likely to be predicted. The default Linear SVM parameter is a good choice to start because it is an algorithm that tends to separate the classes finding a border line that maximizes the distance between the target variables. Also, Using SGD in the pipeline is very convenient because it allows to change, through tuning, the Linear Classifier inside the pipeline without creating a new pipeline for each Linear Classifier we want to test.

To deepen about the parameters' meaning of the elements in the pipeline check the documentation in the notes.

Benchmark

Even if there are not implementations in the Italian language of text classifiers and it is difficult to find a direct comparable benchmark, I will compare my outcome against two main benchmarks:

- A Random Agent Classifier
- An English equivalent Classifier

The Random Agent Classifier is the Classifier that simulate a random guess, for instance if we have 5 classes the accuracy of the random agent should be 20%.

The English equivalent classifier is a classifier created with the same Pipeline but using an equivalent dataset (same number of reviews and kind) in English created starting from this webpage:

- http://jmcauley.ucsd.edu/data/amazon/

Methodology

Data Preprocessing

The preprocessing is in the first part of the “sentiment_analysis” notebook and it consists of the following steps:

1. Read the JSON file “reviews.jl” and to load it in memory
    - Number of records: 11,147
2. Store the content of the single JSON records in two different variables:
    - The reviews as String vectors in the X_text vector
    - The ratings in the y_text vector
3. Create a label encoder able to convert in both ways the ratings into numeric vectors
4. Shuffle and split the dataset into training set and test set
    - X_train and y_train size is 10,032 (90% of the original dataset)
    - X_test and y_test size is 1,115 (10% of the original dataset)

Because I don’t have a huge amount of reviews, I have chosen to retain the 90% of the dataset for the training set. Even if the test set is much smaller than the training set, the 10% of the data is equivalent to more than a thousand records and it is big enough to validate properly the model.

Tab 1 The following table shows three random records’ examples of X_text and y_test.

index	y_text[index]	X_text[index]
3	5	e' un ottimo traduttore. Eccezzionale il fatto di poterci scrivere in cinese. se si e' spesso in cina e' indispensabile.
20	4	Non ho dato il massimo di valutazione perchè non ho ancora fatto il backup del vecchio hard disk e quindi non l'ho messo alla prova. Devo soltanto dire che la consegna è stata perfetta, come sempre, rispetta al massimo la descrizione del prodotto e poi devo aggiungere che difficilmente potrò dare un giudizio negativo,perchè il samsung SSD 850 EVO è uno dei migliori. A lavoro finito ,nel caso dovessi trovare dei difetti, aggiungerò immediatamente le note negative.
450	3	Non ha la stessa resa che ha il "leggi tutto" che tutti conosciamo....VLC. Spero che ci siano aggiornamenti che lo rendano piu fruibile. Per ora poco utilizzo su Fire 2015. Confido le cose cambino presto!!

Tab 2 The following table shows the logic behind the label encoder, in particular it is important to highlight the different type of the label before and after the transformation.

Label	1	2	3	4	5
Type of Label	String	String	String	String	String
Transformed	0	1	2	3	4
Type of Transformed	int64	int64	int64	int64	int64

Implementation

The implementation is divided in two Jupiter notebook files:

- sentiment_analysis.ipynb
- sentiment_analysis_2classes.ipynb

The implementation in the "sentiment_analysis.ipynb" notebook consists in the creation of 5 classifiers:

- Italian Classifier
- Italian Tuned Classifier
- Random Agent Classifier
- English Classifier
- English Classifier Big Dataset (trained from a big dataset)

The **Italian Classifier** is the standard classifier used in all the analyses and it is benchmarked against all the other classifiers. It is created using and implementing a pipeline as reported in the *Algorithm and Techniques* section.

The **Italian tuned Classifier** is an Italian Classifier tuned using GridSerchCV from sklearn in order to find the best mix of hyper-parameters that maximize the F1 Score.

The **Random Agent Classifier** is a classifier that simulate random guesses. Since we implement a random agent that can predict 5 kinds of labels, the expected value of precision and recall should be 0.2 (or 20%) and then the expected F1 Score should be 0.2 with a small margin of error due to the not uniform partition of the dataset (see Fig 1).

The **English Classifier** is the English counterpart of the Italian Classifier. It is trained and tested using a dataset created from the link reported in the *Benchmark* section. The English dataset has the same number of records (11,147) and diversity of the Italian Dataset.

The **English Classifier Big Dataset** is analogous of the English Classifier but it uses a dataset that is 10 times larger (111,470 records) than the Italian Classifier.

After creating and training all the classifiers, they are compared using Confusion Matrix, Precision, Recall and F1 Score.

The main steps of the "sentiment_analysis.ipynb" notebook are the following:

- Load the Italian dataset and put the reviews in the X_text variable and the ratings in the y_text variable
- Create a label encoder to encode and to decode the y_text
- Split the X_text and the y_text in training and test set
  - 90% of the data in the training set
  - 10% of the data in the test set
- Create a pipeline object as described in *Algorithms and Techniques* section
- Fit the classifier with the training set (X_train_text and y_train_text)
- Make prediction using the test set (X_test_text and y_test_text)
- Compute Confusion matrix, Precision, Recall and F1 Score using the previous predictions
- Tune the classifier using GridSearchCV
  - It allows to train the classifier using different combination of parameters and select the combination that maximize the F1 Score
- Create a Random Classifier that guesses randomly the prediction of the classess

- Create one more classifier using the same procedure as the Italian Classifier loading the English dataset
- Create another classifier using the same procedure as the Italian Classifier loading the English Big dataset
- Compute F1 Score of all the previous classifiers
- Print a table to sum up the F1 Score for all the classes

The implementation in the "sentiment_analysis_2classes.ipynb" notebook consists in the creation of a classifier that has only 2 kinds of labels. As it is possible to view in Fig 1, the 4 and 5 stars reviews account for 57% of the reviews and the 1 to 3 stars reviews account for the 43%. This leads to the conclusion that we could have two sets of positive reviews (4-5) and negative reviews (1-3). The classifier created in this notebook is compared against the original Italian Classifier.

The main steps of the "sentiment_analysis_2classes.ipynb" notebook are the following:

- Load the Italian dataset and put the reviews in the X_text variable and the ratings in the y_text variable
- Reduce the classes in y_text renaming:
    - 1, 2 and 3 star label -> 1-3 Star
    - 4 and 5 star label -> 4-5 Star
- Create a label encoder to encode and to decode the y_text
- Split the X_text and the y_text in training and test set
    - 90% of the data in the training set
    - 10% of the data in the test set
- Create a pipeline object as described in *Algorithms and Techniques* section
- Fit the classifier with the training set (X_train_text and y_train_text)
- Make prediction using the test set (X_test_text and y_test_text)
- Compute F1 Score using previous predictions

During the development of the notebooks I have faced some complications when using GridSearchCV to tune the classifier. Given the nature of the GridSearchCV algorithm, the number of combinations of parameters could rapidly grow exponentially. GridSearchCV tries all the combinations and in my case I have 3 x 3 x 3 x 3 x 3 = 243 total combinations. This means that the running time should be around 243 times the training time of the original pipeline. It took around 15 minutes on my computer that is still an acceptable time for the scope. To reach this low enough number of combinations I have previously used GridSearchCV on single parameters to find the most important ones and the best values to try. This dramatically reduced the number of combinations and led to have a feasible running time.

Refinement

The Italian Tuned Classifier was boosted by tweaking the following parameters and values:

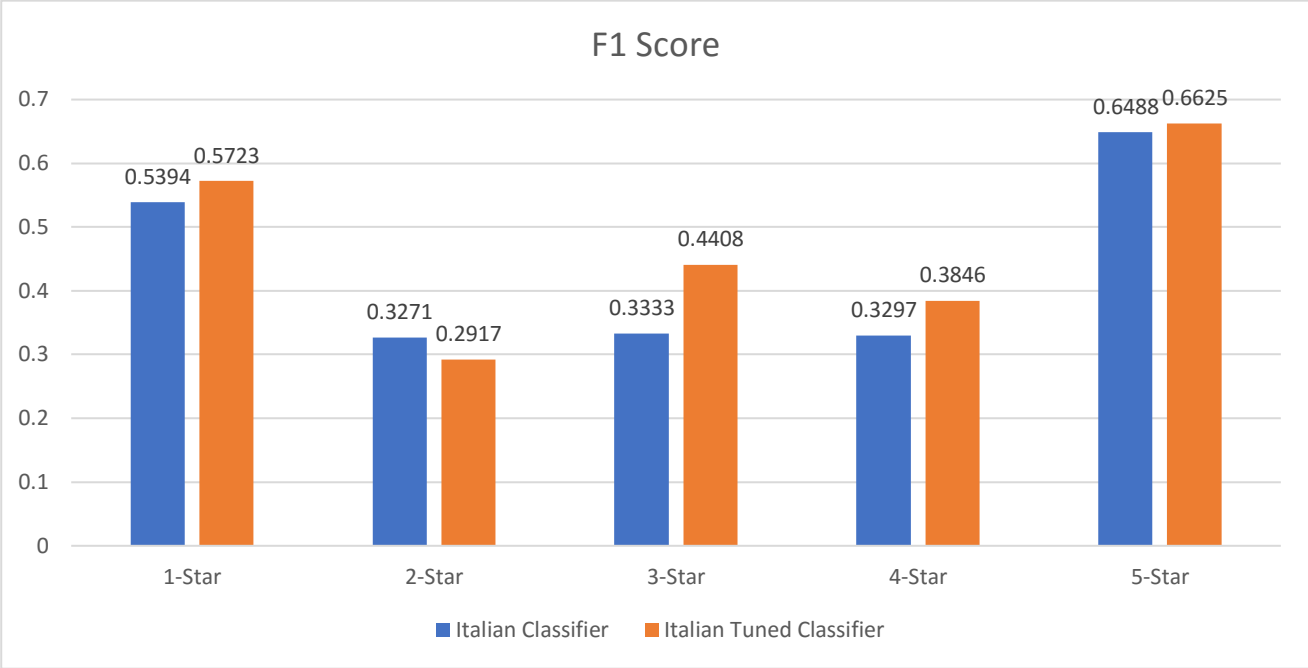
- ngram_range: (1, 2), (1, 3), (1, 4)
- strip_accents: ascii, unicode, None
- norm: l1, l2, None
- alpha: 0.0001, 0.0003, 0.001
- loss: hinge, squared_hinge, perceptron

The best parameters found by GridSearchCV are:

- ngram_range: (1, 3)
- strip_accents: unicode

- norm: l2
- alpha: 0.0003
- loss: hinge

Fig 3 The following figure shows the F1 Score of the 5 labels for the Italian Classifier and the Italian Tuned Classifier. There is a general trend to improve the score, in particular for the 3-star label, apart from the 2-star label that suffers a slightly worse performance.



Despite the Tuned Classifier shows an average slightly improved performance, it is better to use the standard one as the standard Classifier to benchmark. This will allow a more neutral comparison against other classifiers and datasets.

Results

Model Evaluation and Validation

The first validation of the Italian Classifier was made using the test set (10% of the original dataset), to have a better view of the results I have computed the confusion matrix, precision, recall and F1-score.

Tab 3 The following table is the confusion matrix for the 5 kinds of labels.

	1-Star	**2-Star**	**3-Star**	**4-Star**	**5-Star**
1-Star	89	20	8	11	24
2-Star	44	35	21	14	21
3-Star	31	10	50	37	55
4-Star	8	8	25	75	156
5-Star	6	6	13	46	302

Tab 4 The following table contains the Precision, Recall and F1-Score for the 5 kinds of labels computed as reported in the *Metrics* section.

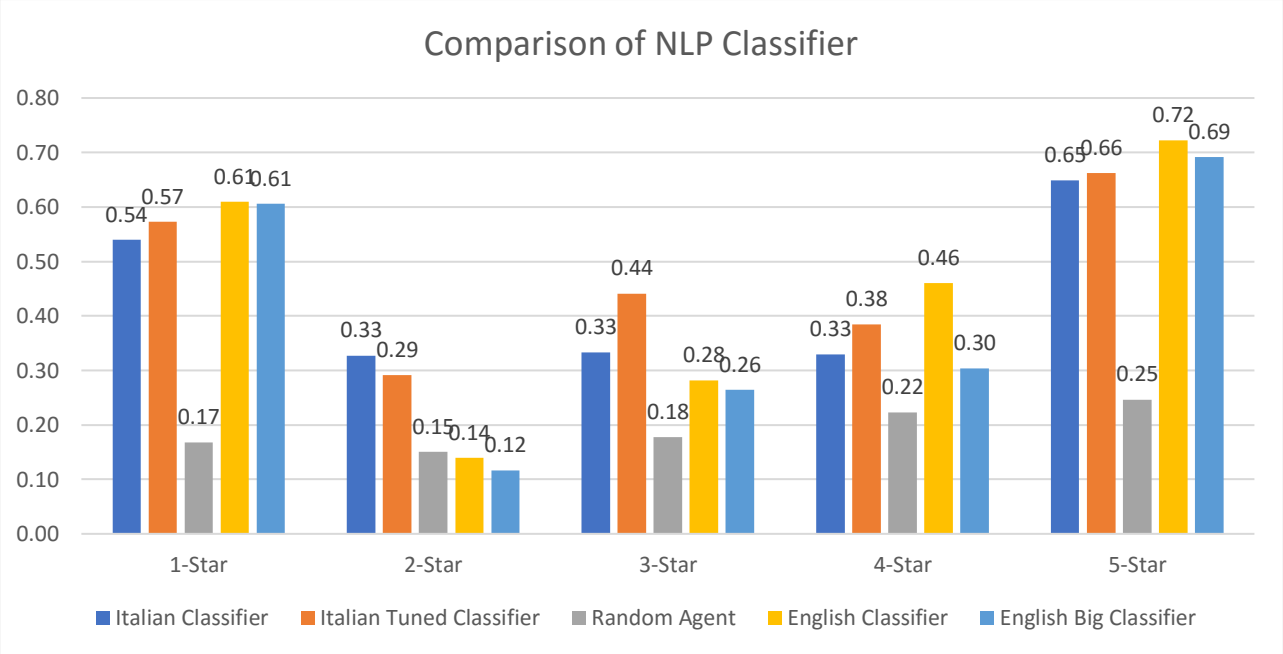
Label	Precision	Recall	F1-Score
1-Star	0,50	0,59	0,54
2-Star	0,44	0,26	0,33
3-Star	0,43	0,27	0,33
4-Star	0,41	0,28	0,33
5-Star	0,54	0,81	0,65

Despite the Precision is around 0.50 for the 1 and 5 star labels and around 0.40 for the rest, the Recall shows more divergent results. The extreme labels (1 and 5 star) have a very solid recall (0.59 to 0.81) but the labels in the middle (2, 3 and 4 stars) have a very unsatisfactory score. This is evident also in the Confusion Matrix (Tab 3) and in particular in the F1-Score. The F1-Score is a solid indicator to highlight the good general performance of the 1 and 5 star labels and the mediocre results of the 2, 3 and 4 star labels.

Justification

As seen previously, the F1 Score is a solid metric to measure and compare the performance of a multi class Classifier. That is why I have used it to benchmark the Italian Classifier again other models.

Fig 4 The following figures show the comparison of the 5 classifiers described in the *Implementation* section.



The first thing to highlight is the comparison with the Random Agent. As previously predicted the random agent's F1 Score is around 0.2 with small fluctuations. This means that the Italian Classifier is undoubtedly better than random guesses for every label, even in the labels where does not perform at best (2, 3 and 4 star labels).

The second thing to analyze is the comparison against the English Classifier. The English classifier is a classifier trained with a similar, in size and quality, dataset with English Amazon reviews (as

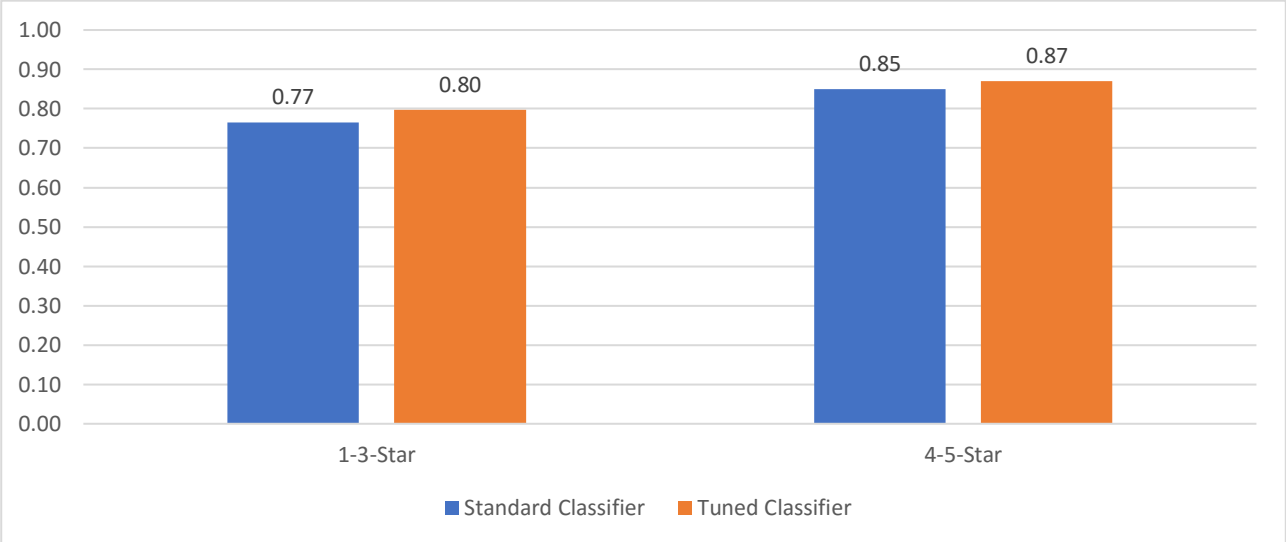
described in the *Benchmark* section). Although the English Classifier has a small score difference in the 2, 3 and 4 star labels (probably because they are the most difficult classes to predict), it has perfectly in line results for the rest. This means that the Italian Classifiers works effectively and that the machine learning algorithms work in Italian as good as English, despite the different nature of the languages.

Another very important observation is the performance of the English Big Classifier. There is a small positive and negative variation of the F1 Score in the labels, this means that increasing the dataset from around 10,000 words to 100,000 words does not increase the performance of the machine learning model.

Fig 5 The following figure shows the results of the Classifier trained from two groups of data 1-3 and 4-5 stars labels, as described in the final part of the *Implementation* section.

The outcome is surprisingly very positive. The Precision and Recall are very satisfying and it results in an excellent F1 Score.

Fig 6 The following figure shows the comparison of the 2-class classifier with a tuned version.



The previous Figure shows an even better performance tuning the 2-class classifier. The final model performs very well having an F1 Score higher than 0.80 in both labels and reaching almost 0.90 in the 4-5 Star label.

Conclusion

Reflection

This project involved many steps that could be sum up with the following:

- Identify a reliable, solid and representative Italian review dataset
- Scrape Amazon.it reviews and ratings
- Create a JSON line dataset file
- Prepare the data
- Train the Classifier
- Tune the Classifier
- Benchmark the Classifier
- Create a new 2-label dataset
- Train and tune another Classifier
- Benchmark the 2-label Classifier

The first model created was a Classifier that was able to predict if an Italian review belongs to a 1, 2 3, 4 or 5 stars group. The results were definitely much better than having a Random Agent and were comparable to other English models, but they were not good enough to have a solid predictor. On the other hand, compressing the dataset from 5 classes to 2 classes (1-3 Star and 4-5 Star) led to a truly solid predictor (F1 score > 0.80) that could compete against other predictors in the market.

I am pleased to have created a final model that can have many implementations in the real world. For instance, it can be used in Italian websites that have many product reviews but do not implement any form of rating to measure the agreeableness of products. The website's company could commission to implement the Italian Classifier to estimate quantitatively if a product is positively received or not.

Improvement

This project implements some tested NLP techniques in the Italian languages, but there are many other algorithms that can be tried such as Word to Vector and Deep Learning.

A main intuition that can be extracted from Fig 4: if we observe and compare the English Classifier with the English Big Classifier we notice that increasing the size of the dataset by a factor of 10 does not lead to better results. This means that the English Classifier suffer from high Bias, the Classifier is not complex enough to capture all the characteristics of the dataset. Given that we have used the same classifier for the Italian dataset, we could assume that the Italian Classifier suffer from high Bias too.

Having deducted that, Word to Vector (a.k.a. Word2Vec) should be the next step to implement because it is a slightly more complex model that could capture more aspects of the dataset. The following step could be to implement some complex form of Deep Learning over the Word2Vec model. However, Deep Learning could be too much complex and lead to have high variance, in that case we should also try to increase the size of the original Italian Dataset.