Document subjectivity classification on Italian news articles dataset

Oleksandr Olmucci Poddubnyy

1 Introduction

The goal of this project was to explore some approaches to perform binary document subjectivity classification using a small Italian newspapers articles dataset called SubjectivITA[1]. The dataset was composed of 74 training documents containing 1614 sentences in total, and 29 testing documents, containing 227 sentences. In order to deal with small data limitation, we've employed sBERT[2], a library containing pretrained, multi-language sentence embedding models. Using these pretrained models, one can easily produce meaningful representations for individual sentences, which in their turn can be used as features for simpler classical machine learning models to predict the document class.

2 Experiment 1

Model	Embedding Dim	Size (MB)	Average Perf
distiluse-base-multilingual-cased-v1	512	482	70.43
paraphrase-multilingual-MiniLM-L12-v2	384	418	73.80
paraphrase-multilingual-mpnet-base-v2	768	969	75.39

Table 1: Different sentence embedder models that were considered and used in our experiments.

[1] `https://github.com/francescoantici/SubjectivITA`

[2] `https://sbert.net/`

This project is open source and available at `https://github.com/alexpod1000/Document-Subjectivity`

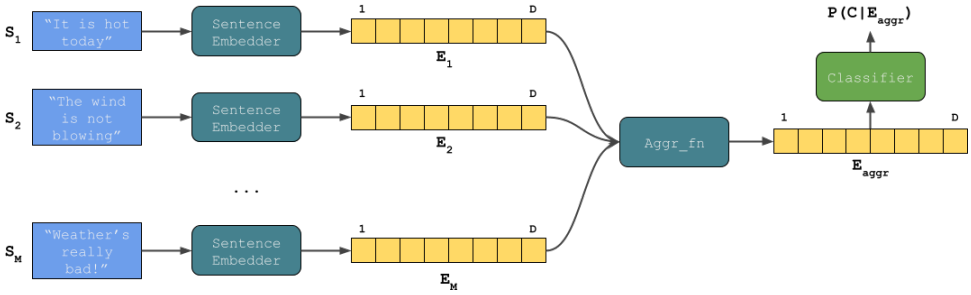

Figure 1: Sentences S_1,...,S_M are embedded using a shared and frozen sentence embedder model, producing D dimensional representations E_1,...,E_M. Those representations then get combined by an aggregation function to produce an aggregated representation E_{aggr}. Finally a trainable classification head is used on top of E_{aggr} to predict the document's class.

2.1 Setup

Our first experiment of document classification was performed using a pretrained, frozen parameters (i.e. no finetuning) sentence embedder model to encode every sequence, obtaining a latent representation of each. After embedding, all the sentence representations are combined into a single vector using some aggregation operator. In our experiments, we've used a simple mean operator, however, more complex ones such as a recurrent neural network could be used as well.
This aggregated vector can be finally used as an input to a trainable classical machine learning model (e.g. svm, decision tree, etc.) to predict the document class. The schema of our approach can be seen in Figure 1.

2.2 Results

We've tried three different document embedder models from sBERT, that were pretrained on multi-lingual data that included Italian:

1. distiluse-base-multilingual-cased-v1.
2. paraphrase-multilingual-MiniLM-L12-v2.
3. paraphrase-multilingual-mpnet-base-v2.

A comparison of those three models can be seen in Table 1.

In Figure 2 we report an average across 3 runs of accuracy and F1 score macro_avg metrics, obtained with different embedder models and different classification heads. As expected, *paraphrase-multilingual-mpnet-base-v2*, being a bigger model, has produced better representations for the classification heads to exploit.
The results for F1 score macro_avg when using *paraphrase-multilingual-mpnet-base-v2* can be also seen in Table 2.

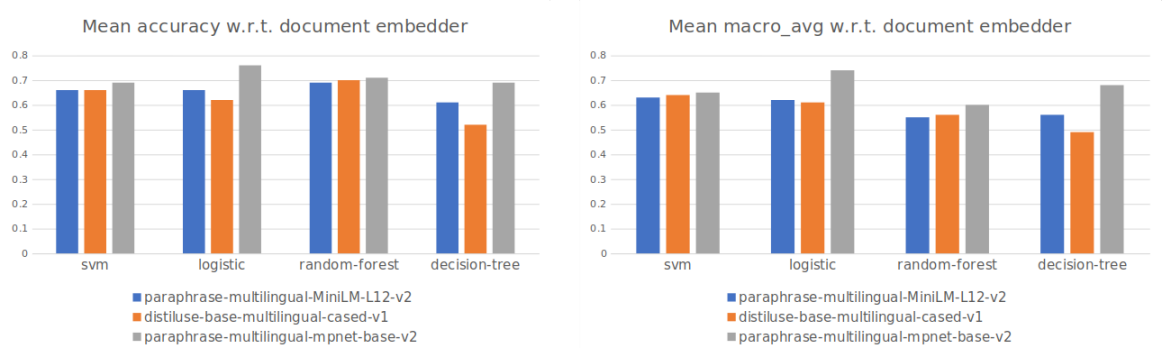

Figure 2: Results for experiment 1 in terms of accuracy and F1 score macro_avg metrics. Classification heads were trained using their default parameters.

Model	Best F1 score macro_avg
svm	0.65
logistic	0.74
random-forest	0.6
decision-tree	0.68

Table 2: F1 score macro_avg taken for each model from its best document embedder.

3 Experiment 2

The second experiment was done as an extension of the first, where we would also include statistical features given from the sentence level dataset's ground truth. More specifically we've used $\#objective/\#total$ ratio as an input feature to the model. Additionally to this ratio, we found that it was beneficial to also include the source of the article as a categorical feature.
In order to properly use article source as categorical feature, we've created $K + 1$ columns, where K is the cardinality of different train set sources, and an additional column for "unknown" sources, that do not appear in the training set.
The final feature vectors on which the models get trained are thus a concatenation of: aggregated document embedding, objectiveness ratio and article source.

Note: In a scenario with more training data available, a neural network could be used to learn an embedding matrix from category source to some smaller size dense representation. Additionally, the $\#objective/\#total$ ratio was used based on the available groundtruth data, however in a more realistic scenario, an additional sentence level network/head would need to be

used to estimate this ratio.

3.1 Results

		Objectiveness ratio + source		Objectiveness ratio only		Source usage improvement (%)	
		Accuracy	F1 Macro_avg	Accuracy	F1 Macro_avg	Accuracy	F1 Macro_avg
distiluse-base-multilingual-cased-v1	svm	0.79	0.76	0.72	0.71	0.09	0.07
	logistic	**0.86**	**0.85**	0.79	0.79	0.08	0.07
	random-forest	0.7	0.58	0.7	0.57	0	0.02
	decision-tree	0.73	0.7	**0.83**	**0.81**	-0.14	-0.16
paraphrase-multilingual-MiniLM-L12-v2	svm	**0.79**	**0.74**	0.69	0.65	0.13	0.12
	logistic	0.69	0.66	0.69	0.65	0	0.02
	random-forest	0.68	0.54	0.68	0.54	0	0
	decision-tree	0.69	0.6	**0.76**	**0.72**	-0.1	-0.2
paraphrase-multilingual-mpnet-base-v2	svm	0.76	0.73	0.66	0.6	0.13	0.18
	logistic	**0.79**	**0.78**	0.69	0.65	0.13	0.17
	random-forest	0.66	0.56	0.67	0.58	-0.02	-0.04
	decision-tree	0.76	0.73	**0.81**	**0.79**	-0.07	-0.08

Figure 3: Comparison of using document embeddings plus objectiveness ratio plus source vs document embeddings plus objectiveness ratio only, by different classification models and different embedder models.

Model	Emb + Obj + source	Emb + Obj	Obj only
SVM	0.76	0.71	**0.82**
Logistic	**0.85**	0.79	0.82
Random Forest	0.58	0.58	**0.69**
Decision Tree	0.73	**0.81**	0.66

Table 3: Comparison of F1 score macro_avg of different classifiers by taking best result (across embedders) and different training features. Last column represents the result of training a model using groundtruth objectiveness ratio as the only input feature.

As in experiment 1, we've tried three different document embedders, and for each, four different classifiers, using two different sets of features:

- Document Embeddings + Objectiveness ratio + Article source.
- Document Embeddings + Objectiveness ratio.

Moreover, as a baseline we train the four classifiers using only objectiveness ratio, without document embeddings.

The results can be found in Figure 3, and they suggest us that tree model, in particular decision tree, has performed better when given only the objectiveness ratio, together with document embeddings. If the source is also included as a feature, then the better model becomes logistic regression.
In Table 3 we can also see that using document embeddings jointly with statistical features deteriorates the performance of SVM and Random Forest models.

4 Experiment 3

In our third experiment we've unfreezed the sentence embedder model, and finetuned it using a logistic regression as a classification head. Once finetuned, the logistic regression head was removed, and replaced with other simpler classical models. In order to elegantly implement training of such a dynamic nature, where padding and masking would be required due to variable number of sentences per article, gradient accumulation technique was used (refer to Appendix A for details).

4.1 Results

Embedder	Model	Accuracy	Macro Avg	Accuracy (GA=8)	Macro Avg (GA=8)
distiluse-v1	SVM	0.62 ± 0.12	0.52 ± 0.09	0.76 ± 0.08	0.75 ± 0.14
	Logistic	0.62 ± 0.14	0.52 ± 0.10	0.76 ± 0.06	0.72 ± 0.12
	Random Forest	0.66 ± 0.06	0.57 ± 0.06	0.72 ± 0.05	0.72 ± 0.11
	Decision Tree	0.69 ± 0.11	0.60 ± 0.10	0.72 ± 0.05	0.72 ± 0.11
MiniLM-L12-v2	SVM	0.79 ± 0.07	0.76 ± 0.08	0.76 ± 0.04	0.69 ± 0.06
	Logistic	0.79 ± 0.08	0.76 ± 0.09	0.76 ± 0.05	0.71 ± 0.07
	Random Forest	0.79 ± 0.07	0.76 ± 0.08	0.72 ± 0.03	0.66 ± 0.05
	Decision Tree	0.72 ± 0.09	0.69 ± 0.09	0.76 ± 0.05	0.74 ± 0.07
mpnet-base-v2	SVM	0.66 ± 0.11	0.40 ± 0.05	0.76 ± 0.06	0.71 ± 0.10
	Logistic	0.66 ± 0.11	0.40 ± 0.05	0.76 ± 0.06	0.71 ± 0.11
	Random Forest	0.59 ± 0.06	0.48 ± 0.04	0.76 ± 0.05	0.71 ± 0.10
	Decision Tree	0.69 ± 0.10	0.65 ± 0.10	0.72 ± 0.05	0.71 ± 0.07

Table 4: Finetuning embedder model without using gradient accumulation (equal to batch size of 1), and with using gradient accumulation of 8 (simulating batch size of 8).

Embedder	Model	Accuracy (% increase avg)	Accuracy (% increase max)	Macro Avg (% increase avg)	Macro Avg (% increase max)
distiluse-v1	SVM	-0.02 ± 0.13	0.13	-0.15 ± 0.33	0.15
	Logistic	0.06 ± 0.09	0.18	-0.05 ± 0.26	0.15
	Random Forest	-0.04 ± 0.09	0.4	0.21 ± 0.22	0.28
	Decision Tree	**0.24 ± 0.13**	**0.35**	**0.19 ± 0.30**	**0.37**
MiniLM-L12-v2	SVM	0.04 ± 0.05	0.13	-0.01 ± 0.10	0.09
	Logistic	0.02 ± 0.07	0.13	-0.01 ± 0.11	0.13
	Random Forest	0.00 ± 0.07	0.08	0.11 ± 0.11	0.28
	Decision Tree	**0.11 ± 0.10**	**0.28**	**0.09 ± 0.12**	**0.34**
mpnet-base-v2	SVM	-0.05 ± 0.09	0.09	-0.12 ± 0.26	0.08
	Logistic	-0.14 ± 0.10	0.00	-0.27± 0.31	-0.04
	Random Forest	**0.03 ± 0.14**	**0.18**	**0.05 ± 0.32**	**0.37**
	Decision Tree	-0.10 ± 0.09	0.04	-0.20 ± 0.18	0.04

Table 5: Increment in percentual (w.r.t. starting metrics) for accuracy and macro_avg when finetuning using GA=8.

In Table 4 we can see results averaged over 10 runs per configuration, obtained by finetuning the document embedder for 5 epochs, using AdamW optimizer from PyTorch with learning rate of 0.0001 and weight decay of 0.001.
We can notice that using gradient accumulation, of 8 in this case, could help to stabilize the finetuning procedure, however it could decrease the average performance for some document embedders.

Additionally, in Table 5 we can see the relative improvement brought by finetuning. We report also a maximum improvement in percentage to show the most optimal finetuning case. In general we can see that finetuning can decrease the metrics, on average. We suspect this behaviour comes from having a small dataset to operate on.
Overall, since the finetuning requires very small resources, multiple models can be effortlessly trained, choosing the better performing one.

5 Conclusion

We conclude this work by presenting some considerations and further directions that we think are promising to pursue:

1. Try an ensemble of fine-tuned models: for more stability, take an ensemble where individual models have been fine-tuned on different samples, or at least in different order.

2. Try training a bigger number of models, and select a pool of better performing ones and/or more stable, to be used for actual inference.

3. Multi-task learning doesn't look promising considering the small dataset size.

4. Gather a bigger dataset of articles.

Appendices

A Gradient accumulation

Gradient accumulation is a technique that consists in accumulating gradients for N steps, before performing one optimizer step. The accumulation is usually averaged before every backward pass, in order to keep the gradients normalized.

Using this technique we can simulate the model training on larger batches, without actually having to compute a memory intensive forward pass on such large batch. The computation is instead done in multiple forward and backward passes of smaller batch size.

In case of Experiment 3, as a direct implementation using batch size N would require complicated masking and padding (due to variable number of sentences per article), gradient accumulation was used to simulate the training with regular batches of size N while maintaining the simplicity of

code that uses batch size of 1.
From the point of view of maths, we've transformed the parameter update step from:

$$\theta^{new} \leftarrow \theta^{old} + \alpha \nabla_\theta (\frac{1}{N} \sum_{i=1}^{N} L(f(x^{(i)}; \theta), y^{(i)})) \tag{1}$$

to:

$$\theta^{new} \leftarrow \theta^{old} + \alpha [\sum_{i=1}^{N} \nabla_\theta \frac{L(f(x^{(i)}; \theta), y^{(i)})}{N}] \tag{2}$$

This way, we perform backpropagation N times before performing one update step.