UNIVERSITÀ DEGLI STUDI DI BARI ALDO MORO
DIPARTIMENTO DI INFORMATICA

NLP project presentation

Sentiment analysis on movie reviews

Antonio Silletti
Elio Musacchio

Table of Contents

01

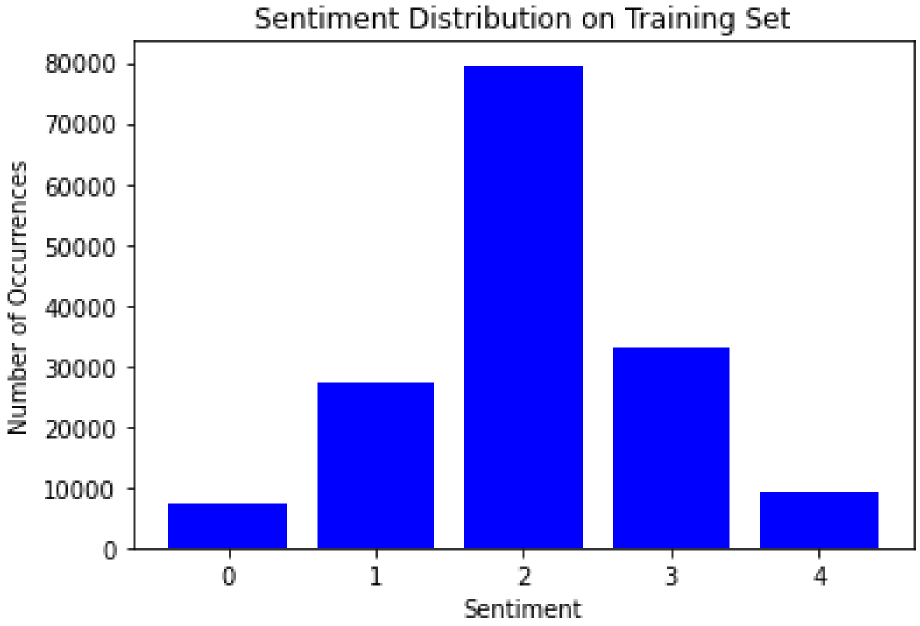
Challenge description

Challenge description

The competition consists in labeling phrases (from the Rotten Tomatoes dataset) based on the sentiment associated to them. Five values are possible for the sentiment labels:

- 0 -> negative
- 1 -> somewhat negative
- 2 -> neutral
- 3 -> somewhat positive
- 4 -> positive

Unbalanced Dataset



Dataset is unbalanced towards the ***neutral*** sentiment

Inconsistency in labeling

- The fullstop at the end lowers the sentiment?

```
77   2    is worth seeking .   3
78   2    is worth seeking     4
```

- Maybe not…

```
101 3   would have a hard time sitting through this one .   1
102 3   would have a hard time sitting through this one     0
```

Sentiment Analysis is a difficult task even for humans, but it is even more difficult on this particular dataset!

02

SVC approach

Experimental protocol

HoldOut technique with 80% of phrases «held» for the train set

(naïve approach to obtain a first feedback)

SVC approach

- Preprocessing via Spacy:
    1. Word tokenization (with tokens also being lowercased in the process)
    2. Lemmatization
    3. NER

The first two operations are trivial (they are usually applied in order to reduce dimensionality of the vector space).

As for NER, the basic idea was to replace movie titles, actor’s name, director’s name, etc. with their respective tokens, as to avoid the occurrence of words such as “good” or “bad” that do not influence the Sentiment of the phrase

```
13216   569 The first question to ask about Bad Company
```

SVC approach

Once preprocessed, we used the CountVectorizer method of sklearn in order to turn each phrase into its sparse vector representation. Furthermore, we have set CountVectorizer to consider the occurrences of each word within a sentence only once.

> Because in Sentiment Analysis task we are more interested in the occurrence of a word rather than its frequency in the corpus

Once obtained the vector representation of each phrase, we fed them to an SVC classifier with 'rbf' kernel, obtaining the following result on the test set:

03

Transformers approach

With hyperparameters search via Ray Tune

Experimental protocol

Hyperparameters search → HoldOut technique with 80% of phrases «held» on 20% of the entire dataset

Training → Kfold with K=5 for training and validation

Transformers approach

We decided to try transformers because, differently from the previous approach, embeddings obtained by a hugging face model consider *contextual information*.

Due to the above, performing preprocessing operations it's not advised, aside from the tokenization operation (which is mandatory) performed by the tokenizer associated to the chosen model.

Thanks to the addition of context, we expected better results

(and we obtained them!)

Transformers approach

We choose the following models, based on results computed on 10% of the entire dataset:

- *bert-base-uncased*
- *gpt2*
- *albert-base-v2*

We also experimented with other models (such as *roberta-base*), but we obtained slightly worse results.

The models' choice was also guided by their size:

- We wanted to perform a comparison between a small model (*albert-base-v2*), a medium one (*bert-base-uncased*) and a large one (*gpt2*)

Transformers approach

Obviously, we needed to fine tune on a downstream-task each of the chosen models.

Since Hugging Face allows to attach any head (the module responsible for obtaining results) to the models it provides, we were able to attach the **SequenceClassification** head to models trained for a different task such as gpt2, which is used for text generation tasks.

We will now see the pipeline performed to fine tune each model.

Hyperparameters search

We searched for best hyperparameters using only 20% of the whole dataset (in order to reduce computational effort).

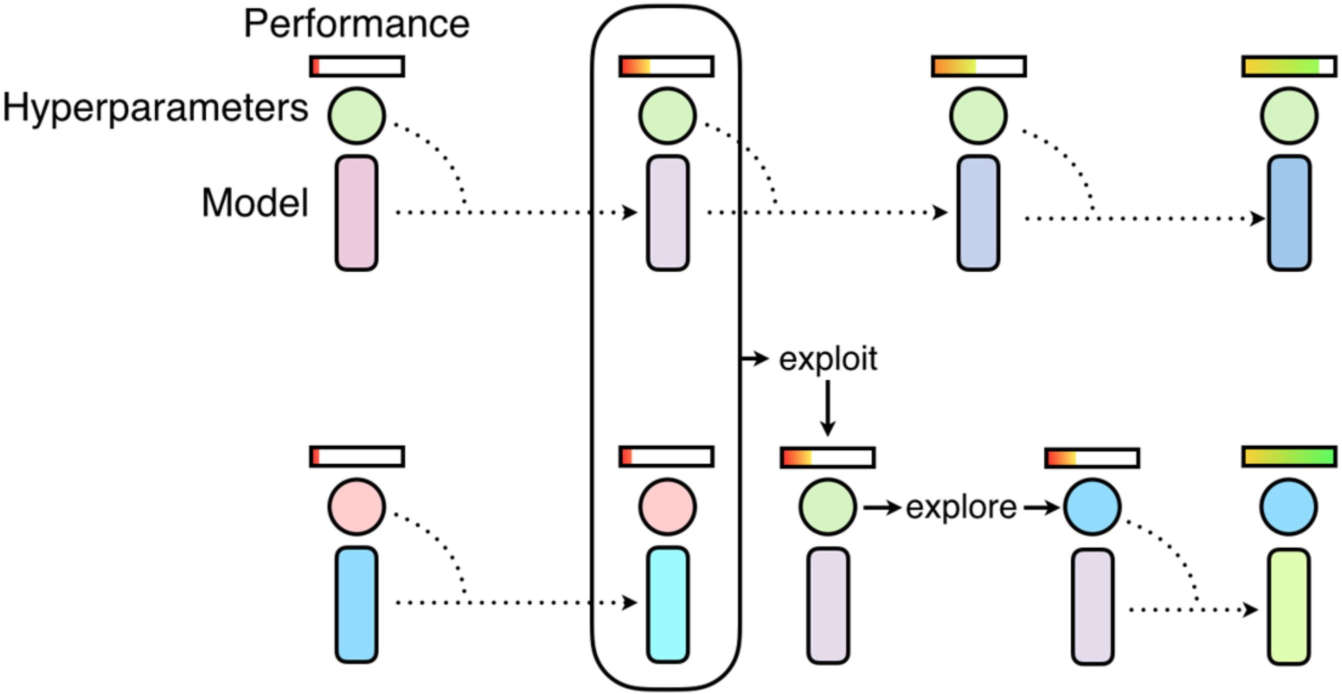
- The assumption is that the hyperparameters found are the best ones also for the rest of the dataset

The scheduler used is the **Population Based Training (PBT)**:

PBT takes its inspiration from genetic algorithms where each member of the population can exploit information from the remainder of the population

As the training of the population of neural networks progresses, parameters of the best models are copied to worse performing models, and at the same time they are randomly perturbed. This is done in order to hopefully obtain even better results than "previous generation" models.

PBT visualized



https://docs.ray.io/en/latest/tune/tutorials/tune-advanced-tutorial.html

Hyperparameters search

- Search space

```
tune_config = {
    # search space
    "per_device_train_batch_size": tune.choice([4, 8, 16, 32, 64]),
    "num_train_epochs": tune.choice([2, 3, 4, 5]),
    "seed": tune.randint(0, 43),
    "weight_decay": tune.uniform(0.0, 0.3),
    "learning_rate": tune.uniform(1e-4, 5e-5),
    "lr_scheduler_type": tune.choice(['linear', 'cosine', 'polynomial', "cosine_with_restarts"]),
}
```

- PBT also requires the definition of the Perturbation space (how the above parameters can be perturbed):
  in our case it's an exact copy of the search space

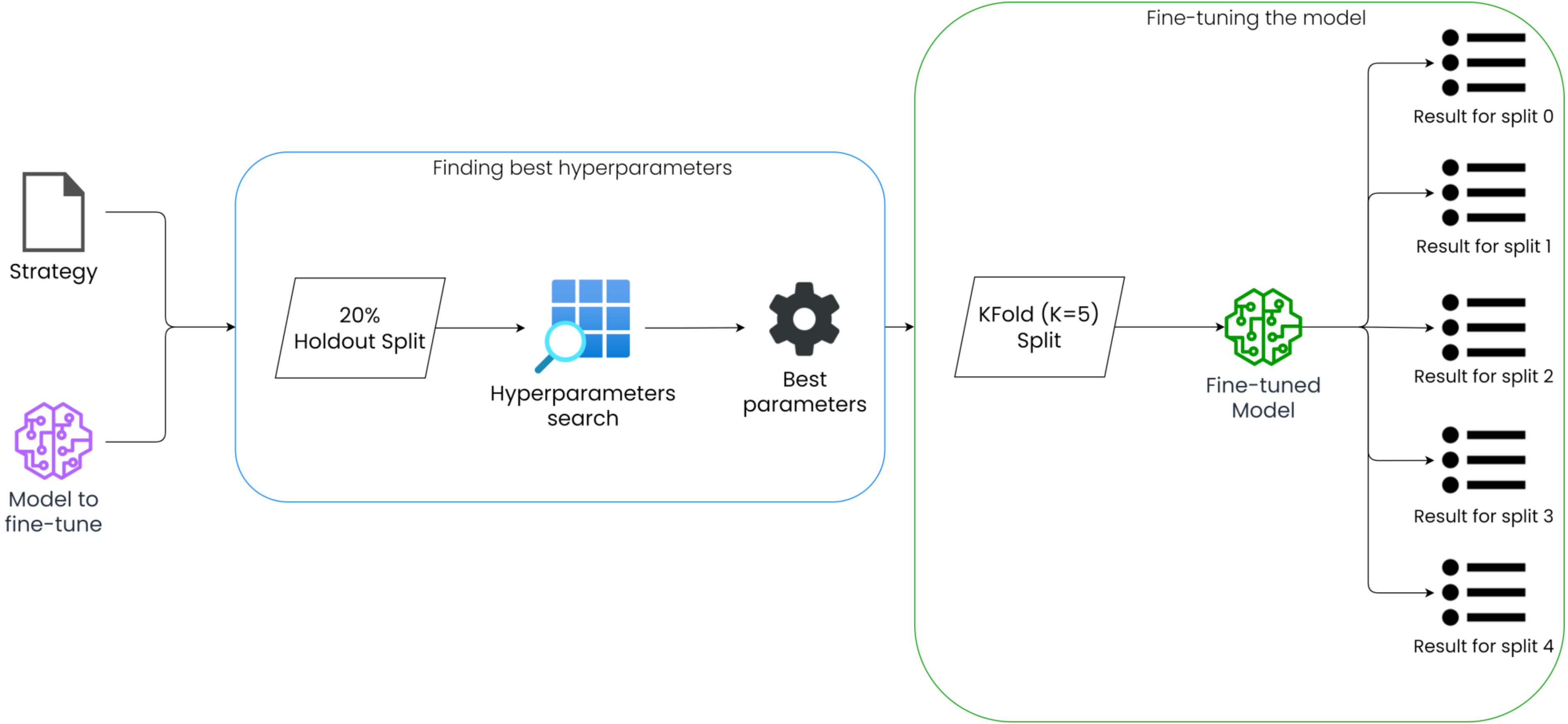
Training

Each model is fine-tuned following each of the following strategies:

- **"Only phrase"** strategy: the single phrase is fed into the model
- **"With reference"** strategy: the single phrase and the original sentence from which the phrase comes from are fed into the model
- **"With POS"** strategy: the single phrase and its pos tags are fed into the model

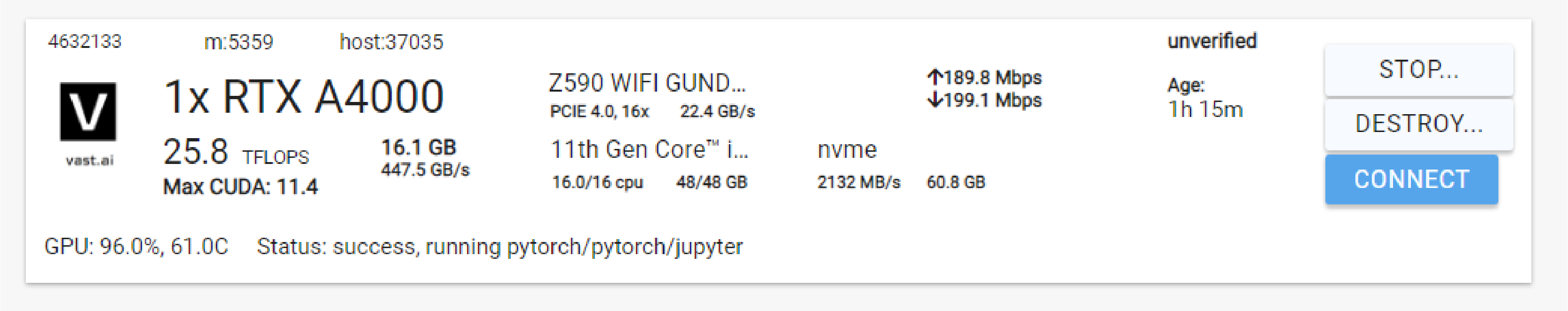
We performed KFold partitioning with K=5 and we computed results for each split for each strategy.

Whole training visualized



Training

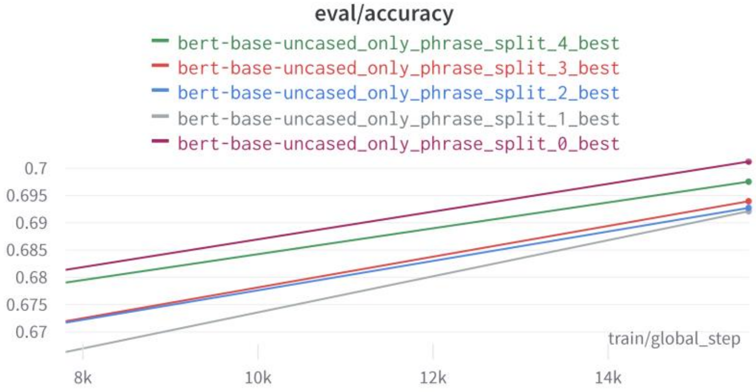
Due to the computational cost of the whole process, we rented a remote machine to perform the training:



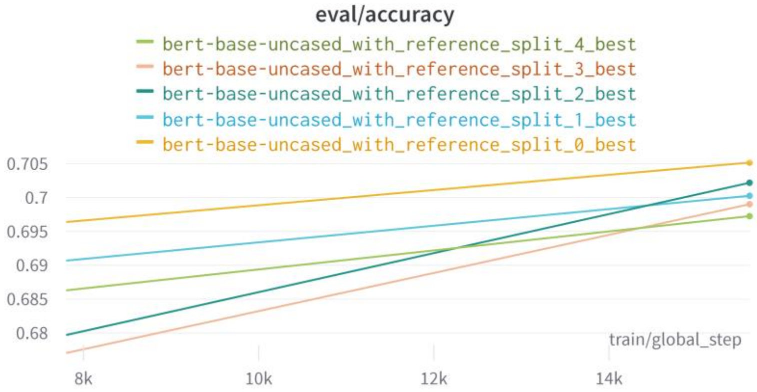
Even with this powerful machine, the complete training took roughly 2 days

Results BERT – 2 epochs

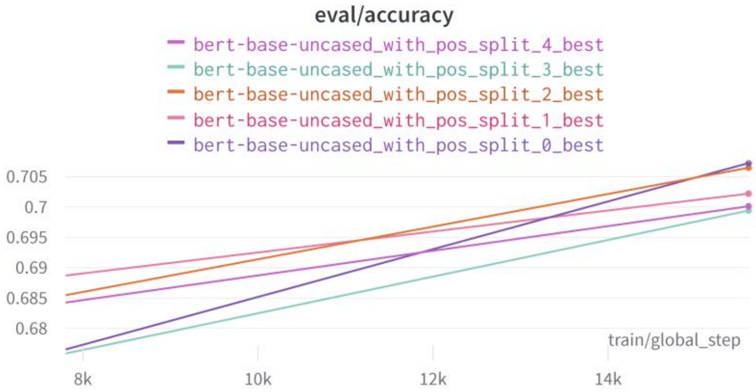
«Only Phrase» strategy



«With reference» strategy



«With POS» strategy

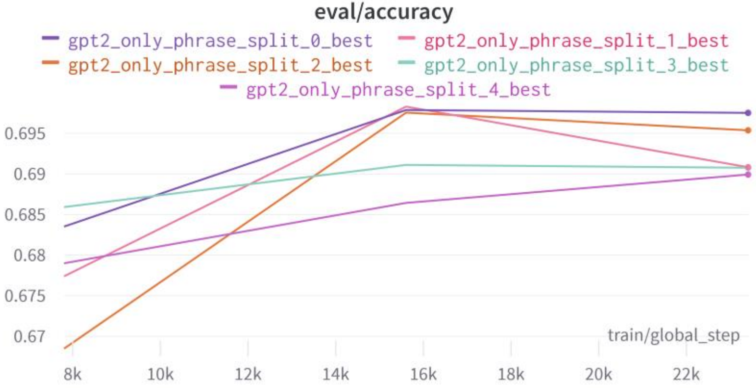


BEST results:

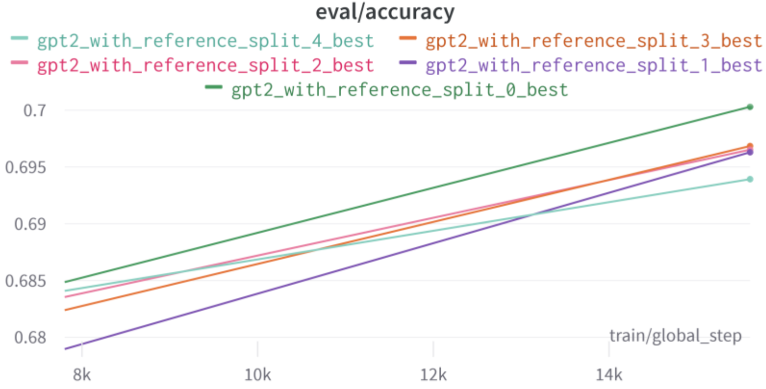
- «Only phrase» -> 0.7012
- «With reference» -> 0.7052
- «With POS» -> **0.7072**

Results GPT2 – 2..3 epochs

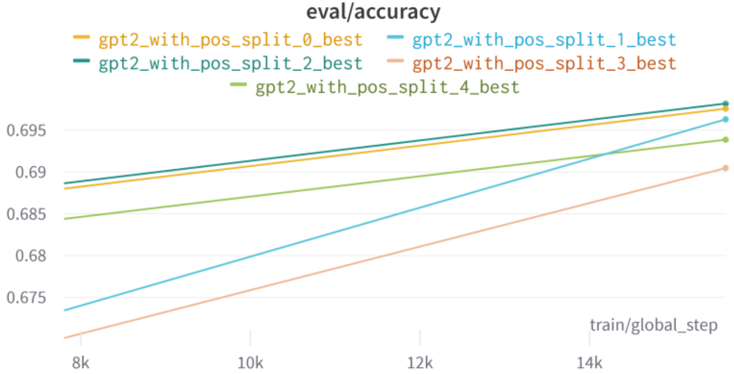
«Only Phrase» strategy



«With reference» strategy



«With POS» strategy

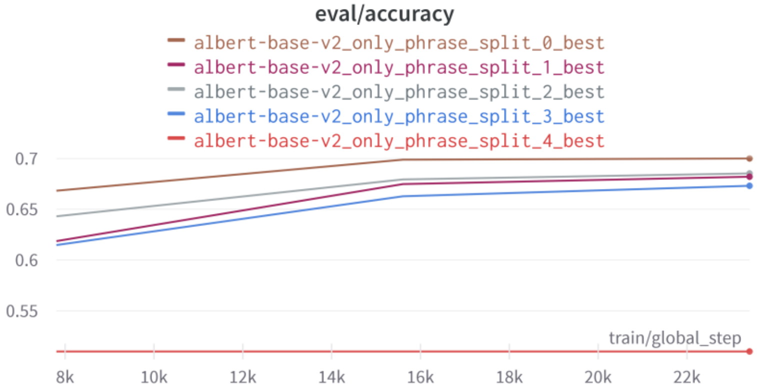


BEST results:

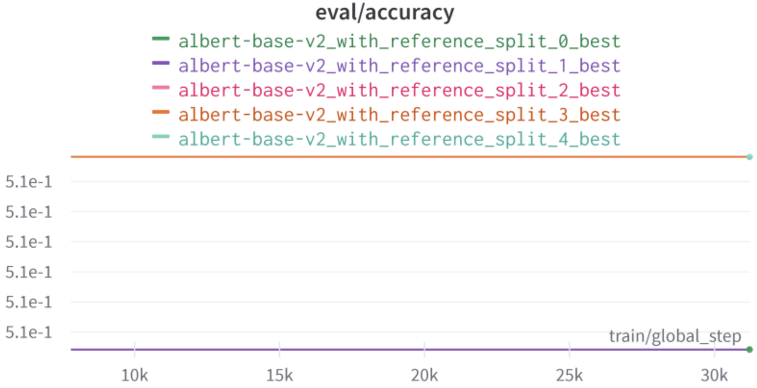
- «Only phrase» -> 0.6975
- «With reference» -> **0.7003**
- «With POS» -> 0.6982

Results ALBERT – 2..3 epochs

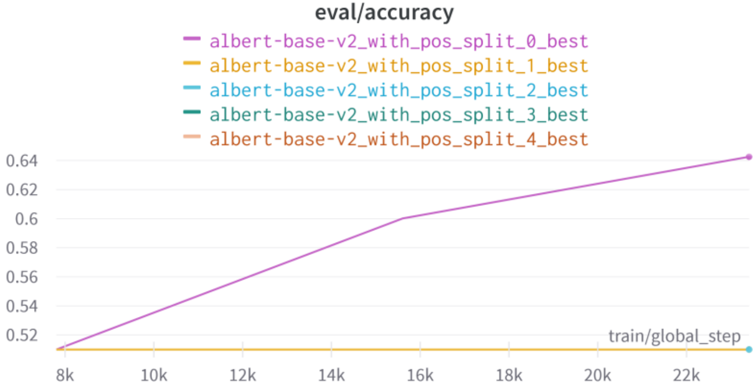
«Only Phrase» strategy



«With reference» strategy



«With POS» strategy



BEST results:

- «Only phrase» -> **0.7001**
- «With reference» -> 0.51
- «With POS» -> 0.6426

TOP-5 Results on the test set

Submission and Description	Private Score	Public Score
bert-base-uncased_with_pos_split_2.csv 2 days ago by UNIBA_NLP2122_Leshi add submission details	0.69572	0.69572
bert-base-uncased_with_pos_split_0.csv 2 days ago by UNIBA_NLP2122_Leshi add submission details	0.69469	0.69469
bert_with_pos_split3.csv 2 days ago by UNIBA_NLP2122_Leshi add submission details	0.69300	0.69300
bert_with_pos_split1.csv 2 days ago by UNIBA_NLP2122_Leshi add submission details	0.69237	0.69237
best_cm_1less_conv2d.csv 7 hours ago by UNIBA_NLP2122_Leshi add submission details	0.69224	0.69224

Final considerations

- Albert performed worse than the others (as expected) but trained much faster and produced smaller models w.r.t GPT2 and BERT
  - *1.6 gb for 5 splits vs 10 gb for 5 splits*

- GPT2 was the most consistent model across all the strategies, with its best result on the test set being **0.68975**

- The clear winner is BERT uncased with POS strategy: as a matter of fact, the models generated for each split with this combination performed better than all the other models on any strategy, and 4 out of 5 splits broke the 0.7 accuracy wall on the validation set

---

All graphs are visible at the following link

04

SBERT approach

Experimental protocol

HoldOut technique with 80% of phrases «held» for the train set

(naïve approach to obtain a first feedback)

SBERT approach

We also performed some experiments on embeddings generated using the **sentence_transformers** library.

In this case, we used the pre-trained model *all-mpnet-base-v2* which, according to SBERT documentation, provides the best quality of results.

No preprocessing operations were applied since the model required the original textual data and no fine-tuning operation was performed this time.

TOP -5 SBERT PRETAINED MODELS

Model Name	Performance Sentence Embeddings (14 Datasets)	Performance Semantic Search (6 Datasets)	Avg. Performance	Speed	Model Size
all-mpnet-base-v2	69.57	57.02	63.30	2800	420 MB
multi-qa-mpnet-base-dot-v1	66.76	57.60	62.18	2800	420 MB
all-distilroberta-v1	68.73	50.94	59.84	4000	290 MB
all-MiniLM-L12-v2	68.70	50.82	59.76	7500	120 MB
multi-qa-distilbert-cos-v1	65.98	52.83	59.41	4000	250 MB

How we used SBERT embeddings

Three different techniques were considered using SBERT-generated embeddings:

1. Clusters represented by centroids of training embeddings with the same label and classification via cosine similarity (picking the label of the cluster's centroid with highest similarity);
2. Classification via Multi Layer Perceptron;
3. Classification via Dense Neural Network.

The first two techniques didn't provide satisfactory results during the experimental protocol after different trials (the obtained accuracy on the validation set was less than 0.6).
Using a dense neural network for classification the accuracy reached 0.66 top on the test set.

05
Custom NN

Why Custom NN?

We inspected the HEAD that performs text classification provided by HuggingFace, and it basically only performs a **dropout** followed by a **dense layer** on the pooled output of the 13 layers of the encoders.

We tried to attach a much more complex Custom Head that performs classification

We tested our Custom Head on embeddings generated by SBERT (by using the all-mpnet-base-v2 model) and bert-base-uncased with POS strategy (the model which gave us the best results on the Transformers approach)

Custom NN architecture

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 13, 1, 384]	1,534
LeakyReLU-2	[-1, 13, 1, 384]	0
Conv2d-3	[-1, 13, 1, 192]	1,534
LeakyReLU-4	[-1, 13, 1, 192]	0
BatchNorm2d-5	[-1, 13, 1, 192]	26
Conv2d-6	[-1, 13, 1, 96]	1,534
LeakyReLU-7	[-1, 13, 1, 96]	0
BatchNorm2d-8	[-1, 13, 1, 96]	26
Conv2d-9	[-1, 13, 1, 48]	1,534
LeakyReLU-10	[-1, 13, 1, 48]	0
BatchNorm2d-11	[-1, 13, 1, 48]	26
Conv2d-12	[-1, 13, 1, 24]	1,534
LeakyReLU-13	[-1, 13, 1, 24]	0
BatchNorm2d-14	[-1, 13, 1, 24]	26
Dropout-15	[-1, 13, 1, 24]	0
Flatten-16	[-1, 312]	0
Linear-17	[-1, 5]	1,565

- Input is the embedding obtained by bert-uncased averaged to get a single embedding for the sentence *(dim=2)*

- The idea is to use all the layers of the encoders *(dim=1)* and not pool them or using only the last layer *(information bottleneck)*

Custom NN best-result

Submission and Description	Private Score	Public Score
bert-base-uncased_with_pos_split_2.csv 2 days ago by UNIBA_NLP2122_Leshi add submission details	0.69572	0.69572
bert-base-uncased_with_pos_split_0.csv 2 days ago by UNIBA_NLP2122_Leshi add submission details	0.69469	0.69469
bert_with_pos_split3.csv 2 days ago by UNIBA_NLP2122_Leshi add submission details	0.69300	0.69300
bert_with_pos_split1.csv 2 days ago by UNIBA_NLP2122_Leshi add submission details	0.69237	0.69237
best_cm_1less_conv2d.csv 7 hours ago by UNIBA_NLP2122_Leshi add submission details	0.69224	0.69224

5-th best result (with bert embeddings)

Leaderboard results

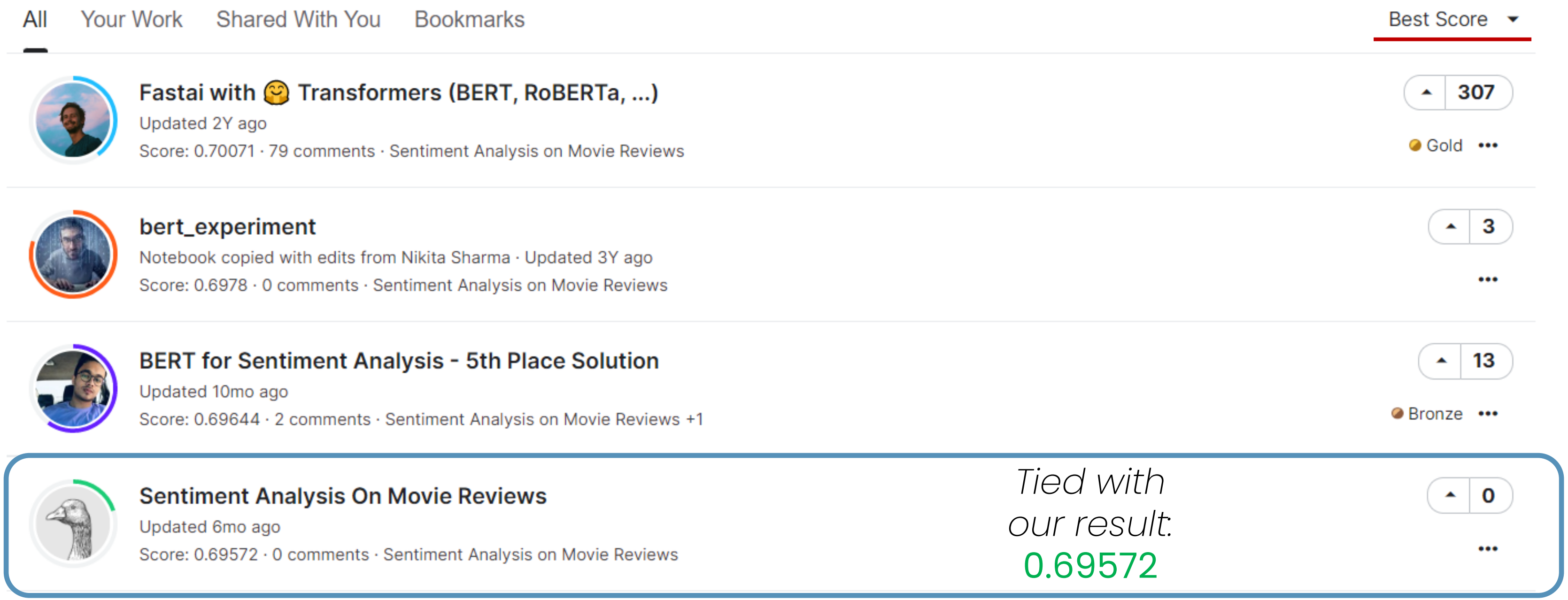
Old leaderboard

#	Team	Members	Score
1	Mark Archer		0.76526
2	Armineh Nourbakhsh		0.76096
3	Merlion		0.70936
4	Puneet Singh		0.70789
5	Yoon		0.68765
6	DrStrangelove		0.67931

Our result:
0.69572

With our best result we would be 5-th over 861 total groups in the old leaderboard

Public lederboard (more recent)



With our best result we would be tied for 4-th place over ?
total groups in the public and more recent leaderboard

Thanks!