Active training with OFROM+

Bachelor project – François Delafontaine

***Abstract*** *– For the purpose of testing active training, we take the OFROM+ dataset, extract a reference subset then train a CRF model iteratively over a separate, initial subset that we increment at each step. We then compare the learning curve obtained for a passive, traditional training with data selected at random and for an active training, with data selected using a strategy based on confidence score and data cost.*

# Introduction

The OFROM+ corpus contains transcribed recordings of spoken French that are annotated in PoS (Part-of-Speech, meaning grammatical labels such as verb, noun, etc.) using an automatic annotation tool, DisMo (Christodoulides et al. 2018). That tool’s accuracy is at 0.97-98, yet the annotation remains too faulty for research. Our core motivation has been to select better data for correction in order to re-train a model.

Active training is an iterative process that seeks to maximize the learning curve, that is, get the highest accuracy with as little data as possible. Our purpose here will be to compare passive (traditional) and active training for the purpose of PoS annotation with OFROM+ data. Active training is expected to be semi-manual but for the purpose of this experiment, the entire process will be automated. Existing labels, generated automatically and therefore faulty, will be used as if manually corrected.

We will present the data and model (1), then explain the process and learning strategy (2) before discussing the results (3).

# 1. Data & Model

Our data, about 2 million datapoints, represents spoken French. We will discuss its specifics (1.1) as well as how that data was pre-processed (1.2) to turn it into an input for our model. We will also briefly present the reference subset (1.3) before covering the model (1.4) used for our experiment.

## 1.1. Data

The OFROM+ corpus is a collection of datasets (from OFROM and CFPR projects) considered here as a whole. Whereas written discourse would use the graphical *word* as its minimal unit (barring *morphemes*), spoken discourse relies on its technical equivalent, the *token* (defined as text split at spaces, dashes, etc.).

Transcriptions are stored in TextGrid annotation files (Boersma & Weenink 2025) that organize tokens in *intervals* (or *segments*), that is, arbitrary sets of tiers within a time span, and *tiers*, that are sets of intervals allowing parallel visualization of text along the audio signal, as showcased in the 5 seconds excerpt of Figure 1, with three tiers for the same speaker: one with the tokens, the second with their corresponding PoS labels and the last with their corresponding lemmas.

Une image contenant ligne, texte, diagramme, Tracé

Description générée automatiquement  
Fig.1: OFROM+ transcription

Among the tokens are reserved symbols: (a) the “\_” pause, (b) the “#” anonymization, (c) the “@” third-party speaker, (d) the “%” inaudible speech and (e) the “-” truncation. Symbols (a-d) appear inside their own *interval* while (e) appears at the end of a token.

PoS labels are structured in fields:

XXX:field2:field3

The first field is a three-letters code indicated the main grammatical category: VER for verb, PRO for pronoun, etc. Second and third fields further detail that category: VER:inf for an infinitive, PRO:per:sjt for a subject personal pronoun, etc. As many fields as needed can be added.

## 1.2. Pre-processing

We first extracted from our set of OFROM+ files the list of all tokens (occurrences) with, for each token, their metadata (file, speaker, start/end timestamps, etc.). That includes for each token its PoS label provided by DisMo. Again, those labels will be used for the purpose of this experiment as if they were manually corrected.

From there, we parsed our list of tokens to group them into *sequences* and *sequences* into files:

* *sequence*: a set of tokens. Linguistically, it corresponds to an IPU (Inter-Pausal Unit) with a pause threshold of 0.5s.
* *file:* a set of sequences. It corresponds to the content of the original TextGrid file. Files are the minimal unit for manual correction and sharing.

The sequence will be used as a minimal unit for our model. The file is there to reproduce as best as possible real conditions for our experiment.

During that parsing, several more changes occur.

First, we removed all reserved symbols presented in (1.1): pauses, truncations, etc. This has two justifications.

For our first reason, we need to introduce the notion of *non-problematic* token: when a token (type) has only one possible label, irrelevant of its position in a sequence, it is considered non-problematic. That notion assumes no false negative, which from experience is false, but we will rely on it for the rest of our experiment. Reserved symbols are non-problematic and don’t need a model to be annotated.

They can however impact the positioning in a sequence (an anonymized interval may be a noun). Our second justification is that the gaps they create are already a phenomenon in spoken discourse that our model will have to contend with regardless. Our hypothesis would be that, in fact, symbols would only add complexity to the model. We won’t expand further on this matter and simply provide, after pre-processing, the available data count in figure (2).

|  |  |  |
| --- | --- | --- |
| Nb. Files | Nb. Sequences | Nb. Tokens |
| 1,475 | 145,514 | 2,032,274 |

Fig.2: Count of files, sequences and tokens.

Beyond symbols, we also added two informations about our files:

1. *value*: how many tokens of interest the file may contain, as a percentage of its total.
2. *cost*: how many tokens are expected to be manually corrected.

Both rely on existing labels from DisMo. Tokens of interest are grammatical tokens: their labels should contain ‘ADV’, ‘CON’, ‘DET’, ‘PRO’ or ‘PRP’. Tokens to be corrected are any problematic ones (with +2 candidate labels). The cost is comparable to an ENUA (Expected Number of User Actions, Arora & Agarwal 2007) with a single manual correction. It assumes non-problematic tokens are not revised/corrected (for false negatives, labels the model would have missed) and considers the need to verify a token as an action, even though few labels will be changed.

Finally, when parsing we already separated the tokens and their PoS labels: files therefore contain two lists of sequences, one for the tokens, the other for the labels. Technically, within a sequence each token is a dictionary with one ‘token’ key containing the token text as value. This is because sequences are there specifically to train a CRF model. That separation and formatting is there purely to run the process faster.

## 1.3. Reference dataset

During training, we are expected to evaluate our model on a *reference dataset*. In theory this should be manually corrected data but, for this experiment, we select ~100,000 tokens at random from our available dataset and declare this our reference.

As for the size, a *file* is usually around ~1-2k tokens; corrections/revisions usually cover ~10k tokens; a training set is expected to have ~100k tokens. That last number is the one we choose and, within our available data, leaves most of it available for selection.

While files and metadata are pre-processed, the reference dataset is technically part of the process, meaning that each time it will be different.

## 1.4. Model

The original DisMo annotation tool is multi-layered: (a) dictionaries, (b) two layers of CRFs and (c) a set of linguistic rules. Our model is, on the opposite, the simplest we could conceive.

We rely on a CRF (Conditional Random Field) model as it is the reference for language annotation. It builds upon a Markov model where values in a sequence are conditioned by what precedes them (contiguously) in that sequence; the CRF simply adds conditions for the value itself.

Technically, we use Python’s scikit-learn library, more precisely the *sklearn\_crfsuite* library. When training that model, sequences are a set of dictionaries, with each dictionary containing the conditions to select the value. For *tokens*, this would typically be the token itself, its 2-3 last letters, etc. In our case, those dictionaries only contain the token itself, meaning it can be considered roughly equivalent to a classic Markov chain.

It would be possible to further simplify the task by only training the model to label the first field of our labels (the main grammatical category). Earlier tests planned for two CRFs, one for the main category and the second, using the former’s output, for the other fields. Those tests showed that a single CRF on the full labels had a high enough accuracy to abandon such distinctions.

Earlier tests have also set the hyperparameters: 0.22 for ‘c1’ and 0.03 for ‘c2’. For lack of understanding, we haven’t revised those hyperparameters since.

# 2. Process

We have so far pre-processed our data and selected a model. What remains is to set up an automated pipeline (2.1) and, within that pipeline, to define a strategy for active learning (2.2).

## 2.1. Pipeline

The pipeline is in 3 steps:

1. Select an initial subset.
2. Train on the subset and evaluate on the reference dataset.
3. Select more files and add them to the subset, then repeat 2.

The initial subset is, within this experiment, always picked at random. Its size corresponds to the size of each addition at step (3). This means that if we start with a 10k subset, we will increment it by batches of 10k.

However, that amount is the number of tokens: what we actually add is files. This means that we keep adding files to the subset until we reach (or exceed) that amount. For small files this is not a problem, as they are likely to be under the threshold; for larger files, with +4k tokens, this may have a more noticeable impact. As a result, steps under 10,000 tokens (occurrences) should be avoided.

len( [token for token in sequence for sequence in file] ) >= target\_number

Once we have our initial subset, we start training.

For training we use scikit-learn’s CRF default method. We then evaluate our trained model on the reference dataset to obtain an accuracy score (not using the class’ built-in F1 score method but scikit-learn’s default function). When predicting, however, we will also be interested in confidence scores: for that purpose we build upon the *predict\_marginals\_single* method. Only the accuracy score is returned when training: this is to plot the learning curve.

The confidence score is used when selecting additional files.

So far the pipeline has been identical for passive (traditional) and active trainings. The divergence occurs at step (3) where passive training selects more files at random, whereas active training requires a more complex strategy.

Before discussing that strategy, we will briefly cover a technicality: due to how lengthy operations can be and the need to for replication (see point 3), each time we complete the pipeline we save the resulting accuracies in a json; by appending repeatedly on it, we obtain the input to plot our learning curves.

## 2.2. Active training

With active training, we want to select files that are most likely to improve our trained model. In theory we would not have labels for data we want to annotate and, therefore, we cannot use the file with the lowest accuracy score. But when predicting labels, we do get a confidence score for each token (occurrence).

What we do then is to average the confidence scores over the entire file and use that to weigh the files: the lowest score is the file we want.

min( file\_confidence\_score )

We however have more information available, namely the file’s value (weight) and cost. We can also modify how much we care about each criteria (confidence score and file value as defined in 1.2); only the cost is kept constant.

weight = ( (modifier1\*(1-file\_confidence\_score)) \* (modifier2\*file\_value) )  
 / file\_cost

For this experiment we actually set modifier1 to 1. and modifier2 to 0., meaning we only cared about the confidence score. This was done mostly to simplify the formula and avoid unexpected effects. In theory, the confidence score will select files with the most problematic cases, making a file’s value redundant; the idea of a separate value stems from the need to ensure a diverse enough dataset, for which more testing would be needed.

As a last remark, since we want to minimize the score but the highest weight, for this formula we take its inverse, (1-file\_confidence\_score). With both the confidence score and the file’s value being between 0 and 1, the numerator will always be lower than 1. However, the file’s cost is an absolute number that can easily go in the hundreds. We cannot turn it into a percentage as the cost of a file is not relative. We don’t expect it to have too much of an impact when using that strategy but needed to not the difference in scale.

Enacting that strategy requires to predict labels over the entire remaining data (dataset – reference dataset – subset). This adds a non-negligible overhead to the process but, again, active learning is expected to be semi-manual, with manual corrections following a selection. Seconds of runtime would be dwarved by hours of work.

# 3. Results

We should have everything at this stage to discuss the results of our experiment: the dataset, reference dataset and subset, as well as the pipeline’s steps and the active learning formula.

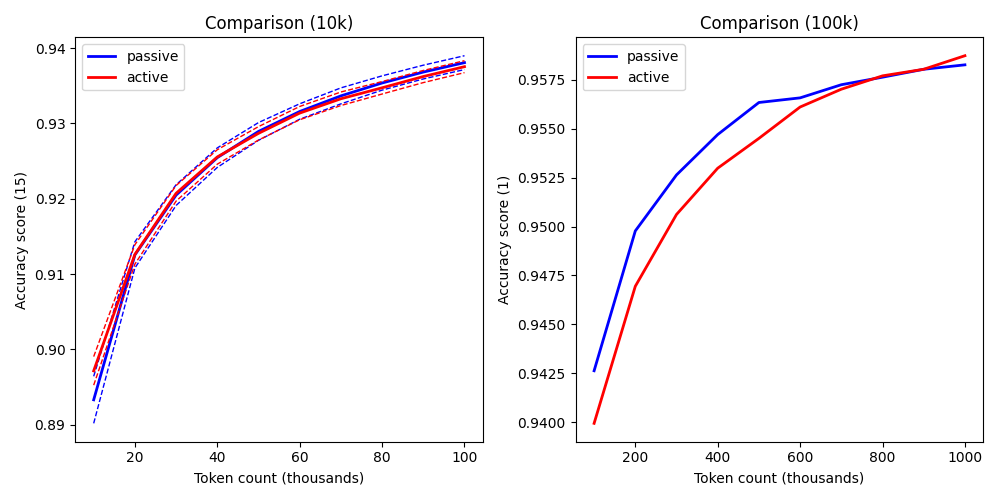
First, we ran the experiment as presented and only varied the size of the subset (and subsequent additions), each time with ten rounds.

Our first observation is how high the variability is. Without several dozen replications, the deviation makes any attempt at comparison impossible.

For comparison purposes, and due to the duration of the process and the need for replication, we limited our observations to two sets:

1. One series going from 1 to 10 thousand tokens in steps of 1,000 tokens.
2. One series going from 10 to 100 thousand tokens in steps of 10,000 tokens.

Figure (4) presents the results for passive and active training. The title contains the step in number of tokens and the y-axis the number of replications for both trainings, each time in parenthesis.

  
Fig.4: learning curve comparisons

Our second observation is about the difference, for passive training, between the end of series (a) at ~0.91 accuracy and the beginning of series (b) at ~0.895. As can be seen for the beginning of passive and active training in both graphics, when selected at random, on average, the result will be comparable. The difference, thus, is likely due to how additional data is selected: not by tokens by files, meaning that at each step, more than 1,000 tokens is likely to be added. This means that the first graph may actually end at 12-15k tokens.

Still, the difference is noticeable enough to question whether comparison is still reliable; it would suggest the need to rework the code so as to control added data globally instead of at each step.

Our third observation is the accuracy score itself, irrelevant of the purpose of our experiment: the simplest CRF model, at 100,000 tokens, provides an accuracy score at ~0.93. We can consider 0.92 as an absolute floor for any annotation tool. In general, it has been a concern of the OFROM+ corpus to properly evaluate annotations: a score > 0.9 is not only expected, following Zipf’s law it is no indicator of the reliability of that annotation. It is good for general cases but says nothing of difficult cases.

This, however, is unrelated to evaluating active training.

# References

Arora, Shilpa & Agarwal, Sachin (2007). Active Learning for Natural Language Processing. *Language Technologies Institute School of Computer Science Carnegie Mellon University 2*, 2007.

Boersma, Paul & Weenink, David (2025). *Praat: doing phonetics by computer [Computer program]*. http://www.praat.org.

Christodoulides, George, Avanzi, Mathieu & Goldman, Jean-Philippe (2014). DisMo: A Morphosyntactic, Disfluency and Multi-Word Unit Annotator. An Evaluation on a Corpus of French Spontaneous and Read Speech. *LREC 2014*. hal-01703495.