Active learning on PoS annotation

Bachelor project – François Delafontaine

Abstract *– To experiment with active training, we use the OFROM+ dataset, extract a reference subset then train a CRF model iteratively over an initial subset that we increment at each step. We then compare the learning curve obtained for a passive, traditional learning with data selected at random and with data selected through strategies centered around the confidence cost. We observe no improvement on a learning curve starting at 10,000 tokens, not without severely limiting the available data.*

Introduction

The OFROM+ corpus (Avanzi et al. 2012) 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 motivations are to better evaluate an annotation tool and to select better data for correction in order to re-train a model.

Active learning 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 learning for the purpose of PoS annotation with OFROM+ data. Active learning is expected to be semi-manual but, for the purpose of this experiment, the entire process will be automated. Existing labels, while faulty and automatically generated, will be used here as if manually corrected[1].

We will present the data and model (1), then present the current experiment with its results (2) before discussing it (3).

1. Data & Model

Our data comes from a database of spoken French counting ~2 million datapoints[2]. We will present that data's specifics (1.1) as well as how we pre-processed it (1.2) to turn it into an input. 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 dataset (from OFROM and CFPR projects) of audio recordings that have been manually transcribed in TextGrid annotation files (Boersma & Weenink 2025). We will, to describe our data, use a series of terms:

[1] The labels have been partially corrected during an earlier phase of this work.

[2] Specifically from the May 2025 version of the OFROM+ corpus.

- *token*: a minimal unit corresponding roughly to a word.
- *label*: the PoS label attached to a token.
- *segment*: or *interval*, a set of tokens bounded by two timestamps.
- *sequence*: a set of tokens corresponding to an IPU.
- *tier*: a set of segments along the audio signal.
- *speaker*: a set of tiers (usually one transcription + annotations).
- *file*: a set of tiers (from all speakers).

Some of those terms describe the TextGrid structure: text is put in *intervals* to align it on the audio using timestamps; the OFROM+ corpus calls them *segments*. *Tiers* are a convenient way to put parallel information along the audio signal: annotations, overlapping speakers, etc. *Tiers* can contain any type of information but, for spoken discourse, we expect a set of speakers and, for each speaker, one reference tier (the transcription) with others attached to it (annotations). A TextGrid file is the set of all tiers related to an audio recording.

Files should be thought as a set of datapoints and the minimal unit for correction and sharing. This is important as, when selecting data, we will want to select files, not *tokens*.

The *token* is a technical unit. Text is split (tokenized) using a set of symbols (space, dash, etc.) as boundaries; the resulting list are tokens. Linguistically, the minimal unit would be either the *phoneme* or *morpheme* but in practice, *tokens* have been the minimal unit that oral linguistics has worked on.

As for the *sequence*, it is specific to this experiment and corresponds to an IPU (Inter-Pausal Unit), that is any set of tokens between two pauses. Types of pauses are defined by their duration: there is a threshold under which a pause isn't an IPU boundary anymore. We won't discuss it further: we chose 0.5 second as our threshold. This means that by a *sequence* we mean any arbitrary set of tokens between two pauses of at least 0.5 second (or at the start/end of a *tier*).

Figure 1 showcases a TextGrid with a 5 seconds audio excerpt and three tiers containing, in order: the tokens, their corresponding PoS labels and their corresponding lemmas.

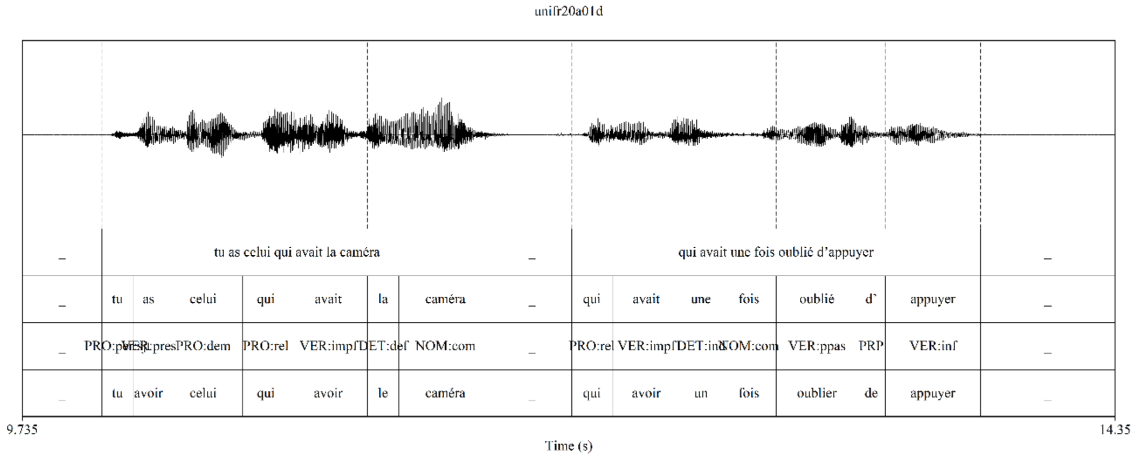
unifr20a01d
_ tu as celui qui avait la caméra _ qui avait une fois oublié d'appuyer _
_ tu as celui qui avait la caméra _ qui avait une fois oublié d' appuyer _
_ PRO:per:suj VER:pres PRO:dem PRO:rel VER:impf DET:def NOM:com _ PRO:rel VER:impf DET:ind NOM:com VER:ppas PRP VER:inf _
_ tu avoir celui qui avoir le caméra _ qui avoir un fois oublier de appuyer _
9.735
14.35
Time (s)

Fig.1: OFROM+ transcription

Not all tokens are text. The OFROM+ corpus reserves some symbols for special cases: (a) the "_" pause, (b) the "#" anonymization, (c) the "@" third-party speaker, (d) the "%" inaudible speech and (e) the "-" truncation. What matters here is not hat those symbols are or how they are used but that they exist; they do not require further discussion.

Our last remark related to the data concerns PoS labels. We won't provide an exhaustive list (there are +60 of them) but discuss their structure. Those labels have fields separated by ":" symbols:

XXX:field:field

The first field is a three-letters code for the main grammatical category: VER for verb, PRO for pronoun, etc. From there, as many fields as needed can be appended, although a single one is the norm and it rarely exceeds two: VER:inf for an infinitive, PRO:per:sjt for a personal pronoun in a subject role, etc. This means that it would be possible to narrow the annotation task to two or even a single field: keeping it to the main grammatical category would reduce the number of labels to 12.

1.2. Pre-processing

With our data defined, most of the work went into processing it for use in our experiment. We already knew what model we intended to use (1.4) and made choices in advance.

An early step consisted of extracting from the dataset a table of all tokens (occurrences) with, for each token, their metadata (file, speaker, start/end timestamps, etc.) including their PoS label. We then grouped those *tokens* again, into *sequences*, then *sequences* into *files*. When doing so we modified the data:

a) We removed all reserved symbols (see 1.1).
b) We added utility and cost values for our files.
c) We structured our tokens and labels into input for our model.

To discuss reserved symbols, we need to introduce the notion of *non-problematic* tokens. When all occurrences of a token have been labelled, as has been done with DisMo, we can list all possible labels that a token can take, regardless of its position in a sequence. If that number of labels is 1, the token is considered *non-problematic*. That notion ignores false negatives (missing labels from lack of data or faulty annotation) but, for the purpose of this experiment, we assume that our labels are correct.

Some exploratory work let us visualize *problematic* tokens by using DisMo's confidence scores to parse them (with a 0.9 cutoff point) and mapping their relations at the main grammatical category level. Figure 2 shows all relations between those categories and suggests that the number of *problematic* tokens (as a type, not an occurrence), is rather limited, especially for non-lexical tokens: about ~40 grammatical tokens (in ADV, CON, DET, PRO and PRP) in DisMo are *problematic* with a low confidence score.

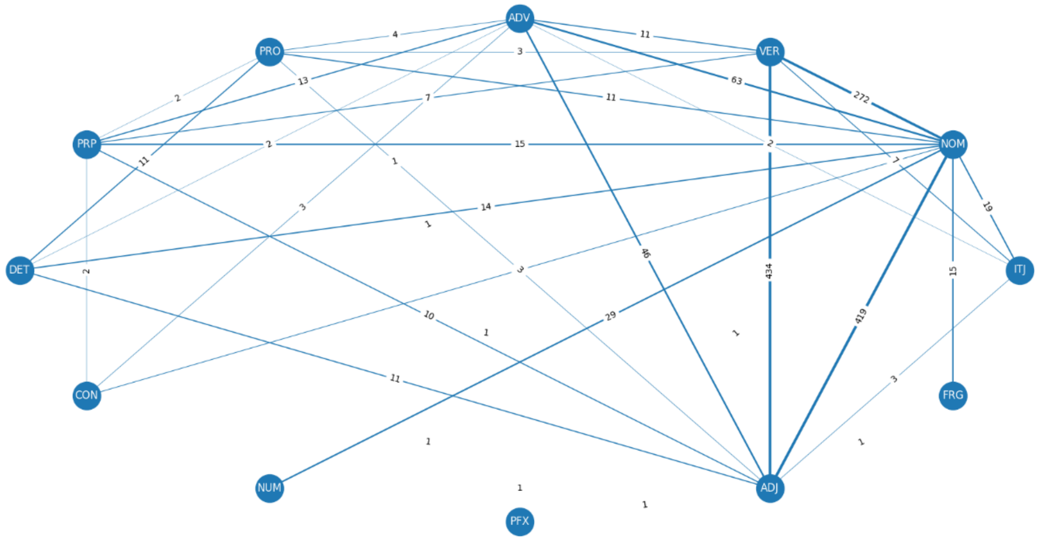

Fig.2: Ambiguities between main categories

Since they are non-problematic, we discard reserved symbols, truncations included. This does leave gaps in sequences (for example an anonymized noun) but such gaps already exist as a phenomenon in spoken discourse and our model will have to contend with it regardless. We hypothesize that removing those symbols actually helps the model, although that has not been tested. The remaining count of datapoints after removal is presented in figure 3.

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

Fig.3: Count of files, sequences and tokens.

We also used the notion of *non-problematic* tokens to give our files a *utility/cost*:

- *utility*: how many tokens of interest the file may contain.
- *cost*: how many tokens are expected to be manually corrected.

The *utility* is born from a concern for a reference dataset as diverse as possible. In practice, however, it is simply the sum of all occurrences of grammatical tokens (ADV, CON, DET, PRO and PRP) and therefore of limited relevance.

The *cost* is any *problematic* token (with +2 possible labels). This is comparable to an ENUA (Expected Number of User Actions, Arora & Agarwal 2007) with a single manual correction. It assumes non-problematic tokens won't get any revision and assumes an equal cost for all problematic ones, without distinction for the different types of actions: observing the labels, picking a correct one for faulty labels, writing the new label. While this is a basic approximation, our cost is still fairly close to a practical estimation of the workload for corrections.

The cost has been normalized to keep its value between [0, 1].

Both *utility* and *cost* rely on a pre-existing annotation. In our case, it is the DisMo annotation. Such information may not be available from the data itself, in which case we would expect it to be readily available from open-source dictionaries. How to account for that pre-existing information when training a model is beyond the scope of this experiment.

Our last modification is purely technical. We know we will feed a CRF model (see 1.4) and therefore already separate the tokens and labels, and turn the tokens from strings to dictionaries (with a 'token' key and the token as value). This is purely to avoid any formatting when training the model.

1.3. Reference dataset

Our initial approach was to split what data we had to train the model into a training set and a testing set, and to user the latter for our accuracy score. We switched since to using a single separate, reference subset on which any model is evaluated.

In theory that reference dataset would be fixed – our ambition is, again, to obtain such a set – but, to try and avoid any bias, especially in case of that dataset not containing certain problematic tokens (type), we instead chose to select it at random. This means that, when running our pipeline (point 3), the reference dataset will change.

A *file* is expected to contain ~1-2k tokens; corrections/revisions usually cover ~10k tokens; for training sets, the usual number is ~100k tokens. We chose that last number for our reference subset that is, again, taken at random from the dataset just before the pipeline is run; the selected data is naturally unavailable for the pipeline itself.

1.4. Model

Once we have pre-processed data and a reference dataset, all that we lack is a model. The original DisMo annotation tool is multi-layered: (a) dictionaries, (b) two layers of CRFs and (c) a set of linguistic rules. While we originally wanted to emulate that structure, we ended up doing the opposite.

CRFs (Conditional Random Field) are the model of reference for PoS annotation. They allow for the selection of a variable's value depending on both the variable's factors and the variable's position in a sequence. The theory behind them, as well as the different types of existing CRFs, is fairly complex. Starting from a naive Bayes classifier, it adds a sequential and a conditional aspect:

- sequential by implementing a Hidden Markov Model
- conditional by moving to a logistic regression

The former, relying on the Markov property of a variable, in a given sequence, being only related to the previous, contiguous variable in that sequence, handles positioning; the latter, by replacing a single weight with a set of them. The combination of a logistic regression and a hidden markov model is a linear-chain CRF. The formula would be (Sutton & McCallum 2010: 23):

$$p(y|x) = \left(\frac{1}{Z(x)}\right) * \prod_{t} \exp\left(\sum \theta f(y_t, y_{t-1}, x_t)\right)$$

That is, leaving the normalization aside, a label's probability is the product of probabilities for each label of the sequence, with each being a sum of features (with θ their parameters) and each feature being a function based on the considered label, the label in the previous position and some condition.s. The sum is a regression and the parameters of the feature function contain the sequential nature through 'y_{t-1}' and the conditional nature through 'x_t'.

We use scikit-learn's *sklearn_crfsuite* implementation and do not now how it is implemented internally. The input for each sequence is two arrays 'x' and 'y' with 'x' being an array of dictionaries containing the features.

We chose to build a single-layered model, meaning a single CRF. As for the features, we only consider the *token* itself. This approach will fail if the model is tasked with predicting the behavior of new tokens: it is expected to add features such as the 2-3 last letters, or if it is capitalized, etc. We however chose to pick the simplest possible CRF model. This choice was made during the exploratory phase, when we observed accuracy scores above 0.9 (further discussed in point 4). We sought simplicity for control but also as a floor on what to expect for a PoS annotation task.

The only way to further simplify the task would be to reduce the number of labels. As said in point (1.1), we could cut their numbers by only training our model to label the main grammatical category. However, given the high accuracy score obtained on the full set, such a step was not justified.

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. Experiment

The general process of our experiment 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, taken from the OFROM+ dataset, is always selected at random, as is the reference dataset (1.3). We determine a batch size, in number of tokens, that corresponds to the size of the initial subset and the data size added at step 3: if that size is 1k (one thousand), the subset will grow by 1k each time we loop to step 3. By default, we ran 10 loops. As pseudo-code:

```
for i in range(nb_loops):
    X, y = select(…, nb_tokens)
    crf_model = CRF().fit(X, y)
    X_ev, y_ev = reference_dataset
```

```
        accuracy, avg_confidence = predict(crf_model,
                                           X_ev, y_ev)
        yield accuracy
```

Where *nb_tokens* is the batch size.

The core of the experiment happens during file selection. We will present the default passive strategy (2.1), then cover active learning with a file strategy (2.2) and then a token strategy (2.3).

2.1. Passive learning

During selection, the most basic approach is to select files at random. This is the passive strategy, emulating a traditional training while building a learning curve through the experiment.

Yet even this is not trivial. We said that the batch size is is number of tokens, but what we actually select is sets of tokens (the files). Files are expected to be in the 1-2k size range, and can go as low as 200 tokens or as high as 4,000. Again, to emulate real conditions we cannot feed our model with part of a file: therefore any batch size under 5k has proven unreliable: at each step the compared learning curve have an ever-growing difference in size.

We therefore ran two batch sizes: one of 10k tokens and one of 100k. Figure 4 shows the learning curves in two graphs. The title contains the batch size in parenthesis and the y-axis the number of replications in parenthesis. Confidence intervals (in dashed lines) will always be with a 95% confidence.

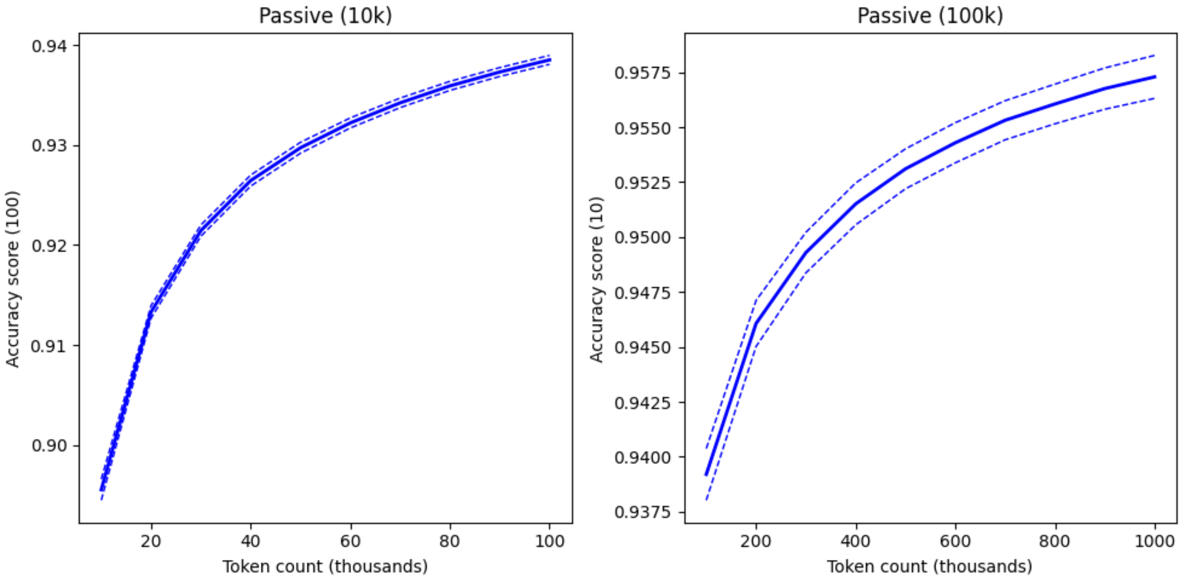

Fig.4: Passive learning

At 1k tokens, the average accuracy score is ~0.83; at 10k it reaches ~0.89; at 100k, the usual training size, it should be ~0.94. Pushing it to 1mn (one million) tokens only gets us to ~0.95. This is, again, with as simple a CRF model as can be conceived and the full tagset. It suggests an accuracy floor for PoS models at 0.93 – with the caveat that, again, the labels we use were automatically generated.

From there, strategies for file selection become *active*. We will present them in order of complexity.

2.2. File strategy

The simplest active strategy is to select files by confidence scores. Each time we have trained and evaluated a model, we use that model to predict the labels of the entire remaining dataset (dataset – reference dataset – current subset). During that prediction, we generate a confidence score per token (occurrence). We then average the confidence scores per file and use that average as weight.

file_weight = (1 – avg_confidence_scores)

Since we want to select the file with the lowest average (having therefore the most problematic tokens (occurrences)), but maximize the weight, we invert the value.

However, since most confidence scores are expected to be >0.9, the average has a small deviation. To amplify it, before the average, we only retain the scores under a threshold: that cutoff point is the accuracy score obtained on the reference dataset. We could assume a monotonic relation or simply use it as it is close to previously observed values, ~0.9. In pseudo-code:

```
confidence_scores = predict(X, y)
confidence_scores = confidence_scores < accuracy
avg_confidence_score = mean(confidence_scores)
```

The result of this strategy is shown in figure 5 (replications in the y-axis are for active learning) : this strategy is essentially indistinct from a random one.

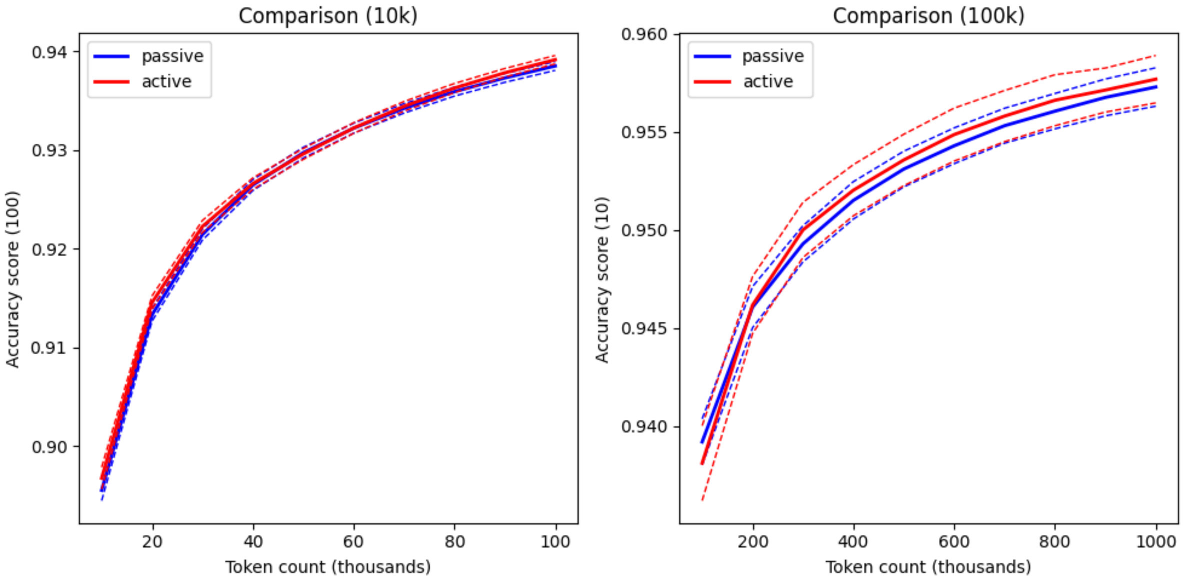

Fig.5: Active learning with file confidence

That the two learning curves don't start exactly at the same point is normal: their respective subsets and reference datasets are random. They eventually meet with enough replications, as shown in the 10k batch experiment. The average could give the impression of a better performance in the 100k batch but this is only due, again, to randomness, as shown by confidence intervals (dash lines).

Having no improvement on the learning curve is not the expected result. Other strategies, however, have fared even worse.

We considered the general strategy:

file_weight = ((m1*confidence_score) + (m2*file_utility)) / (m3*file_cost)

Where *confidence_score* is the inverted file confidence score, file_utility/cost the values presented in (1.2) and m1-3 coefficients for each value (m3 is handled differently due to its position in the denominator). We kept m2 at 0 through all strategies as the utility is redundant with the confidence_score, makes interpreting the results harder and fails at its purpose: to ensure that the growing subset is as varied as possible. We likewise kept m1 at 1 and so the real change was adding cost to our strategy.

file_weight = (1 – avg_confidence_score) / file_cost

We tried that strategy, still with a cutoff point before th average. The result, shown in figure 6, shows no improvement. Rather, the way the cost is computed actually worsens the selection.

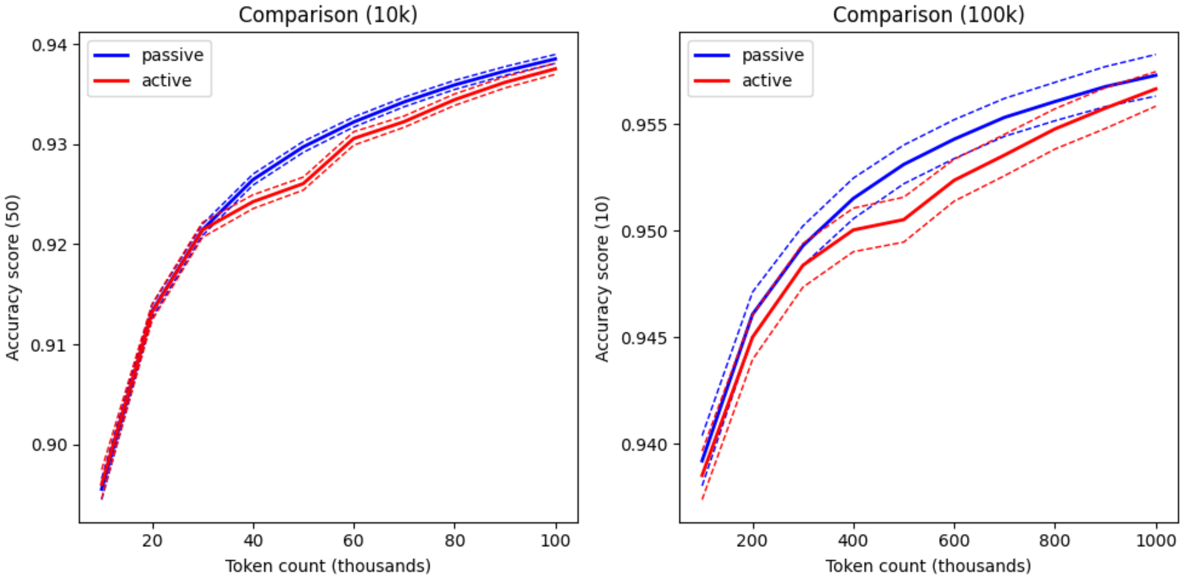

Fig.6: Active learning with file cost

We could invert the cost to see if it produced a better result but this would go counter to intuition: we want the lowest possible correction cost (hence the division), when inverting it would be seeking the most expensive files for correction. Yet this paradox may explain why our way of calculating cost is counter-productive: those files are expensive because they likely have more *problematic tokens*. By taking correction cost into account we may actually get less productive files.

In short, *file cost* as defined here (1.2) must be abandoned.

One last strategy we attempted in this approach was to revise the average confidence score again, not only with a cutoff point but also by selecting the type of token for which the confidence score was taken into account: grammatical *tokens* specifically, based on their main category. That main category is drawn, again, from a pre-existing dictionary of tokens and their possible labels, irrelevant of position. This fully integrates the *file utility* into the confidence score but still worsens the result, as shown in figure 7.

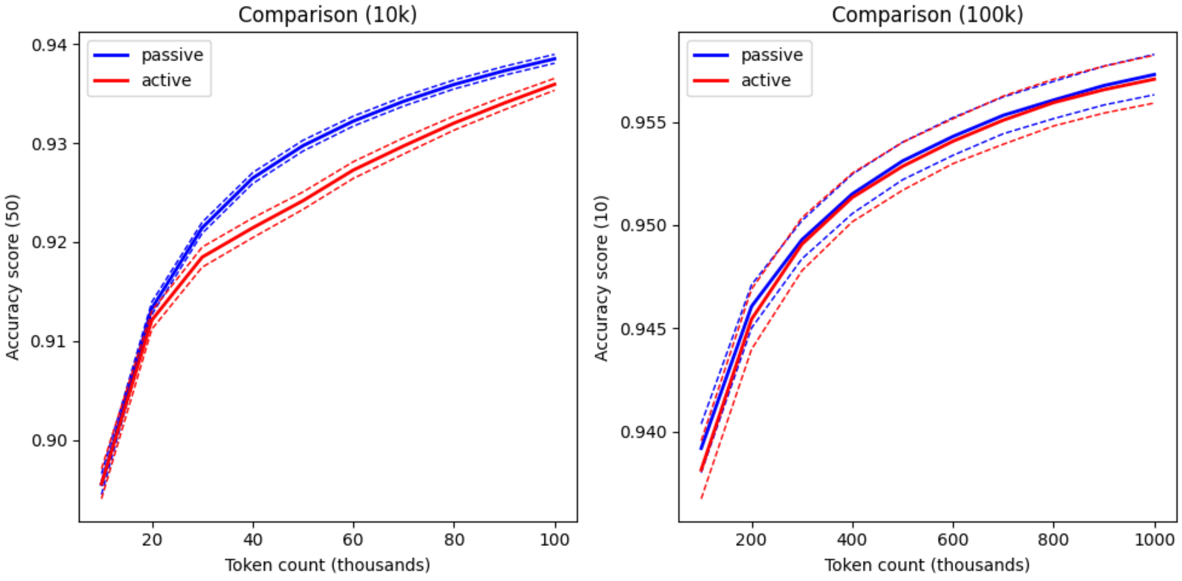

Fig.7: Active learning with file PoS

We don't seem able to improve the active strategy through that approach: our best result is equivalent to a random selection. It may be that the right algorithm escapes us. However, we haven't exhausted all options.

2.3. Token strategy

A different approach is, instead of using the general confidence score, to focus on a small selection of tokens. This requires:

a) Selecting a given number of tokens from either the reference dataset or current subset, using their confidence score.
b) Selecting the next file.s by counting the number of occurrences of those tokens, by their average confidence score.

For the token selection, we will use this formula:

token_weight = log10(nb_tokens)*(1 - token_confidence_score)

When using it on the reference dataset, since its size never changes, the number of tokens will remain constant and can be ignored; when run on a growing subset, it helps remove tokens (types) that don't grow while the logarithm prevents very frequent tokens from dominating. The cutoff point still applies: the accuracy used for it is still from the reference dataset even when the tokens are selected from the subset instead.

For the file selection, the same formula as in (2.2) is used, but the average confidence score is the number of occurrences of our selected tokens weighed by their average confidence score:

avg_confidence_score = mean(sum(token_confidence_score * nb_occurrences))

For each token we count their number of occurrences in the file, multiply that by the token's average confidence score and add that to the sum. There is no inversion since the confidence score has already been inverted at the token level. The result, shown in figure 8, is essentially the same as our initial file confidence strategy.

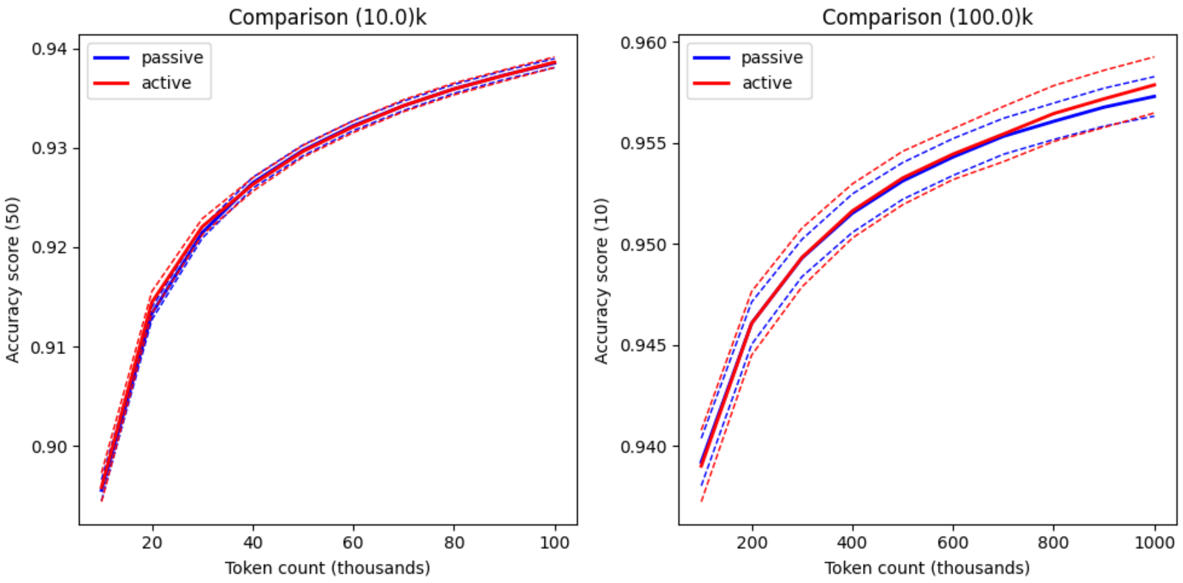

Fig.8: Active learning with tokens

We opted to select tokens from the reference dataset, to select 10 tokens and, again, to ignore the file's cost.

We could vary the formula to either select our set of tokens or apply their confidence score to the *file weight* but, given the lack of improvement, there is no reason to believe that this approach would fare any better than one at the file level (2.2). This only adds complexity to the process for no gain.

3. Discussion

The results we obtain are counter-intuitive. We would expect the active learning strategy to generate a steeper learning curve, getting to a higher accuracy earlier on before flattening. Even Chaudhery et al. (2021), when reporting confusion for PoS model training, still obtain such an improvement.

Instead, none of our strategies fared better than a selection at random:

- Confidence level of the entire file

- Cutoff point and PoS selection
- File cost
- Confidence level of a select set of tokens

One hypothesis would be that we use too much data. The authors we cited (*idem*: 7) use a scale between 0-1k tokens reaching ~0.85 accuracy at 1k tokens, at which point active and passive learning meet again. This would suggest that past 1k tokens, active learning would not perform better.

This is easily tested. To do so, we rework our parsed data to have 1 sequence-per-file, then only retain the first 50k sequences (for faster processing). With the problem of file selection gone, we can now run our experiment on low amounts of data. Figure 9 does exactly that, with batches of 10, 100 and 1,000 tokens; the active strategy is the file confidence (2.1), with an accuracy cutoff but no cost and no grammatical selection.

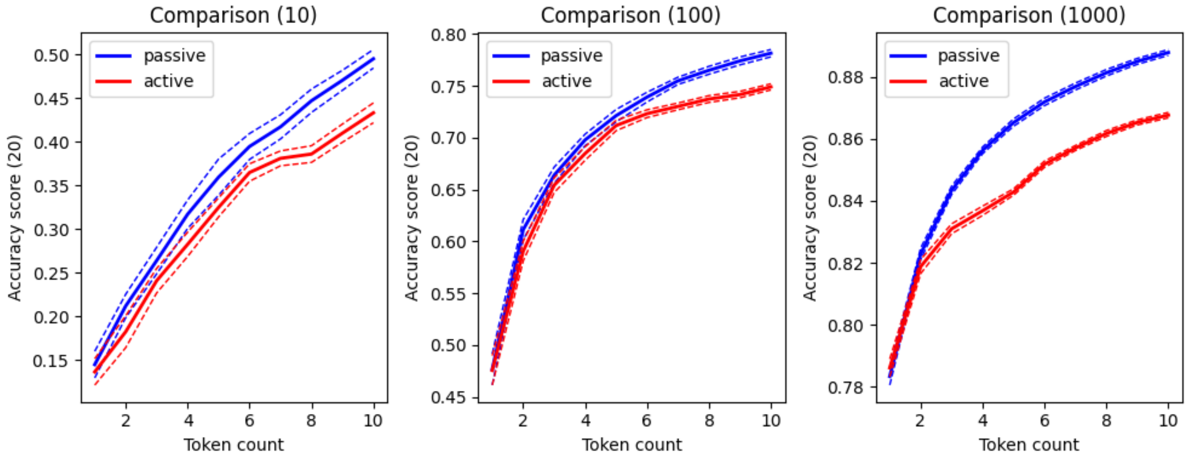

Fig.9: Low-size file confidence selection

If not the data size, then we could suspect that our model is too simple, incapable of genralization. So we added features:

1. Two for the last 3 and 4 token letters
2. One for the PoS tag if *non-problematic*
3. Six for the cotextual tokens among the 10 most frequent

We won't discuss those features (cluttering, the notion of *cotext*, their selection...): the point was to improve the model and see if it would effect active learning. It did not.

Figure 10 shows a better accuracy score overall for a traditional training (with a likely ceiling at ~0.965 achieved before 100k tokens for our selected features). Active learning also improves but shows the same systemic break off as before. We would generalize from there that the model is not at fault – and that our simple version is valid for the purpose of this experiment.

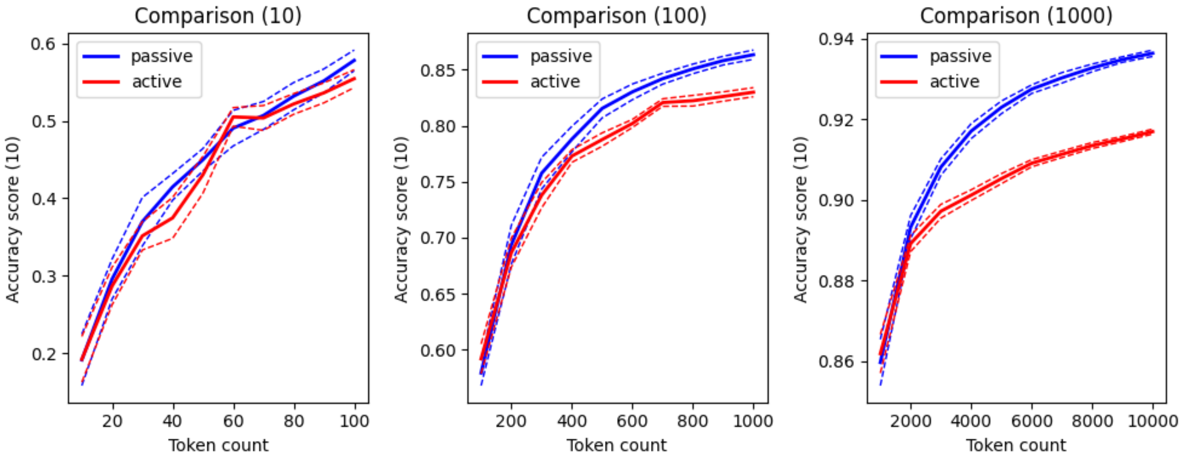

Fig.10: Low-size, added features file confidence selection

We also tested an oracle strategy using the following formula:

file_weight = (1 – file_accuracy)

With an accuracy score from PoS labels that we should not have access to. Even then, that strategy underperformed. We also attempted to move from a simple average to a macro average and then to calculating entropy, to no avail.

Finally, we removed most of the randomness from our experiment:

a. We implemented a “always take the last file” strategy.
b. We used it to select a fixed reference dataset.
c. We removed 1.8 million tokens worth of files.

This means we were left with a fixed 100k reference dataset and a remaining 100k dataset to select from. We compared both the oracle and basic confidence strategies, both without additional features, in that context. The result, in figure 11, shows both strategies marginally beating the baseline; confidence intervals are present for all.

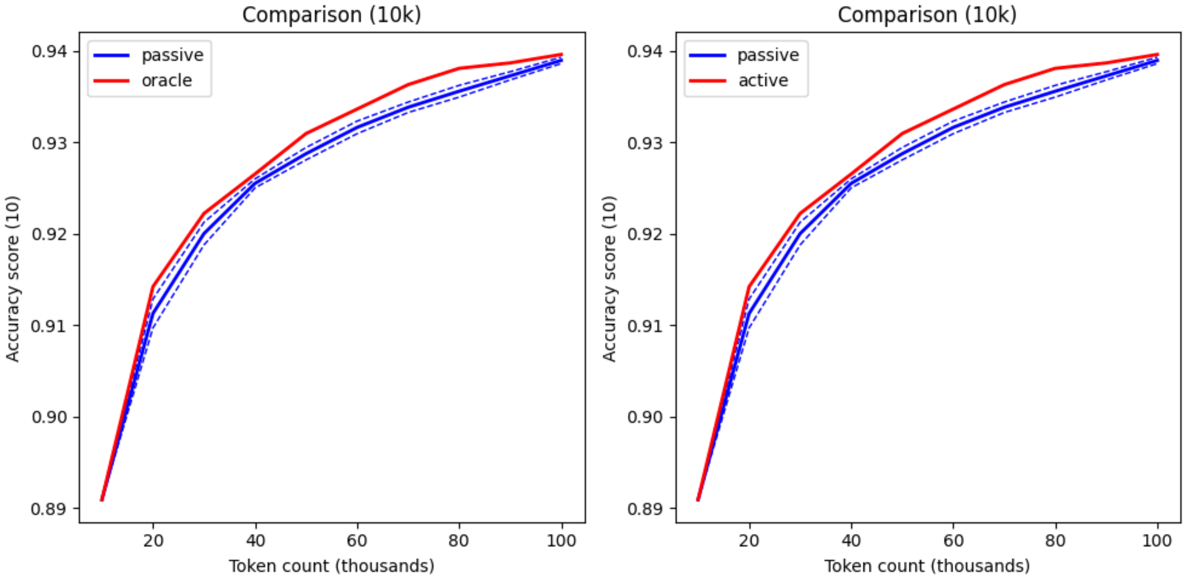

Fig.11: Fixed reference, limited data

It is possible, with this dataset and the strategies we applied, to beat the baseline. What this requires, though, is to severely limit the amount of data for selection. We still can’t determine the reason and could only speculate (diversity, noise, model bias…).

Conclusion

What looked like a simple exercise on a well-known task (PoS annotation), model (CRF) and method (active learning) with an equally established database (OFROM+) ended up being a confusing pursuit to explain and avoid counter-intuitive results.

Almost all of it is due to the constraints we put on that exercise: namely to use such a large data pool, which ended up being the core cause of our results; and to insist on selecting data at the file level, which while seemingly not being the cause of our strategies’ shortcomings did make us work on large steps and already advanced states of training. One crude way to put it would be that, at this scale, a random selection actually becomes the best strategy.

Eventually, we were able to beat the baseline (of random selection); that however is meaningless as it requires going against the expect application of our work.

There would still be a lot more to attempt:

- Observing the distribution of confidence scores,
- Checking the relation between confidence scores and accuracy,
- Applying another cutoff point for the lowest confidence scores,
- Experimenting with strategies based on diversity…

We could even, for the purpose of selecting a reference dataset, use an oracle strategy by testing a model file-by-file on each addition; this would be the most resource-intensive process and also the most optimal for the selected metric.

Which leaves the biggest problem for us: when 0.98 is considered unsatisfying still, is the accuracy score really the metric we want to use?

References

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

Avanzi Mathieu, Béguelin Marie-José, Corminboeuf Gilles, †Diémoz Federica & Johnsen Laure Anne (2012-2025). *Corpus OFROM – Corpus oral de français de Suisse romande*. Université de Neuchâtel, <ofrom.unine.ch>.

Boersma Paul & Weenink David (2025). *Praat: doing phonetics by computer [Computer program]*. <praat.org>.

Chaudhary Aditi, Anastasopoulos Antonios, Sheikh Zaid, & Neubig Graham. (2021). Reducing confusion in active learning for part-of-speech tagging. T*ransactions of the Association for Computational Linguistics* 9, 1–16. DOI: 10.1162/tacl_a_00350.

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.

Sutton Charles & McCallum Andrew (2010). An Introduction to Conditional Random Fields. *arXiv preprint*, 1011.4088.