Department of Informatics, Systems and Communication

An HMM-based Approach for Misspelling Correction

Probabilistic Models for Decisions

Giorgia Adorni 806787

Elia Cereda 807539

Nassim Habbash 808292

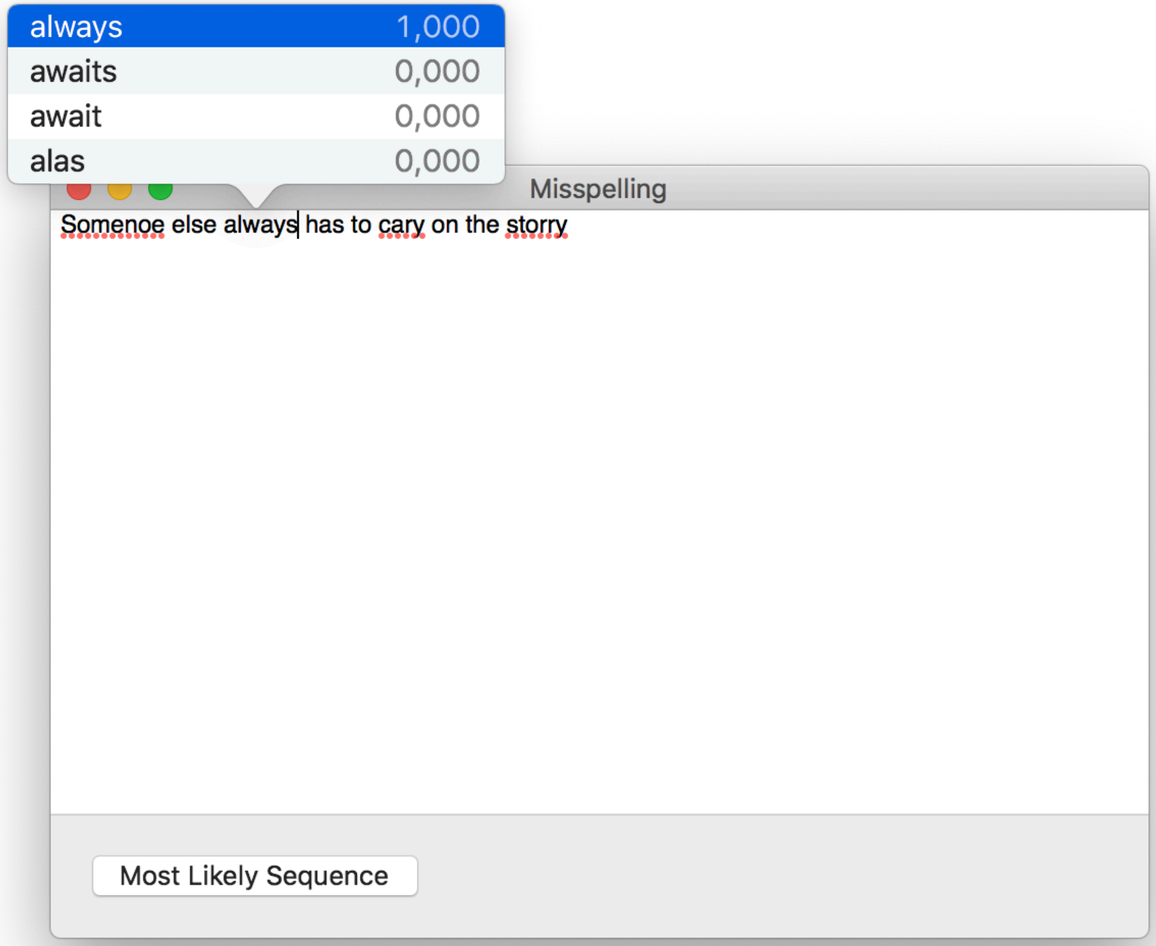
Academic Year 2018-2019

Introduction

Problem Formulation

Given a sentence, we are interested in **detecting** and **correcting** its *misspelling errors*.

This project aims to implement an **Error Model**, capable of offering corrected candidates to a single misspelled word, and a **Hidden Markov Model**, capable of finding the most likely sequence of candidates for each word in a sentence.



Design Choices

Language of choice: English

- Most of the literature approaches this problem using English
- Doesn't use special characters, such as accents
- Doesn't have as many verb tenses and special forms as Italian

Main assumptions

- We don't consider all punctuation and special character symbols, but only letters and sometimes numbers
- A word in a sentence only depends on the previous one (first order HMM)
- The number of errors in a word follows a Binomial distribution depending on the length of the word

Software

- Project developed using the **Python** programming language, taking advantage of various open-source libraries:
  - `edlib` to compute the sequence of edit operations to transform a word in another one
  - `networkx` to represent the trellis graph produced by *Viterbi's* algorithm
  - `pandas` to store the evaluation results
  - `matplotlib` and `graphviz` to visualize the trellis graph
- Graphical user interface implemented as a native macOS application written in **Swift** that interacts with our **Python** code.
  It provides real-time local spell checking and on-demand visualization of the trellis graph.

Datasets

Datasets

Sentences Dataset

Used to construct the **transition probability** matrix

- `big_clean`: concatenation of public domain book excerpts from Project Gutenberg.
- `lotr_clean`: collection of the "Lord Of The Rings" books.

Each corpus has been cleaned of special characters and punctuation, then divided in lower-case sentences.

Language Dataset

Represents the **frequency** of words in a certain language

- `frequency-alpha-gcide`: a lists of the 65537 most frequent words from Wiktionary and the British National Corpus
- `lotr_language_model`: based on the frequency of the words in the **lotr_clean** dataset

Datasets

Typos Dataset

- Collection of 79677 **typos** collected from:
  - Birkbeck native-speaker spelling error corpus
  - Corpus assembled by Atkinson for testing the GNU Aspell spellchecker
  - Misspellings made by Wikipedia editors
  - Errors scraped from Urban Dictionary
  - A Twitter spelling error corpus

- We created another dataset of 62759 typos starting from `lotr_clean`: for each word of the corpus we generated five typos according to the **Perturbation Algorithm**.

Each dataset was then divided into two corpora: 80% is used as a train set and 20% is used as a test set.

Perturbed Dataset

We generated three different texts from each dataset, with varying the percentage of errors introduced.

Perturbation Algorithm

For each line of the input texts we generates a new perturbed string, according to the following steps:

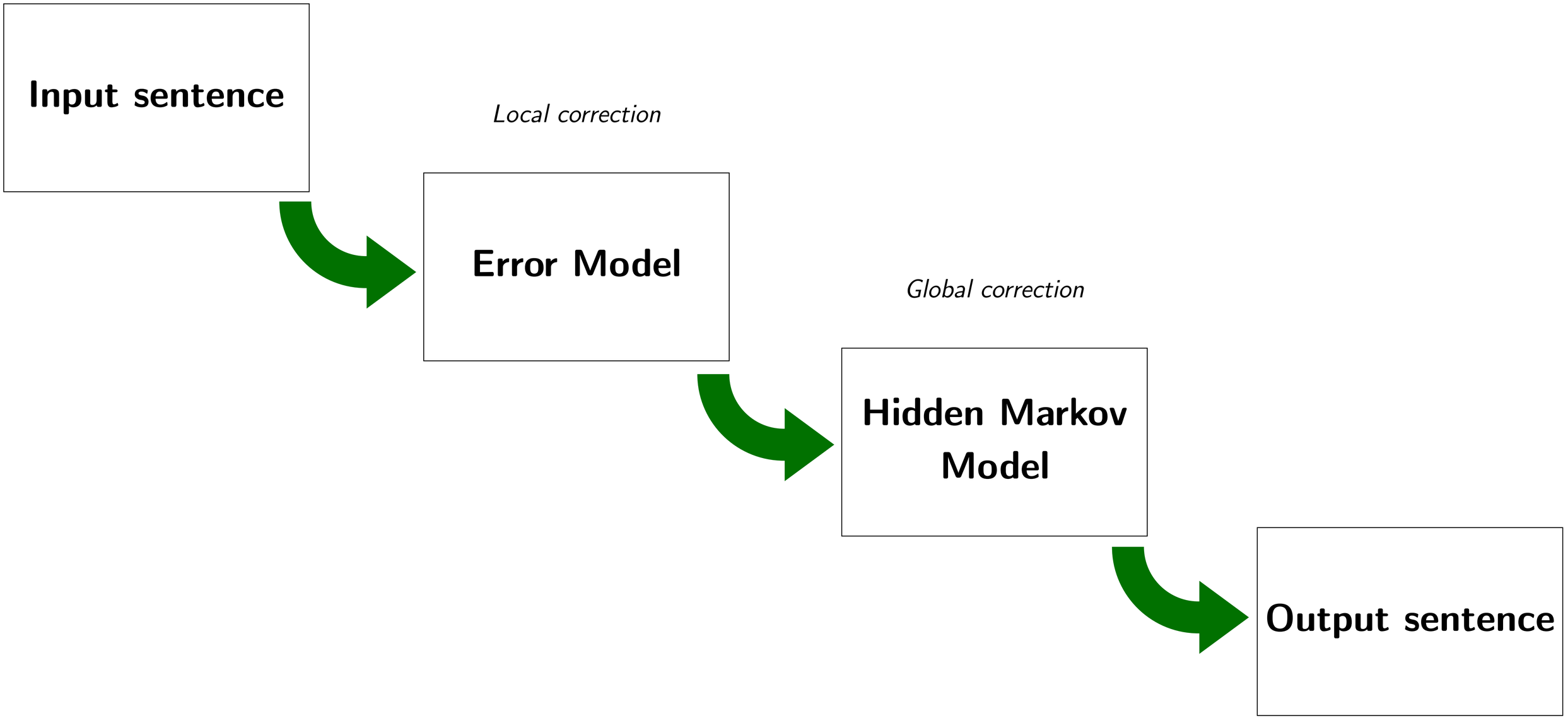
1. Compute the **probability that a word has an edit** by multiplying the value of p (the probability that a certain letter has an edit), contained in the error model, by 5-10-15% that ideally represents the percentage of errors desired.
2. Compute the **number of edits to be introduced *x***, for each word of length n, according to the probability distribution $x \sim \text{Bin}(n, p)$.
3. Choose randomly the x characters to be changed inside each word.
4. Choose randomly the type of edit and apply it accordingly the probabilities contained in the error model (deletion, insertion, substitution, transposition).

Swap errors are only introduced if there are no further changes in the word.

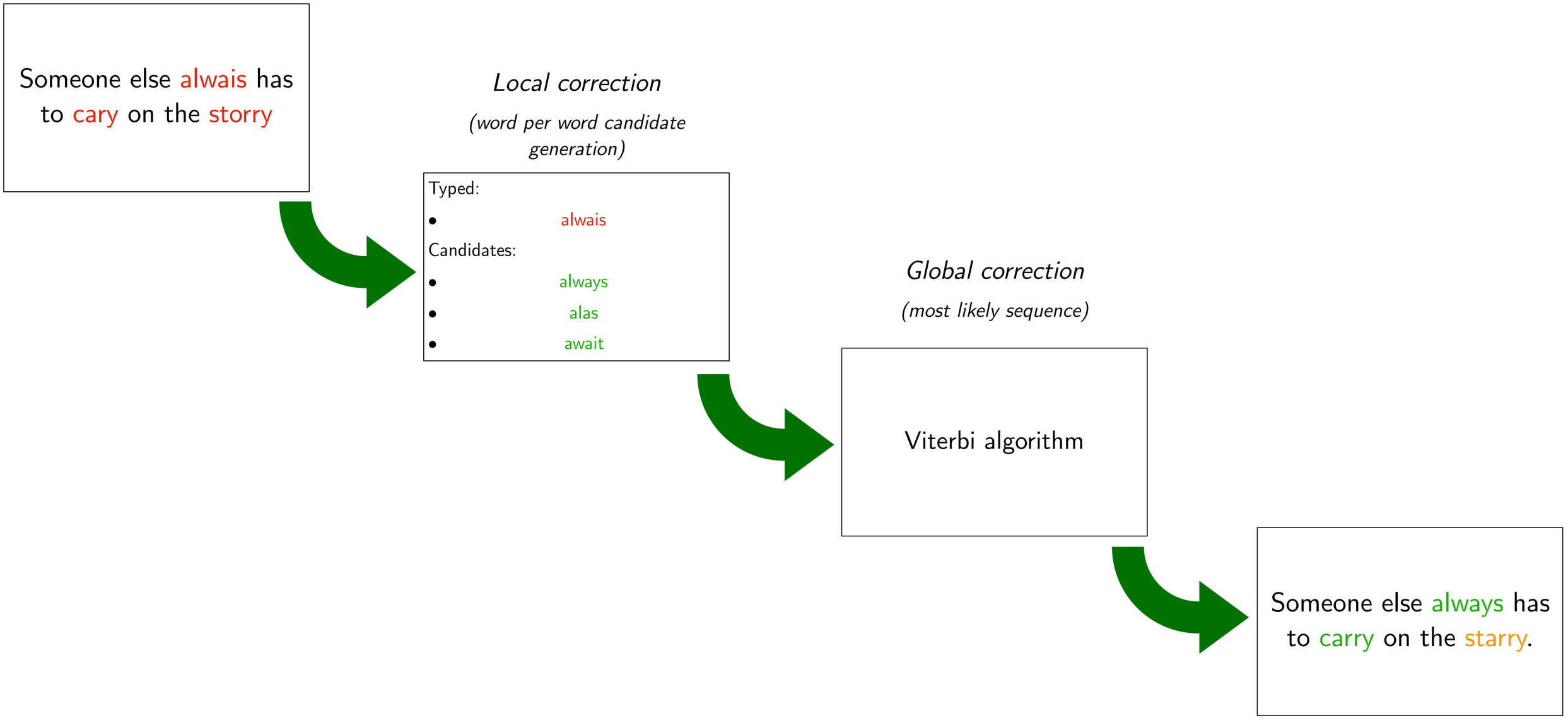
Cases of elimination of a whole word are excluded.

Models

Pipeline


Input sentence
Local correction
Error Model
Global correction
Hidden Markov Model
Output sentence

Pipeline


Someone else alwais has to cary on the storry
Local correction
(word per word candidate generation)
Typed:
alwais
Candidates:
always
alas
await
Global correction
(most likely sequence)
Viterbi algorithm
Someone else always has to carry on the starry.

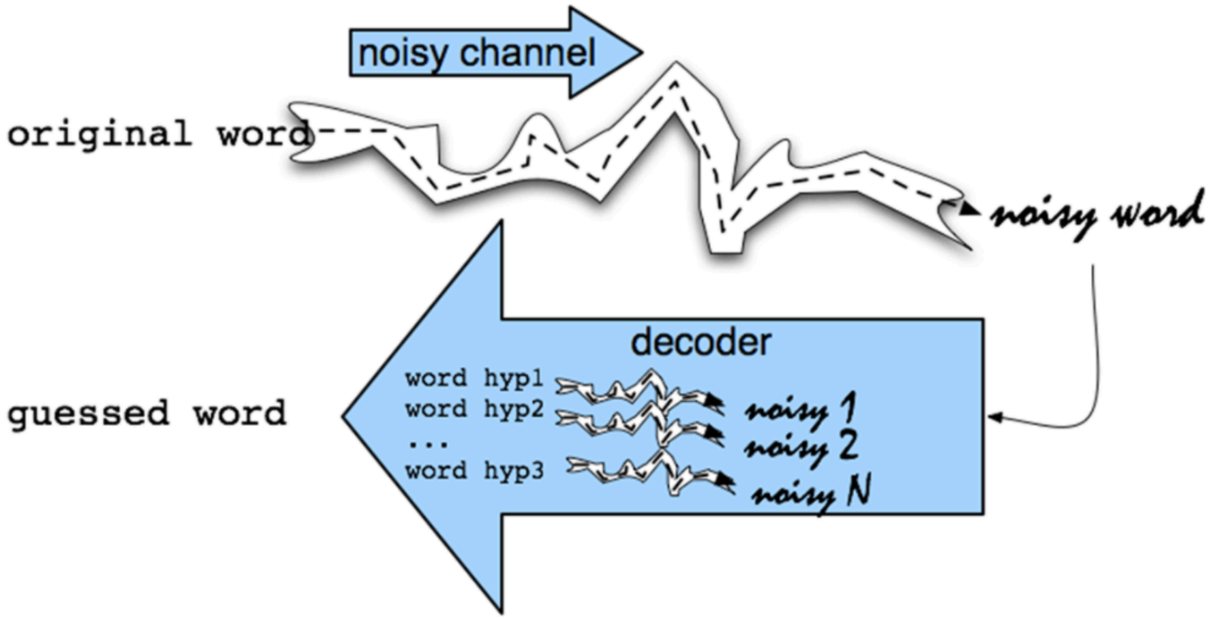
Error Model: Noisy Channel Model

From a **noisy** word, generates a set of **candidate** corrections.

Trained on the **typo** dataset.

Corrects two types of errors:

- **Non-word** spelling errors (graffe/giraffe)
- **Real-word** spelling errors (worm/word)



Error Model: Noisy Channel Model

$$\hat{w} = \arg\max_{w \in V} P(w|x)$$

$$\hat{w} = \arg\max_{w \in V} P(x|w)P(w)$$

Prior or **Language Model**

Likelihood or **Channel Model**

The Noisy Channel Model is a kind of **Bayesian Inference**.

Given a vocabulary **V**, a typed word **x** (e.g. graffe) and an intended word **w** (e.g. giraffe), we're interested in finding the word **w** such that **P(w|x)** is maximized.

We apply this approach to a list of candidate words **w** generated from **x** consisting in all **known** possible edits of x at a given **edit distance**, limiting the set of computed candidates.

Error Model: Noisy Channel Model

The **prior** is estimated thanks to the **Language Dataset**. **P(w)** is the **frequency** of w in the training corpus.

The **likelihood** is estimated by training on the **Typo Dataset**, by assuming that the likelihood of a **typed word** given an **intended word** is **equal** to the likelihood of its **mistyped characters** given the **correct intended characters.**

e.g. **P(oat|bat) = P(o|b)P(a|a)P(t|t)**

To make the model more efficient, we'll have **four confusion matrices**, for four types of errors:

- **Deletions**: $\text{del}[x, y] = \frac{\text{count (xy typed as x)}}{\text{count (xy)}}$
- **Insertions**: $\text{ins}[x, y] = \frac{\text{count (x typed as xy)}}{\text{count (x)}}$
- **Substitutions**: $\text{sub}[x, y] = \frac{\text{count (x typed as y)}}{\text{count (x)}}$
- **Transpositions**: $\text{swap}[x, y] = \frac{\text{count (xy typed as yx)}}{\text{count (xy)}}$

Error Model: Noisy Channel Model

If the typed word is **already correct**, this model is prone to **overcorrection.**

Hence, for **real-word spelling errors** the model, the **likelihood estimation** is tweaked.

We introduce a parameter, α, representing **P(w|w)**, the probability of writing a word correctly.

Given α we redistribute evenly $1 - \alpha$ across every candidate correction **C(x)**. Doing so, the probability of having a correct real word is higher that the possible generated candidates, and depends mostly on α and **P(w).**

$$P(x|w) = \begin{cases} \alpha & \text{if } x = w \\ \frac{1-\alpha}{|C(x)|} & \text{if } x \in C(x) \\ 0 & \text{otherwise} \end{cases}$$

Error Model: Noisy Channel Model

Examples:

Output of `hmm.candidates(typed)`, from `hmm(max_edits=2, max_candidates=5)`

Typed: migt

Candidates:

- might, p = 1.03e-05
- mist, p = 6.19e-07
- light, p = 3.59e-07
- night, p = 2.75e-07
- right, p = 2.06e-07

Non-word spelling error

Typed: hoem

Candidates:

- home, p = 0.02
- hoom, p = 4.21e-08
- does, p = 1.72e-08
- holes, p = 1.60e-08
- seem, p = 9.31e-09

Non-word spelling error

Typed: time

Candidates:

- time, p = 0.001
- tide, p = 3.15e-09
- lime, p = 2.69e-09
- tire, p = 8.29e-10
- tame, p = 6.63e-10

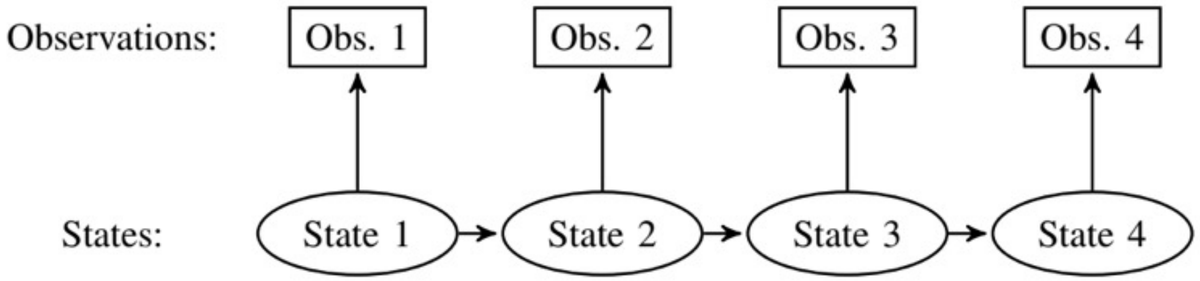
Real-word spelling error

Hidden Markov Model

Having for each word a list of candidate corrections, we are trying to find what is the **most likely sequence** of candidate states that represents the input sentence.

The **HMM** is modeled as follows:

- **Q**, set of hidden states
- **A**, state transitions probability matrix
- **B**, emission probabilities of an observation **o**
- π**,** initial probability distribution



Hidden Markov Model

The hidden states **Q** in the model are effectively the **candidates** generated for each typed word.

The state transitions **A** are extracted from the **Sentences Dataset.**

The initial probability π is extracted from the **Language Dataset**.

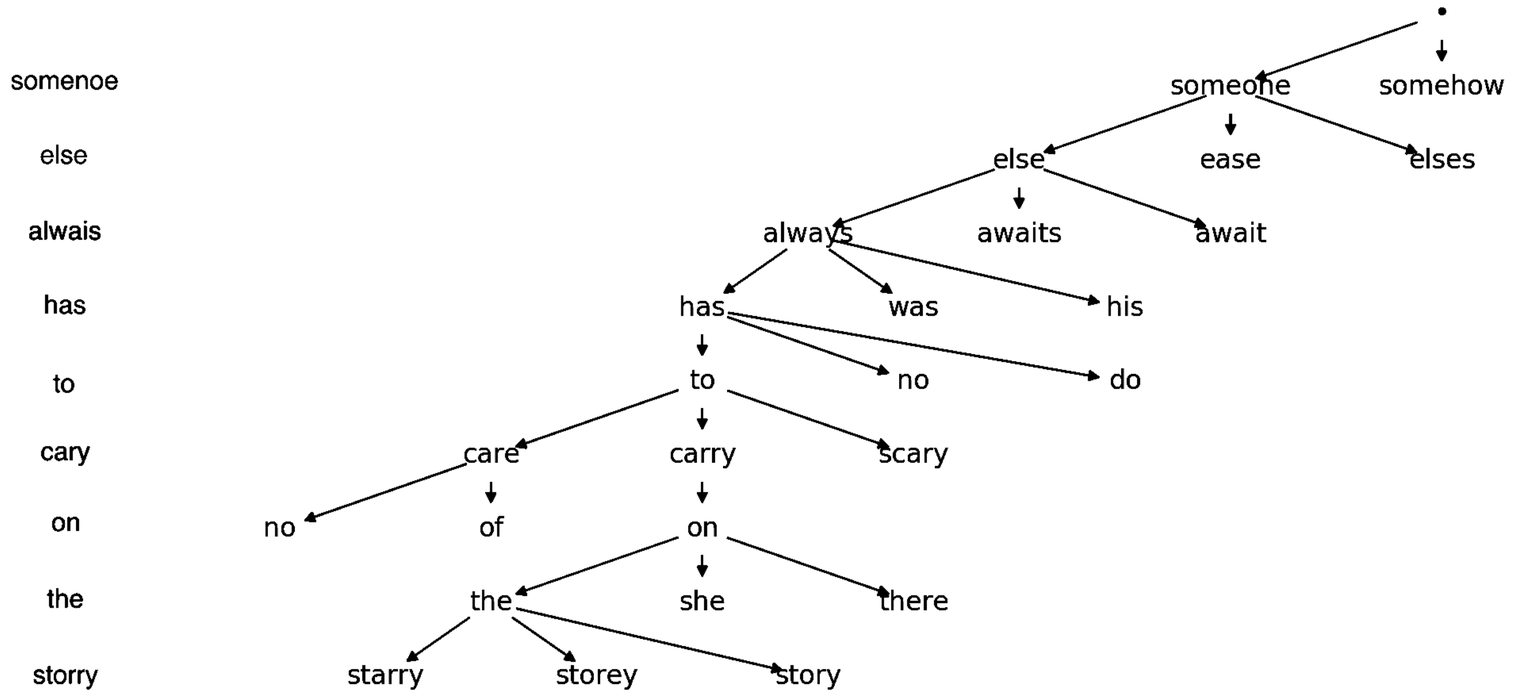
The set of observations **O** isn't **directly part of the model**.

As such, the emission probability is calculated as **P(typed|intended)**, where **typed** is the current observation read from the sentence, and **intended** is the current **state** being evaluated.

This computation is done through the **Noisy Channel Model** explained previously.

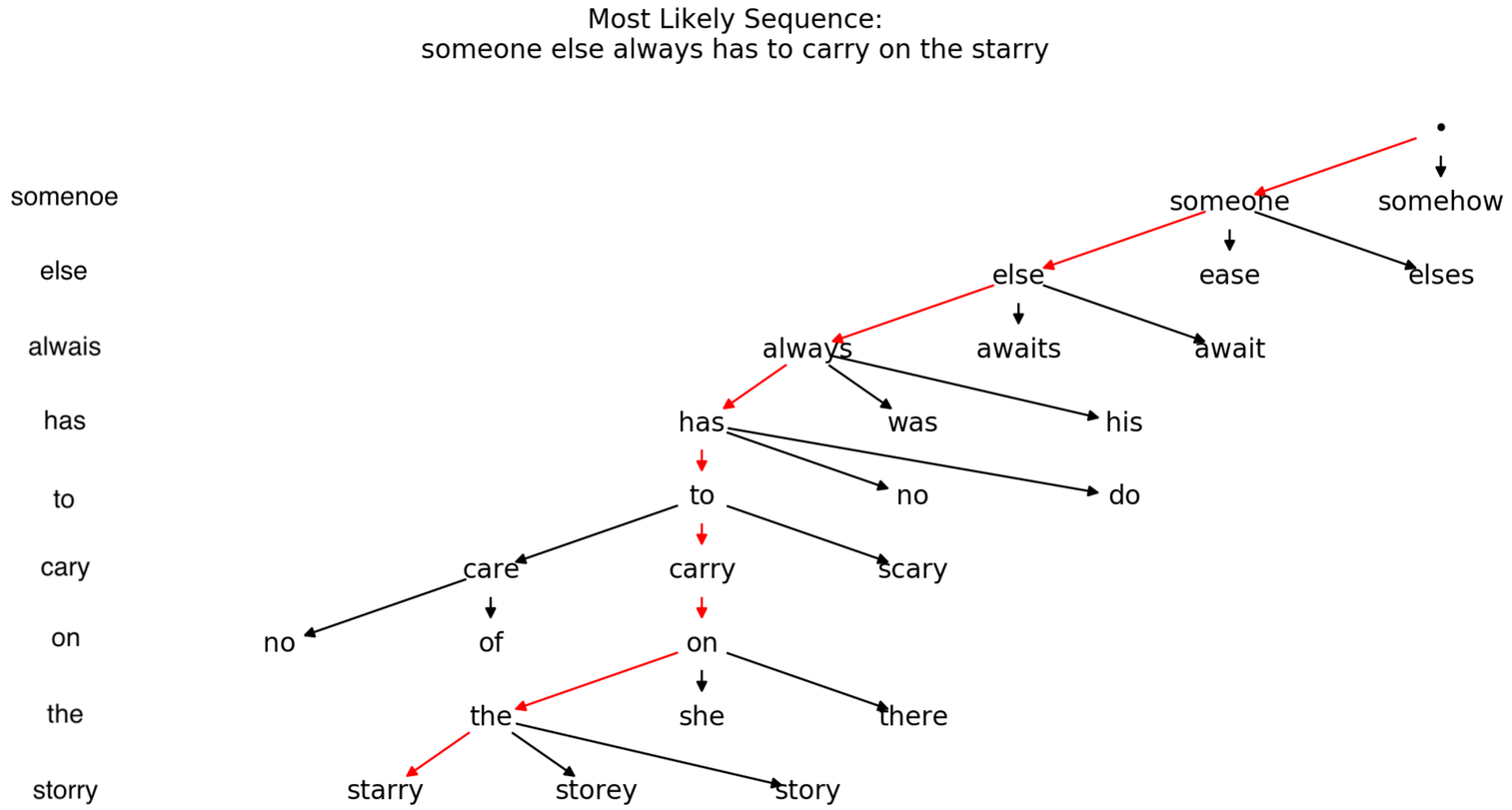
Hidden Markov Model

Being typing **sequential**, for every state corresponding to an observation O_i, we can only go to the states corresponding to O_{i+1}, generating a graph in the shape of a trellis.



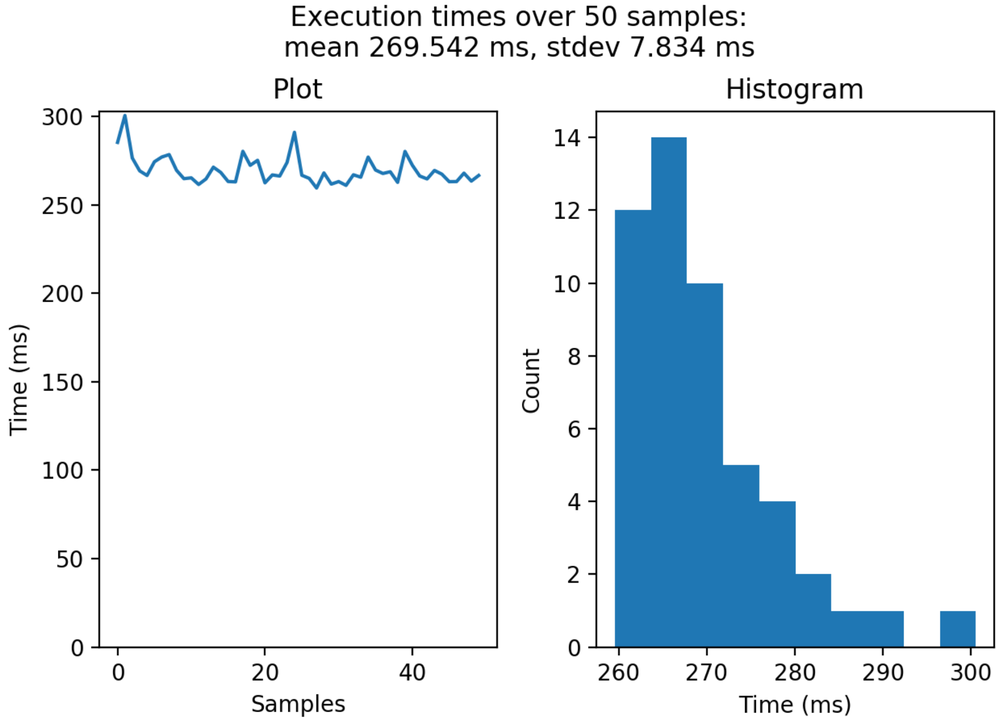
Most Likely State Sequence

Selecting between the final states of the trellis the one with the **highest global probability**, and following back the trellis, gives us the **most likely state sequence**.

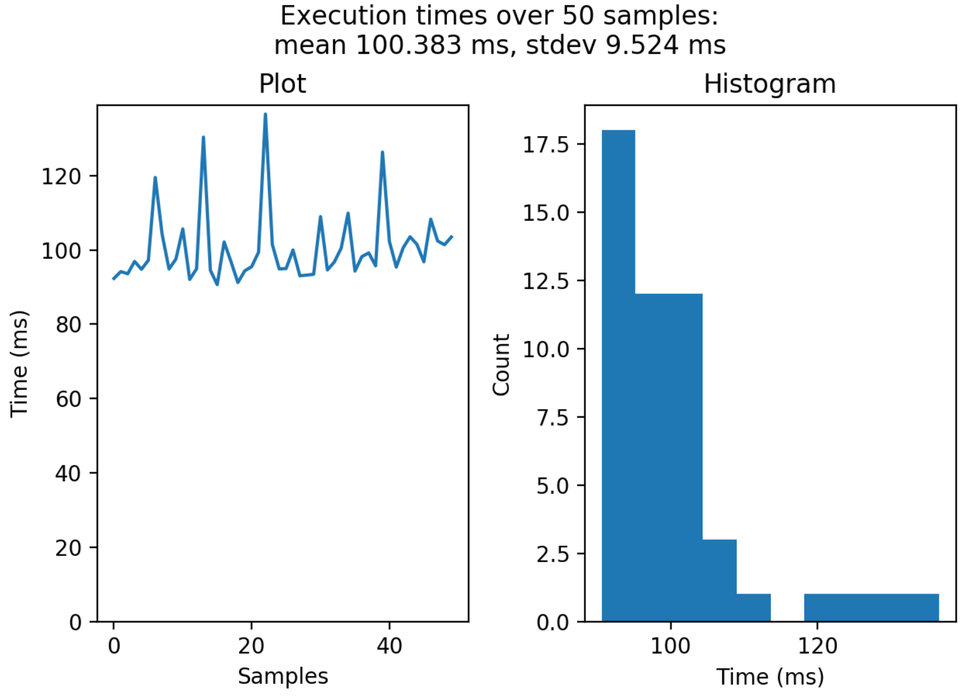


Multiprocessing

To make possible the **real-time using** of the model in the **graphical interface** and to improve the evaluation time, we have parallelised the generation of the candidates obtaining a **speed up of almost 3 times.**



Serial



Parallel (4 CPUs)

Experiments and Results

Evaluation Metrics

Local Correction

We evaluate the performance of the model on individual typos through various measures of accuracy.

In particular, we compute the **Top-1 Accuracy**, comparing the misspelt word and the best candidate predicted by the model. Then we compute the **Top-3 Accuracy** and **Top-5 Accuracy**, comparing the misspelt word with the first 3 and 5 candidates produced by the model, respectively.

Global Correction

For each *perturbed*, *predicted* and *intended* sentence and for each word we verify if it was *perturbed* or not and if it corresponds to the *original truth*.

For each word the following cases can occur:

1. Perturbed word, not correctly predicted
   (a) the model did not attempt to correct the word
   (b) the model attempted to correct the word, but without success
2. Perturbed word, correctly predicted
3. Unperturbed word, not correctly predicted
4. Unperturbed word, correctly predicted

From these four cases we can construct two confusion matrices that represent, respectively, the ability of the model to **detect errors** and the ability to provide the **right corrections**.

Evaluation Metrics

Global Correction

		Model Prediction	
		Detected	Not Detected
Hidden Truth	Perturbed	True Positive Case 1(b). & Case 2.	False Negative Case 1(a).
	Unperturbed	False Positive Case 3.	True Negative Case 4.

Confusion matrix - Error Detection

		Model Prediction	
		Corrected	Not Corrected
Hidden Truth	Perturbed	True Positive Case 2.	False Negative Case 1.
	Unperturbed	False Positive Case 3.	True Negative Case 4.

Confusion matrix - Error Correction

- **Detection-Accuracy**: percentage of words where the model prediction matches the ground-truth

$$\frac{\text{True Positive} + \text{True Negative}}{\sum \text{All}} = \frac{\text{Case 1(b).} + \text{Case 2.} + \text{Case 4.}}{\text{Case 1.} + \text{Case 2.} + \text{Case 3.} + \text{Case 4.}}$$

- **Detection-Recall**: proportion of the detected errors among all the errors

$$\frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} = \frac{\text{Case 1(b).} + \text{Case 2.}}{\text{Case 1.} + \text{Case 2.}}$$

- **Detection-Precision**: ratio of the correct detections with respect to all detections

$$\frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} = \frac{\text{Case 1(b).} + \text{Case 2.}}{\text{Case 1(b).} + \text{Case 2.} + \text{Case 3.}}$$

- **Correction-Accuracy**: percentage of words where the model prediction matches the ground-truth

$$\frac{\text{True Positive} + \text{True Negative}}{\sum \text{All}} = \frac{\text{Case 2.} + \text{Case 4.}}{\text{Case 1.} + \text{Case 2.} + \text{Case 3.} + \text{Case 4.}}$$

- **Correction-Recall**: rate of perturbed words correctly predicted

$$\frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}} = \frac{\text{Case 2.}}{\text{Case 2.} + \text{Case 1.}}$$

- **Correction-Precision**: ratio of the corrected predictions with respect to the significant values returned by the model (the corrections made)

$$\frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}} = \frac{\text{Case 2.}}{\text{Case 2.} + \text{Case 3.}}$$

- **Specificity**, that is the rate of unperturbed words correctly predicted:

$$\frac{\text{True Negative}}{\sum \text{Unperturbed}} = \frac{\text{Case 4.}}{\text{Case 3.} + \text{Case 4.}}$$

Experiment 1

The first experiment was carried out on two different HMM that differ in the choice of the `max_edits` parameter: 1 in the fist case and 2 in the second.

In both cases the HMM is structured as follows:

max states	language model	sentence ds	train typo ds	test typo ds
5	`big_language_model`	`big_clean`	`big_train`	`big_test`

Local Evaluation

	Edit Distance 1		Edit Distance 2	
	`big_train`	`big_test`	`big_train`	`big_test`
Num. observation	63 759	15 918	63 759	15 918
Time	54s	14s	1660s	419s
Accuracy Top1	34.96%	35.46%	37.34%	37.95%
Accuracy Top3	46.05%	46.41%	49.01%	49.25%
Accuracy Top5	50.18%	50.52%	53.40%	53.65%

Global Evaluation

	Edit Distance 1		
Dataset Perturbation	10%	20%	30%
Time	40s	38s	39s
Exact Sentence Match	27.37%	20.98%	13.19%
Detection Accuracy	90.39%	89.04%	86.79%
Detection Recall	82.35%	80.94%	77.42%
Detection Precision	55.15%	69.22%	76.67%
Correction Accuracy	88.75%	85.96%	81.66%
Correction Recall	67.17%	65.71%	58.67%
Correction Precision	49.34%	64.36%	70.68%
Specificity	91.48%	90.99%	90.56%

...

	Edit Distance 2		
Dataset Perturbation	10%	20%	30%
Time	223s	225s	226s
Exact Sentence Match	24.98%	20.88%	16.48%
Detection Accuracy	89.58%	89.61%	89.33%
Detection Recall	92.63%	92.37%	90.35%
Detection Precision	53.47%	68.58%	77.01%
Correction Accuracy	86.99%	84.94%	82.48%
Correction Recall	68.74%	68.97%	65.29%
Correction Precision	45.51%	61.74%	70.35%
Specificity	89.29%	89.08%	89.07%

Experiment 2

In our experimentation we found that a certain candidate word, independently from its rarity, would be **missing** from our language model.

We introduced **lemmatisation** to try solving this issue: lemmatisation takes
into consideration the morphological analysis of the words to link a word
back to its **lemma**.

Introducing this change gave us an improvement of about **~2%** overall accuracy on the models, but also, introduced different problems.

Problems:

- In some cases the lemmatisation algorithm used a somewhat *heuristic approach*, **validating wrong words** to a correct lemma. (e.g. validating bookses as book, even though bookses is not an existing word)
- Loss of **nuance** among different words derived from a single lemma, which may have a different frequency of use (e.g. archaic conjugations vs modern conjugations)
- Introduction of a **large time overhead** in the generation of candidates, way more than **10 times** the initial required time

Experiment 3

This experiment tries to investigate the performance of a model tailored to a specific writing style.

The third experiment too was carried out on two different HMM with edit distances 1 and 2.

In both cases the HMM is structured as follows:

max states	language model	sentence ds	train typo ds
5	`lotr_language_model`	`lotr_clean`	`big_train`

Local Evaluation

	Edit Distance 1		Edit Distance 2	
	`big_test`	`lotr_test`	`big_test`	`lotr_test`
Num. observation	15 918	12 570	15 918	12 570
Time	13s	10s	374s	312s
Accuracy Top1	28.28%	40.37%	28.06%	62.26%
Accuracy Top3	39.00%	45.27%	37.98%	75.90%
Accuracy Top5	45.14%	46.92%	42.15%	80.39%

Global Evaluation

	Edit Distance 1		
Dataset Perturbation	10%	20%	30%
Time	23s	23s	23s
Exact Sentence Match	78.82%	67.63%	59.04%
Detection Accuracy	99.30%	98.42%	97.68%
Detection Recall	90.57%	87.00%	87.36%
Detection Precision	97.17%	98.27%	98.48%
Correction Accuracy	98.90%	97.23%	97.68%
Correction Recall	82.09%	76.21%	75.81%
Correction Precision	96.63%	97.81%	98.06%
Specificity	99.86%	99.83%	99.69%

...

	Edit Distance 2		
Dataset Perturbation	10%	20%	30%
Time	173s	175s	174s
Exact Sentence Match	79.42%	69.53%	63.94%
Detection Accuracy	99.57%	99.24%	98.81%
Detection Recall	94.09%	93.25%	93.35%
Detection Precision	99.23%	99.79%	99.61%
Correction Accuracy	98.95%	97.60%	96.75%
Correction Recall	81.51%	78.75%	79.97%
Correction Precision	98.85%	99.64%	99.61%
Specificity	99.95%	99.96%	99.93%

Conclusions

Conclusions

The approach we presented allows us to obtain **acceptable performance**, but it is strongly linked to the **quality** and **selection** of the **datasets** of both models. In particular it is important to use larger Language Models.

For the models to work optimally each dataset must **share the same language.**

Future Works

- Improve the various datasets: in particular, we could use a **new transition probability dataset** that is consistent with the language model one.
- Compute the **initial probabilities** π from the language model.
- Consider **additional types of errors** in our model.
- Extend the noisy channel error model to also consider **neighbouring characters**.
- Evaluate the performance of other algorithms, such as the **Forward-Backward** (smoothing) algorithm.
- Evaluate the performance of other types of models, such as **Recurrent Neural Networks**.
- Improve the performance of the noisy channel model by **changing how the prior and the likelihood are combined**.
  In the standard model they are just multiplied together.

Thanks for your attention!

References

- James H Martin and Daniel Jurafsky. Speech and language processing: *An introduction to natural language processing, computational linguistics, and speech recognition*. Pearson/Prentice Hall Upper Saddle River, 2009 (cit. on pp. 10, 11, 24).
- Adrian Tarniceriu, Bixio Rimoldi, and Pierre Dillenbourg. "*HMM-based error correction mechanism for five-key chording keyboards*". In: 2015 International Symposium on Signals, Circuits and Systems (ISSCS). IEEE. 2015, pp. 1–4 (cit. on pp. 10, 15).
- Eric Brill and Robert C Moore. "*An improved error model for noisy channel spelling correction*". In: Proceedings of the 38th Annual Meeting on Association for Computational Linguistics. Association for Computational Linguistics. 2000, pp. 286–293 (cit. on p. 23).
- Yanen Li, Huizhong Duan, and ChengXiang Zhai. "*A generalized hidden markov model with discriminative training for query spelling correction*". In: Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval. ACM. 2012, pp. 611–620.
- Grzegorz Szymanski and Zygmunt Ciota. "*Hidden Markov models suitable for text generation*". In: WSEAS International Conference on Signal, Speech and Image Processing (WSEAS ICOSSIP 2002), pp. 3081–3084.