UNIVERSITÀ DEGLI STUDI DI MILANO BICOCCA

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

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent decades, technology has had a strong impact on everyone's life. It plays an important role in the communication process, simplifying different activities for both individuals and businesses.

Nowadays we have advanced communication technology tools available, such as smartphones, tablets and computers that have simplified the way humans communicate.
Companies can write an e-mail and deliver it to all their consumers in a few minutes. People can message their friends at every moment and share an interest with new friends from different countries.

This advancement in communication technology has made it necessary to equip our technological tools with a series of programs and software that control and correct automatically the misspelt words typed.

In this project, we propose and evaluate an automatic spelling correction algorithm, modelling the typing process as an *Hidden Markov Model* (HMM).

FIXME: keyboard?

Chapter 2

Problem Formulation

FIXME: Given an observation sequence, typically a phrase, and the model parameters, we are interested in detect and correct errors estimating the optimal state sequence.

In our HMM, the hidden states represent the intended words and the observations are the typed words.

The initial state probabilities π are given by the word frequencies and the state transitions A_{ij}, that is the probability of one word given its predecessor obtained from ??**FIXME**

The emission probabilities, B_{ij}, represent the probability of a typed word given the intended one, and depend on the confusion probabilities.

FIXME: We also detect the non-word error?

2.1 Purpose

2.2 Design Choices

We have chosen the English language for different reasons. First of all, the great majority of material in literature deals with the problem in question in the English languag. Moreover it is a simple language, both from a grammatical and a lexical point of view: it lacks in certain symbols, like accents and apostrophes, and genres. Furthermore all punctuation and special character symbols were not considered, but only letters and sometimes numbers.

We considered that a typed word only depends on the previous one, being in the framework of Markov chains. If we know the probability of a word given its predecessor, the frequency of each word, and the probability to type word x when word y is intended, we have all the necessary ingredients to use Hidden Markov Models.

2.3 Software

FIXME We have developed the project in **Python**.
The interface is a native macOS application written in **Swift**.

Chapter 3

Dataset

3.1 Dataset Acquisition

3.1.1 Transition Model

We performed our experiments on two different transition models.
The first one is a concatenation of public domain book excerpts from Project Gutenberg, containing about a million word.

The second one has been extracted from the json file LordOfTheRingsBook.json containing the collections of the "Lord of the rings" book.

To each corpora we applied some preprocessing procedures, in particular we divided them in lower-case sentences, and cleaned from special characters and punctuation, obtaining the datasets `big_clean.csv` and `lotr_clean.csv` (**FIXME** something about apostrophes)

FIXMEHow we use these?

3.1.2 Error Model

The basic error model dataset was collected from the following resources [1] [2] [3] [4]:

- BIRKBECK: contains 36 133 misspellings of 6136 words, taken from the native-speaker section (British and American) of the Birkbeck spelling error corpus.
- HOLBROOK: contains 1791 misspellings of 1200 words, taken from the book "English for the Rejected" by David Holbrook (Cambridge University Press - 1964).

[1] `https://www.dcs.bbk.ac.uk/~ROGER/corpora.html`
[2] `https://www.kaggle.com/rtatman/spelling-variation-on-urban-dictionary`
[3] `https://www.kaggle.com/bittlingmayer/spelling`
[4] `http://luululu.com/tweet`

- ASPELL: contains 531 misspellings of 450 words, taken from one assembled by Atkinson for testing the GNU Aspell spellchecker.
- WIKIPEDIA: contains 2455 misspellings of 1922 words, taken from the misspellings made by Wikipedia editors.
- URBAN-DICTIONARY-VARIANTS: contains 716 variant spellings, taken from the text scraped from Urban Dictionary (in UK English).
- SPELL-SET: contains 670 typos.
- TWEET-TYPO: contains 39 172 typos, taken from Twitter.

All the datasets are joined and cleaned. After that, it contains 79 677 row, in each of which there is the typo and the correct word. The dataset obtained is divided into two corpora: 80% is used as a train set (63 679 rows) and 20% is used as a test set (15 998 rows).

We decided to created another dataset of typos starting from the `lotr_clean.csv`. Extracting a list of all the words contained in this corpus, for each of these we have generated a sequence of five typos according to the algorithm that will be defined in the chapter 3.1.4. The final dataset contains 62 759 row, with the same structure as the one described above. Also in this case the two datasets train and test were created, respectively containing 50 058 and 12 701% rows.

3.1.3 Language Model

We used two different language model datasets.
The first one is a lists of most frequent words from Wiktionary and the British National Corpus. We use `frequency-alpha-gcide.txt`, a smaller version derived from the original dataset Google's ngram corpora, that includes wordlists, cleaned up and limited to only the top 65 538 words.

With this dataset we found some problems, for example the lack of proper names, city names, countries, brands etc. Moreover, most of the typical words of the language used in the sentence dataset were missing. For this reason, we decided to create a new language model `lotr_language_model.txt`, based on the frequency of the word in the dataset `lotr_clean.csv`.

Each of these datasets contains, for each row of the corpus, the ranking of the word, the word itself, the count of the occurrences of the word, a percentage of how often each word was being used and a cumulative percentage.

3.1.4 Perturbated Dataset

In order to evaluate our algorithm on whole sentences, we create new perturbed datasets starting from the datasets `big_clean` and `lotr_clean` described in the section 3.1.1.

FIXME:Estimates for the frequency of spelling errors in human-typed text vary from 1-2% for carefully retyping already printed text to 10-15% for web queries. The disturbance introduced presents an error dependent on the error model previously presented, with the difference that it is created starting from the typos belonging to the test datasets. **FIXME**:Analysis of spelling error data has shown that the majority of spelling errors consist of a single-letter change and so we often make the simplifying assumption that these candidates have an edit distance of 1 from the error model.

Three different texts for each dataset, of approximately 50 000 sentences each, have been generated, each of which has a percentage of errors in the text of **FIXME** 10-15-20% respectively.

We implemented a perturbation algorithm, which for each line of our input file generates a new perturbed string.
The input text is perturbed accordingly the following steps:

1. The probability that a word has an edit is computed by multiplying the value of p, coming from the error model, by the percentage of errors desired (10-15-20%).

2. For each word of length n, the number of edits to be introduced x is calculated according to the relation $x \sim \text{Bin}(n, p)$ **FIXME**: why? Assumptions?

3. The characters to be changed within each word are chosen randomly.

4. **FIXME**: The type of edit to be introduced within each word is chosen according to the probability of each type of error. The disturbance goes to alter the letters designated not in a random manner. In fact, we use four different probabilities to define whether a letter will be deleted from the index in question, if a new letter will be inserted after the actual character, or if the current character will be replaced with one of the possible letters according to the error model probability, or if the current character will be swapped with the next or the previous one.

Swap errors are only introduced if there are no further changes in the word. Cases of elimination of a whole word are excluded, as these would heavily influence the evaluation metrics as they are inconsistent with our model.

Chapter 4

Models

We will consider two optimality criteria. The first one chooses the states that are individually most likely and maximizes the expected number of correct individual states. The second criterion estimates the most likely state sequence, or *trellis path*. The algorithm used to implement these criteria is the **Viterbi** algorithm.

4.1 Hidden Markov Model

FIXME: Hmm approach **FIXME**: description of hmm

FIXME: our application ...

A **Hidden Markov Model** (HMM) allows us to talk about both observed events, like misspelled words that we see in the input, and hidden events, like the intended words, that we think of as causal factors in our probabilistic model.

Our HMM is specified by the following components:

- $Q = q_1 q_2 \dots q_N$: a set of N **states**
- $A = a_{11} \dots a_{ij} \dots a_{NN}$: a **transition probability matrix** A.
  Each a_{ij} representing the probability of moving from state i to state j, such that $\sum_{j=1}^{N} a_{ij} = 1 \quad \forall i$
- $O = o_1 o_2 \dots o_T$: a sequence of T **observations**, each one drawn from a vocabulary $V = v_1, v_2, \dots, v_V$
- $B = b_i(o_t)$: a sequence of **observation likelihoods**, also called **emission probabilities**, each expressing the probability of an observation ot being generated from a state i
- $\pi = \pi_1, \pi_2, \dots, \pi_N$: an **initial probability distribution** over states. π_i is the probability that the Markov chain will start in state i.

FIXMEAggiungere immagine

We consider a *first-order* Hidden Markov Model, that instantiates two simplifying assumptions. First, as with a first-order Markov chain, the probability of a particular state depends only on the previous state. Second, the probability of an output observation o_i depends only on the state that produced the observation q_i and not on any other states or any other observations.

4.2 Noisy Channel Model

Non-word errors are detected by looking for any word not found in a dictionary. To correct them we first generate **candidates**, according to a distance given as a parameter to the model (`edit_distance`), that are real words with a similar letter sequence to the error.

The intuition of the noisy channel model is to treat the misspelled word as if a correctly spelled word had been "distorted" by being passed through a noisy communication channel. This channel introduces "noise" in the form of substitutions or other changes to the letters, making it hard to recognise the "true" word.

We see an observation x (a misspelled word) and we want to find the word w that generated this misspelled word (the intended word). Out of all possible words in the vocabulary V we want to find the word $\hat{w}$ such that $P(\hat{w}|x)$ is highest among all the candidates w

$$\hat{w} = \arg\max_{w \in V} P(w|x). \tag{4.1}$$

This noisy channel model is, therefore, a kind of Bayesian inference, in which it is possible to transform the equation 4.1 into a set of other probabilities

$$\hat{w} = \arg\max_{w \in V} \frac{P(x|w)P(w)}{P(x)}. \tag{4.2}$$

Since $P(x)$ doesn't change for each word because we are always asking about the most likely word for the same observed error x, we can conveniently simplify the equation 4.2 by dropping the denominator

$$\hat{w} = \arg\max_{w \in V} P(x|w)P(w). \tag{4.3}$$

FIXME: We apply the noisy channel approach to correcting non-word spelling errors by taking any word not in our spell dictionary, for example *adventhre*, generating a list of candidate words like *adventure*, *adventurer*, *adventured*, ranking them according to 4.3, and picking the highest-ranked one, *adventure*. **FIXME**: We choose the most likely candidate, the one with the highest probability.

The two components of the equation are, respectively, $P(x|w)$ the **channel model** and $P(w)$ the prior probability of a hidden word (a candidate). The prior probability of each correction is the language model probability of the word w in context, which is computed using a **FIXME** *unigram* language model. The likelihood is estimated just using the number of times that the a letter i was substituted for the letter j in the large corpus of errors **FIXME** QUALE=? To compute the probability for each edit we used a confusion matrix that contains counts of errors. **FIXME** How?

Each candidate is scored by $P(\text{candidate})P(\text{typo}|\text{candidate})$ and then normalised by the sum of the scores for all proposed candidates.

FIXME: (How is estimated the prior? How are computed the conditional probabilities?)

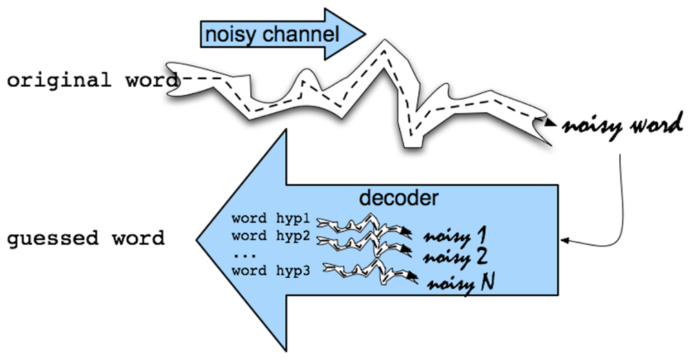

Figure 4.1: Diagram of the Noisy Channel Model

FIXME: When for a input typo we do not have any candidate ??

FIXME: Real word spelling error detection is a much more difficult task, since any word in the input text could be an error. So we don't...

To find this list of candidates we uses a minimum edit distance algorithm in the extended version with transposition called **Damerau-Levenshtein** edit distance ??.

The following type of errors were considered in our model:

- CHARACTER DELETIONS
- CHARACTER INSERTIONS
- SUBSTITUTION OF CHARACTERS: this is the probability to type a character i when the character j was intended ($P(i|j)$). This probability was determined experimentally. **FIXME**: how? from where?

- TRANSPOSITION OF ADJACENT CHARACTERS: two letters are swapped.

4.3 Most Likely State Sequence

The Viterbi algorithm calculates the most probable sequence of hidden states, the words intended. The Viterbi algorithm is a probabilistic extension of minimum edit distance. Instead of computing the "minimum edit distance" between two strings, Viterbi computes the "maximum probability alignment" of one string with another.

The initial probability of being in a state i, π_i, in our case the probability of intend a word i, and the transition probabilities A_{ij}, or the transition from the word i to the next word j, are given. Since we have observed the output $y_1, y_2, \dots, y_t$, that is the sentence written with typos, it is possible to computed the most likely state sequence $x_1, x_2, \dots, x_t$, the sentence intended, starting from the following expression:

$$\begin{aligned}
V_{1,t+1} &= P(x_1, \dots, x_t, x_{t+1}, y_1, \dots, y_t, y_{t+1}) = \\
&= \arg\max_{x_{1:t}} p(x_1, \dots, x_t | y_1, \dots, y_t) = \\
&= \alpha \cdot p(y_{t+1}|x_{t+1}) \cdot \max_{x_t} \Big(p(x_{t+1}|x_t) \max p(x_1, \dots, x_t | y_1, \dots, y_t) \Big)
\end{aligned} \tag{4.4}$$

The initial state probabilities π are actually the word frequencies (? We don't estimate it in a proper way), the state transition probabilities are given by the probability of a word given its predecessor, **FIXME**: come vengono prese and the emission probabilities are the probabilities to type word i when word j was intended.

In our implementation, we construct the *trellis* choosing the **FIXME**: locally/-globally best state. As we can see in the picture below, we display an example of the trellis drawing only the best predecessor of each state.

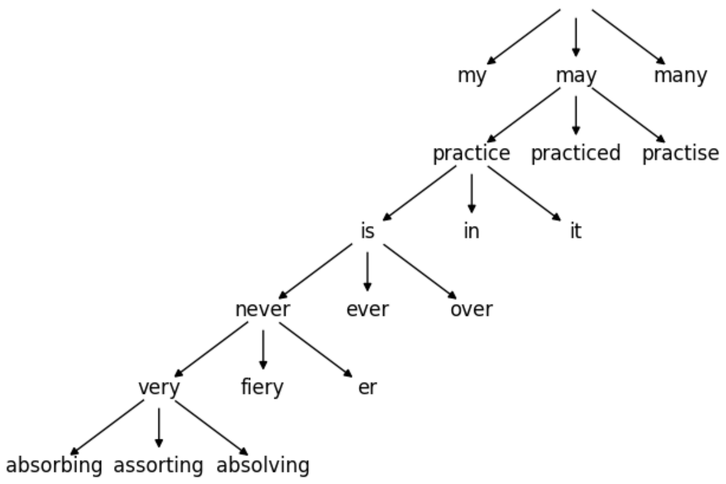

Figure 4.2: Trellis example

In this example, we observed the sentence *my practice iu never iery absorcing* and our algorithm return the most likely state sequence *may practice is never very absorbing*. In this case, the algorithm partially fails, because the intended sentence was *my practice is never very absorbing*.
FIXME: (pi la probabilità iniziale del prima stato non lo abbiamo, non lo facciamo perchè dal nostro dataset non abbiamo sempre la prima parola. . .)

We decide to implement the **Viterbi** based algorithm instead the Forward-Backward algorithm relying on the experiments carried out in the literature. The HMM-Based Error Correction Mechanism for Five-Key Chording Keyboards article [1] explains that the Forward-Backward algorithm estimates the most likely state for each observation, but the resulting state sequence may not be a valid succession of words in natural language (or a very unlikely word sequence) and produce inferior results.

Chapter 5

Experiment and Results

FIXME comparison **FIXME** performance

5.0.1 Evaluation Metrics

We evaluate the local model performances, or that carried out on individual typos, through various measures of accuracy. We compute the TOP-1 ACCURACY comparing the misspelt word and the best candidate predicted by the model. Than we also compute the TOP-3 ACCURACY and TOP-5 ACCURACY, comparing the misspelt word with the first N candidates, in this case 3 and 5, given from the model.

As regards the performance evaluation of the entire sequences, after every training and prediction we save a csv file containing the sentences with the perturbations, the sentences that we want to predict (hidden truth) and the sentences provided by the model.

The idea to evaluate the performance of the model is very simple: we scroll through the lines containing the perturbed, the predicted and the original (intended) text and, for each word we verify if it was disturbed or not and if it corresponds to the original truth. Therefore, for each word, the following cases can occur:

1. *Perturbed word not correctly provided*
   (a) the word was not the subject of attempted correction by the model
   (b) the word has been the subject of attempted correction by the model but without success
2. *Perturbed word correctly provided*
3. *Unperturbed word not correctly provided*

4. *Unperturbed word correctly provided*

We can therefore represent the confusion matrix of the data in output through table 5.1:

		Model Prediction	
		TRUE	FALSE
Hidden Truth	TRUE	True Positive Case 2.	False Negative Case 1(a).
	FALSE	False Positive Case 3. & 1(b).	True Negative Case 4.

Table 5.1: Confusion matrix

From the confusion matrix defined in the table above, it is possible to calculate the following performance metrics in a simple way:

- RATE OF PERTURBED WORDS CORRECTLY PREDICTED:

$$\frac{\text{True Positive}}{\sum \text{Perturbed}} = \frac{\text{Case 2.}}{\text{Case 2.} + \text{Case 1.}}$$

- RATE OF UNPERTURBED WORDS NOT CORRECTLY PREDICTED:

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

- ACCURACY: measures the goodness of the model among the positive correction results obtained

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

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

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

- RECALL: proportion of all the correct results

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

- F-Measure: weighted average (between 0 and 1) with respect to precision and recall
$$2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

Based on these metrics, it is then possible to define whether the models have behaved more or less correctly.

5.0.2 Experiments

We performed three different types of experiments.

The first one using as a transition model the dataset `big_clean`, the associate perturbed dataset and test error models, and the language model `frequency-alpha-gcide`.

A second one using the same datasets but introducing a lemmatisation consisting in a simple dictionary lookup.

The last one using as transition model the dataset `lotr_clean`, the associate perturbed dataset and test error models, and the language model `lotr_language_model`.

In all the experiments to come, reference will be made to the following variables:

- p: the probability that a word has an edit
- ins: the probability that a word has a letter insertion
- del: the probability that a word has a letter deletion
- sub: the probability that a word has a letter substitution
- swap: the probability that a word has a swap between two letters

Experiment 1

The first experiment was carried out on two different hmm that differ in the choice of the `max_edits` parameter. In the first case, in fact, the edit distance considered was 1, while in the second 2. 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`

Table 5.2: Hmm model

In the two tables to follow are shown the results obtained as regards the evaluation of the *local correction* of our model on the typos dataset and the correction of the entire sequence with the *Viterbi algorithm.*

	Edit Distance 1		Edit Distance 2	
	Train	Test	Train	Test
Num. observation	63 759	15 918	63 759	15 918
Time (sec)	53	14	1648	408
Accuracy Top1	35.01%	35.60%	36.09%	36.84%
Accuracy Top3	46.24%	46.62%	48.61%	48.99%
Accuracy Top5	50.19%	50.56%	52.86%	53.16%

Table 5.3: Typos performance evaluation

	Edit Distance 1			
Dataset Perturbation	10%	20%	30%	40%
Time (sec)	201	199	191	186
Perturbed correct	64.06%	60.16%	57.30%	52.81%
Unperturbed not correct	41.70%	43.36%	44.84%	46.58%
Exact match	2.60%	2.32%	1.56%	1.42%
Accuracy	58.82%	57.47%	55.73%	42.60%
Precision	16.00%	25.12%	30.74%	33.75%
Recall	79.66%	77.36%	74.30%	70.40%
F-Measure	46.28%	48.49%	49.65%	49.82%

...

	Edit Distance 2			
Dataset Perturbation	10%	20%	30%	40%
Time (sec)	1110	1121	1116	1121
Perturbed correct	54.96%	55.47%	54.34%	51.76%
Unperturbed not correct	55.19%	56.31%	57.52%	58.43%
Exact match	0.90%	1.00%	0.76%	0.80%
Accuracy	45.81%	46.00%	45.85%	45.14%
Precision	10.67%	18.45%	23.7%	27.22%
Recall	89.91%	90.94%	90.69%	88.21%
F-Measure	38.77%	41.55%	44.37%	46.18%

Table 5.4: Sentences performance evaluation

Experiment 2

p	ins	del	sub	swap
0.5	0.70	0.70	0.70	0.70

Table 5.5: Error Model

We detected some problems with our dataset, in particular it lacks of plural forms and other things

In order to avoid this problem, we decided to try a new approach that use lemmatisation (stemming) ......

	Time (sec)	Accuracy Top1	Accuracy Top3	Accuracy Top5
Train	20 986	41.38%	57.28 %	61.60 %
Test	5270	41.54%	57.18 %	61.15 %

Table 5.6: Typos performance evaluation

#sentence	Time (sec)	Accuracy	Initial Error	Precision	Recall	Specificity
1000	2705	53.46%	15.01%	90.96%	54.50%	10.52%

Table 5.7: Sentences performance evaluation

Experiment 3

As for the previous cases, the third experiment too was carried out on two different hmm with the same edit distances as the previous experiments. The parameters on which the structure is based are shown in the table below:

max states	language model	sentence ds	train typo ds	test typo ds
5	`lotr_language_model`	`lotr_clean`	`lotr_train`	`lotr_test`

Table 5.8: Hmm model

In the two tables to follow are shown the results obtained as regards the evaluation of the *local correction* of our model on the typos dataset and the correction of the entire sequence with the *Viterbi algorithm.*

	Edit Distance 1		Edit Distance 2	
	Train	Test	Train	Test
Num. observation	49 959	12 570	49 959	12 570
Time (sec)	32	9	1207	309
Accuracy Top1	39.75%	40.47%	63.71%	63.87%
Accuracy Top3	44.74%	45.34%	76.44%	76.75%
Accuracy Top5	46.25%	46.92%	80.86%	80.80%

Table 5.9: Typos performance evaluation

	Edit Distance 1			
Dataset Perturbation	5%	10%	15%	20%
Time (sec)	116	114	114	112
Perturbed correct	79.97%	76.55%	73.74%	71.76%
Unperturbed not correct	13.68%	14.46%	14.86%	15.69%
Exact match	30.89%	27.51%	25.23%	22.66%
Accuracy	85.99%	84.62%	83.32%	81.73%
Precision	28.99%	41.97%	48.90%	52.85%
Recall	91.19%	88.30%	86.09%	84.31%
F-Measure	74.19%	72.87%	72.44%	71.97%

...

	Edit Distance 2			
Dataset Perturbation	5%	10%	15%	20%
Time (sec)	866	869	854	849
Perturbed correct	73.26%	72.05%	71.10%	69.90%
Unperturbed not correct	18.47%	18.86%	19.46%	20.16%
Exact match	22.74%	21.26%	19.52%	18.38%
Accuracy	81.07%	80.17%	79.06%	77.79%
Precision	22.40%	34.49%	41.35%	45.37%
Recall	94.31%	94.16%	93.63%	93.43%
F-Measure	68.76%	68.60%	68.53%	68.58%

Table 5.10: Sentences performance evaluation

Chapter 6

User Interface

Chapter 7

Conclusions

it is important to use larger language models than unigrams

For this reason modern systems often use much larger dictionaries automatically derived from very large lists of unigrams like the Google N-gram corpus

7.1 Future Works

- update the various dataset: in particular we can use a new frequency dataset consistent with the language model one.
- Compute the initial probabilities π from the language model
- Consider additional type of error in the model
- improving the noisy channel error model to consider neighbouring characters - Use other algorithms, such as the forward-backward (smooothing) instead Viterbi
- Optimise the code, in particular the lemmatisation/stemming
- Use recurrent neural network instead the hidden markov models.
- We can also 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.

The fact that words are generally more frequent than their misspellings can be used in candidate suggestion, by building a set of words and spelling variations that have similar contexts, sorting by frequency, treating the most frequent variant as the source, and learning an error model from the difference

Bibliography

[1] 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 p. 14).

[2] 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.

[3] 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.

[4] 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.