Speech Recognition Final Project

An Weighted Finite State Machine Implementation of Alignment and Translation Models

Authors:

Andres Muñoz

David Álvarez-Melis

December 17, 2012

Contents

List of Figures

Abstract

We present a translation model based on weighted finite state machines. More precisely, we implement and train an IBM2 alignment model on a weighted transducer and use this transducer to translate sentences from French to English. We provide details of its characteristics.

1 Introduction

Machine translation has proven to be one of the most difficult tasks in Natural Language Processing [2]. The difficulties arising from this problem are countless: the sintaxis of languages is not homogeneous, same words translate to several different things and most of the time multiple words translate to one single word and viceversa.

Since the amount of bilingual text for some languages is huge an statistical approach is natural for attacking this problem and different attempts to solve it have been tried for example: alignment models [1], phrase based translations [4] and syntactic translation [6]. And the results obtained so far are quite impressive considering the hardness of the problem.

The statistical approach to machine learning can be set up as follows: given a sentence from a source langugage f we try to find the most likely sentence from the target language e:

$$e^* = \operatorname*{argmax}_e p(e|f)$$

Following the noisy channel approach we can recast this as

$$e^* = \operatorname*{argmax}_e p(f|e)p(e)$$

The two parts of the noisy channel model are the translation model ($p(f|e)$)

and the language model ($p(e)$). What we will do in this project is to implement an IBM 2 model for the translation part of the problem and use a Backoff model for evaluating the probability measure $p(e)$. In the rest of the paper we will call the source language french and the target language english. The rest of the paper is as follows: first we talk about the IBM models as word alignment tools. Then we present a way of encoding an IBM 2 model into a transducer and how to obtain alignment probabilities and how to evaluate these alignments. Finally we say how we can use this model as an automatic translator.

2 The IBM Models

In a seminal paper (Brown et al, 1993), a team of IBM reasearchers proposed a series of models for statistical alignment, which would become the foundations for many other machine translation techinques. The models are increasingly complex, and incorporate more features and less simplifications than the previous one. The second one, also known as IBM2, is simple enough to be easy to implement but at the same time it incorporates some crucial features shared by all the other models. It is this model that we study in this work.

The motivation for IBM2 comes when dealing with the problem of how to model $p(f_1 \dots f_m | e_1 \dots e_l, m)$. This conditional probability is very complicated to model directly. The idea that all IBM models share is to introduce additional alignment variables, and thus, to try to model

$$p(f_1 \dots f_m, a_1 \dots a_m | e_1 \dots e_l, m) \tag{2.1}$$

instead, where $a = (a_1 \dots a_m)$ is a vector of alignment variables. Here, $a_j = k$ means that the source language word f_j is aligned to the target word e_k. With this, we can effectively obtain the required probability by summing over the newly

introduced alignment variables as follows

$$p(f_1 \dots f_m | e_1 \dots e_l, m) = \sum_a p(f_1 \dots f_m, a_1 \dots a_m | e_1 \dots e_l, m)$$

The next step is to decompose the term in (2.1) into simpler terms. First, let us formalize the argument above by denoting with capital letters the random variables A, F, E and with lower-case letters the corresponding values that they take. Also, let us denote $f_1^m = f_1 \dots f_m$ and $e_1^l = e_1 \dots e_l$ to simplify the notation. Thus, we are interested in simplfying

$$p(F_1^m = f_1^m, A_1^m = a_1^m | E_1^l = e_1^l)$$

for which we can use the product rule to obtain

$$p(F_1^m = f_1^m, A_1^m = a_1^m | e_1^l) = p(A_1^m = a_1^m | E_1^l = e_1^l, L = l, M = m) \tag{2.2}$$

$$\cdot\, p(F_1^m = f_1^m | A_1^m = a_1^m, E_1^l = e_1^l, L = l, M = m) \tag{2.3}$$

For the first of these terms, we make the (strong) assumption that alignments are independent of particular words attached and of other alignments, and thus we can obtain

$$\begin{aligned} p(A_1^m = a_1^m | E_1^l = e_1^l, L = l, M = m) &= \prod_{i=1}^m p(A_i = a_i | A_1^{i-1} = a_1^{i-1}, E_1^l = e_1^l, L = l, M = m) \\ &= \prod_{i=1}^m p(A_i = a_i | L = l, M = m) \end{aligned}$$

Thus, we set

$$p(A_i = a_i | L = l, M = m) = q(a_i | i, l, m)$$

whre q is a probability function to be determined. There are many possibilities for this function. A common choice is to set it so as to penzlize excesive movement

of words during translation. That is, we seek to penalize words that are displaced far away from their original position. One possibility to implement this is to chose

$$q(a_i|i,l,m) = e^{-\alpha|i-j\frac{l}{m}|}$$

For the other term in (2.3) we make an equally strong independence assumption. We will suppose that the value of F_i depends only on the english word that is aligned to it. With this, we can write

$$\begin{aligned} p(F_1^m = f_1^m | A_1^m = a_1^m, E_1^l = e_1^l, L = l, M = m) = \\ = \prod_{i=1}^{m} p(F_i = f_i | F_1^{i-1}, A_1^m = a_1^m, E_1^m = e_1^m, L = l, M = m) \\ = \prod_{i=1}^{m} p(F_i = f_i | E_{a_i} = e_{a_i}) \end{aligned}$$

If we denote $p(F_i = f_i|E_{a_i} = e_{a_i}) = t(f_i|e_j)$, then our goal has been reduced to finding the transition probabilities $t(f_i|e_j)$. These are usually trained by using an Expectation-Maximization algorithm, by iterating on observed parallel sentences. The method can be initialized with $t(f_i|e_j)$ as uniform distributions, although it is a common practice to use a simpler model (in this case, IBM1) to get a "warm" start for the iterative process.

With this model built, we can find the most probable alignment under thw model by computing

$$\arg \max_{a_1 \ldots a_m} p(a_1^m | f_i^m, e_1^l, m)$$

which, in view of the previous analysis, yields

$$a_i = \arg \max_{j \in (0,\ldots,l)} q(j|i,l,m) \cdot t(f_i|e_j)$$

3 Alignment Model Implementation

In this section we present an implementation of the alignment model for machine translation. In Section 3.1 we list the Finite State Machines required for the task and display the Automata and Transducers for a simple example.

3.1 An FSM IBM Model 2

Suppose we are given a pair of translated sentences (e, f). For the purpose of exemplification, we will use the following simple pair of sentences in French and English:

$$e = \text{“}\textit{The red house}\text{”}, \qquad f = \text{“}\textit{La maison rouge}\text{”}$$

The first ingredient required in the FSM implementation of IBM2 are the auotmata encoding these sentences. We will denote them by $\mathcal{F}$ and $\mathcal{E}$. They consist simply of linear automata of lengths $|e| = l$ and $|f| = m$, respectively, with unweighted transitions labeled as the words $e_1 e_2 \dots e_l$ and $f_1 f_2 \dots f_m$. At this point we warn the reader that although $\mathcal{E}$ will not be built explicitly in the actual implementation, it is helpful for the purpose of developing the theory behind to consider it. The automata for the example sentence are shown in Figure 1.

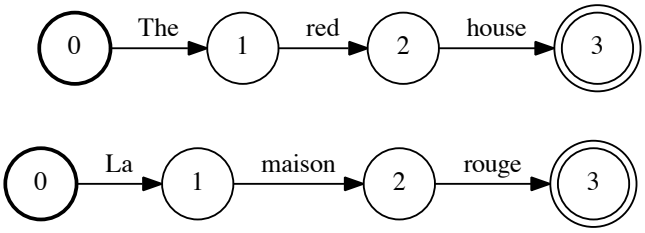


Figure 1: Automata for example parallel sentences.

The next step is to encode the transition probabilities $t(f|e)$ in a weighted transducer $\mathcal{T}$, with input labels in V_f and output labels in V_e. It will we a flower transducer with transition for each possible pair of translated words (f, e). The number of such transitions for a particular word f will naturally depend on the number of english words that appeared in parallel senteces to f in the training corpus. The weights of these transitions, namely $t(f|e)$, are trained externally by using the EM method. The conceptual transducer representing this is shown in Figure 2.

Figure 2: Transition transducer encoding word-to-word translation probabilities.

The next step consists of introducing permutations of positions in the sentece by composing with a suitable transducer. This transducer Π should act on $\mathcal{E}$ and send it to another automaton where each pair of states are joined by l transitions, one for each possible choice of word in english. These transitions have weights corresponding to the alignment probabilities $q(a_i|i, l, m)$, which according to the previous section, can be taken as $e^{-\alpha|i-j\frac{l}{m}|}$ for IBM2. We denote this automaton by $\mathcal{E}_P = \Pi \circ \mathcal{E}$.

For computational purposes, it turned out to be more efficient to directly build $\mathcal{E}_P$ from the target language sentece provided, instead of creating $\mathcal{E}$ and then composing it. For the example sentence, the aforementioned automaton is shown in Figure 3.

Note in the figure above the "off-diagonal" penalty introduced by the choice of function for $q(a_i|i, l, m)$: the cost of aligning *The, red* and *house* with the first, second and third words in french, respectively, is zero in each case. Indeed, as

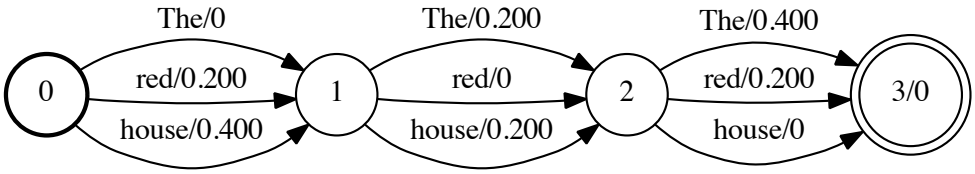


Figure 3: Permutation Automaton for target language sentence.

said before, words are penalized for departing their original position.

With the finite state machines $\mathcal{E}_P$, $\mathcal{T}$ and $\mathcal{F}$ as described above, we can finally obtain alignment probabilities for e and f with the following cascade:

$$\mathcal{A}^* = \mathcal{E}_P \circ \mathcal{T} \circ \mathcal{F} \tag{3.1}$$

By projecting $\mathcal{A}^*$ into the target space a finding the minimal cost path on it, we can effectively compute the most likely alignment of f into e. The `fsm ... library` code required to compute this is the following:

```
fsmcompile -s log -i fwords.syms <AutoFr.txt | fsmcompose ...
   - translator.fsm | fsmcompose - Align.fsm | fsmconvert ...
   -s tropical | fsmbestpath - | fsmproject -2 - ...
   >Predicted.fsm
```

For the example used throughtout this section, the best alignment, along with its transition costs over log semiring is shown in Figure 4.

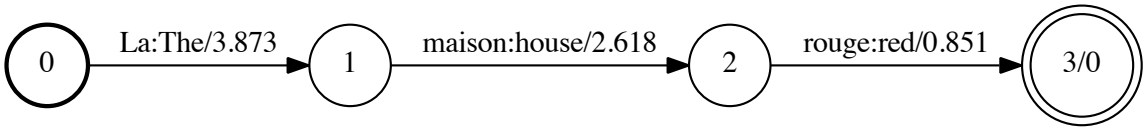


Figure 4: Automaton with most likely alignment.

As we can see in this automaton, the optimal alignment is $a = (1, 3, 2)$, that is, the word *maison*, originally in the second position, is mapped to the third english word *house*. Analogosuly, *rouge* is aligned to *red*.

3.2 Computing the AER

A useful way of measuring the "goodness" of a predicted alignment is the Alignment Error Rate (AER), first proposed by Och and Ney (2003). This metric requires a gold standard manually annotated set of word-word pairs, labelled "Sure" and "Possible". The former are used for measuring Recall, while the latter are used for Precision. If P is the set of possible alignments, S the sure ones and A the predicted alignment, then the AER associated to A has the following form

$$AER(A, P, S) = 1 - \frac{|P \cap A| + |S \cap A|}{|S| + |A|} \tag{3.2}$$

It is possible to implement a FSM that computes the AER of a predicted alignment. For this, the main ingredient is a flower transducer $\mathcal{K}$ with transitions labelled by input and output labels in $I = \{1, \ldots, \max l, m\}$, the set of possible word indices within the sentence. The weights of these transitions corresponds to a $0-1$ loss, where $w(i, j) = 0$ if $i = j$ and $w(i, j) = 1$ otherwise. A conceptual representation of $\mathcal{K}$ is shown in Figure 5.

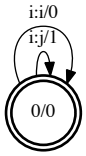


Figure 5: Transducer computing 0-1 loss on pairs of alignments.

In order to use this scoring transducer for a pair of gold and predicted alignments, we need to encode these in terms of the position indices of the alignment.

That is, we will create the automata $\mathcal{P}$ and $\mathcal{G}$ of length $|f| = l$, by labelling the i-th transition with j if f_j is aligned to e_j. transition with the index

The actual `fsm library` code used to compute the AER is shown below.

```
fsmcompose Gold.fsm ScoreTrans.fsm | fsmcompose  - ...
   Predicted.fsm |
fsmbestpath | fsmproject -1 | fsmpush -cf>Scored.fsm
```

Before using this code, we have multiplied the weights of the predicted automaton by 0. This is necessary because `Predicted.fsm` is obtained through the process described in the previous section, and thus in inherently has weighted transitions. The weight in the final state of `Scored.fsm` will correspond to the number of wrong alignments, which can then be used to compute the AER with (3.2).

4 Translation Model Implementation

As the previous sections have shown the IBM models are "alignment" models and they do precisely that, they align which is not necessarily the problem we want to solve. What we need is to find the best possible translation given only a sentence in french.

When evaluating alignments in reality using a transducer doesn't help much in times of speed or space since keeping the data on a table is actually faster to access and uses the same space, the real use of transducers is apparent when we try to obtain real translations in reasonable time.

When translating from French to English, every word in French can be translated to 40-100,000 words depending on which words was it seen with in the training data. For example the word 'est' could be translated to one of 96909

words which is basically all the english training vocabulary. Thus a sentence of 20 words could be translated on average to one of the 1000^{20} possible sentences (being conservative) and this if we only consider english sentences that are the same length as the source one. This gives a huge search space where any search algorithm, even the most efficient would take a really long time to converge. Nevertheless we know that a lot of this sentences are not likely to be accurate translations so we use the weighted transducer machinery to reduce this search space. We explain the procedure:

If we denote the automaton representing the source sentence as $\mathcal{F}$ then we compose $\mathcal{K} = \mathcal{T} \circ \mathcal{F}$. This is by far the most costly operation. To simplify this problem we allow english sentences that are only of the size of the source sentence. We also reduce the search space by letting each french word to only translate to the most likely english words. We achieve this by pruning the translator wtih `fsmprune -c 1`. The change on size in the automaton $\mathcal{K}$ is impressive since the new automaton consists of only 2000 transitions on average. Most if not all of the transitions that were pruned were epsilon transitions from the input language, i.e. transitions that would make the translated sentence larger than the original.

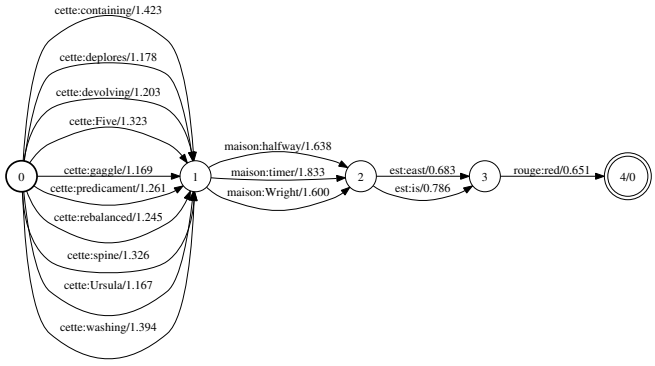


Figure 6: Transducer $\mathcal{K}$ after pruning at level .5 for the sentence "cette maison est rouge". Notice that the real translation is not even a contestant.

We now need to get some information about the likelihood of this sentences in english so we compose the last automaton with a language model. It turns out

that even after pruning composing with a trigram model is computationaly too expensive so we cmopose $\mathcal{K}$ with a Backoff bigram model and obtain the n-best sequnces. In practice $n = 50$ yielded resaonable results.

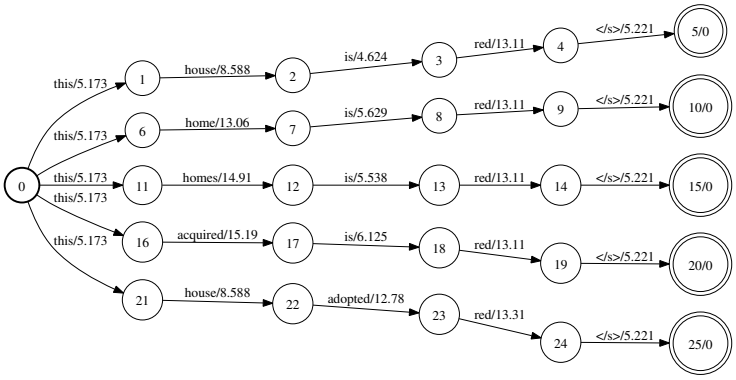
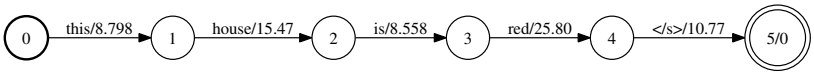


Figure 7: Top 5 hypothesis according to ta bigram model. After composition with a bigram model the score of the real translation appears.

Finally for each of the hypotheses e we create a new automaton that represents the possible permutations (or alignments)of the sentence. This automaton uses the hypothesis of IBM 2 that a word in position i aligns to position j with probability $\propto e^{-\alpha|j-i|}$. Nevertheless since there are english words that appear too often for exampe 'is' or 'of', by letting them take any possible alignment we are going to end up with a sentences that only includes those words, thus we allow only permutations such that e_i is always more likely than any other word in position i. After creating the alignment model we compose this with the trigram model and obtain the best path.



It is worth saying that the use of the language model is crucial in the performance of the algorithm, as you can see in figure 6 the translation model on its own doesn't even consider the real translation as a plausible hypothesis.

5 Results

The Finite State Machines introduced in the two previous sections were implemented using the `fsm library`. It was tested in the Canadian Hanshard dataset, containing transcribed debates of the Canadian House of Commons. This parallel French- English corpus has more than 1,000,000 sentences, although because of computing limitation, we used only $50,000$ of those.

To test the method, we had an additional test set with 447 translated sentences, along with their gold starndard alignments. The AER obtained in this set was 42.1%, while Precision and Recall where 58.93% and 68.72, respectively. With out current implementation, the computation of these measures takes about 30 minutes.

This relatively poor result shows that out implementation, and the underlying IBM model 2 are too simplistic. Common errors found corresponded to french words aligning to too many english words, instead ob being align to the *null* word.

Unfortunately we weren't able to present performance results on the translation (word error rate or BLEUE score) because of time constrains. By running some examples though we saw that it was really bad for sentences of more than 5 words. For really simple sentences though it performed rather well even when the sentences weren't reallly related to the training set. Nevertheless the intention of our project wasn't to beat the state of the art algorithms for translation but to give a reasonable introduction to alignment models and run the translation search on a transducer which eventhough is slow I can only imagine that it would be much worse without the help of the `fsm-library`.

6 Conclusion

We have shown an implementation of alignment and translation models with Weighted Finite State Machines. We offered a motivation for and briefly reviewed the statistical theory and assumptions behind the IBM model 2.

The model used is extremely simplistic and is seldom used in practice anymore. Better, more involved models have been developed progressively over the last years. For example, phrase-based methods attempt to improve on one of the main disavantages on word-based translation by keeping phrases in nonseparable "phrase" nuclei (usually 1 to 3 words), and then permutting alignments of those. Syntax-based models [6], in addition, incorporate POS tags and Syntax Trees, in order to better exploit characteristics of the target and source languages when translating (e.g. verb placement in a sentence).

Despite its defficiencies and simplicity, the IBM model 2 is a valuable tool for understanding the main ideas behind Machine Translation. In addition, it allows for a relatively simple implementation as a cascade of Finite State Machines.

We were really happy with the results of the translator, we are aware that the word allignment error is rather big but considering this was done using a really simplistic model we believe the results are good. The major challenge we faced during this work was that of reducing the search space and taking into account the alignments while translating. After finishing this work we believe there are probably better ways of encoding a translator than a transducer. The biggest problem of a transducer approach on our view is that as opposed to speech recognition there is no 'time' component since the permutations that the translation model induce kill any idea of sequential transformation and so methods like beam search are doomed to fail.

References

[1] P. Brown, S. Della Pietra, V. Della Pietra, and R. Mercer, *The mathematics of statistical machine translation*, Computational Linguistics, 19 (1993), pp. 263–311.

[2] D. Jurafsky and J. H. Martin, *Speech and Language Processing - An introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, Pearson Prentice Hall, second ed., 2008.

[3] K. Knight and Y. Al-Onaizan, *Translation with finite-state devices*, in Proceedings of the Third Conference of the AMTA, Springer-Verlag, 1998, pp. 421–437.

[4] P. Koehn, F. J. Och, and D. Marcu, *Statistical phrase-based translation*, in Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1, NAACL '03, Stroudsburg, PA, USA, 2003, Association for Computational Linguistics, pp. 48–54.

[5] S. Kumar and W. Byrne, *A weighted finite state transducer implementation of the alignment template model for statistical machine translation*, in Proceedings of HLT-NAACL, Association for Computational Linguistics, 2003, pp. 63–70.

[6] K. Yamada and K. Knight, *A syntax-based statistical translation model*, in Proceedings of the Conference of the ACL, 2001, pp. 523–530.

Code

```
import csv, random, re, os,sys, subprocess
```

```
from math import exp

alpha=.2
path1 = "data/test/"
path2 = "testAutomata/"
en_file = open(path1+"test.e","rb").readlines()
fr_file = open(path1+"test.f","rb").readlines()

align_file = open("data/answers/test.wa.nullalign","rb")
line = align_file.readline()
line = line.split()
totalError = 0
validErrors = 0

def P(j,i,m,n):
    return alpha*abs(j-i*n/m);

for sentence_num in range(1,len(en_file)):
        en_text = re.sub(r'(<s snum=\d+> )', ...
            r'',en_file[sentence_num-1])
        en_text = re.sub(r'(</s>)', r'',en_text)
        fr_text = re.sub(r'(<s snum=\d+> )', ...
            r'',fr_file[sentence_num-1])
        fr_text = re.sub(r'(</s>)', r'',fr_text)
        en_words = en_text.split();
        fr_words = fr_text.split();
        en_m = len(en_words)
        fr_m = len(fr_words)


        #First, create alignments
```

```
        en_align_filename = 'AlignedEn'+str(sentence_num)+'.txt'
        en_align_automaton = open(path2+en_align_filename,'w')
        for i in range(0,fr_m):
                s = str(i) + "\t" + str(i+1);
                for j,word in enumerate(en_words):
                        weight = P(j,i,fr_m,en_m);
                        en_align_automaton.write(s + "\t" + ...
                            word + "\t" + str(weight) + "\n");
        en_align_automaton.write(str(fr_m));
        en_align_automaton.close()
        os.system("fsmcompile -s log -i enwords.syms ...
           <"+path2+en_align_filename\
        +"> testAutomata/Align"+str(sentence_num)+".fsm")


        # English - Position Mapper transducer
        en_posMap_filename = 'PosMap'+str(sentence_num)+'.txt'
        en_posMap = open(path2+en_posMap_filename,'w')
        for j,word in enumerate(en_words):
                en_posMap.write(str(0) + "\t" + str(0) + ...
                    "\t" + word + "\t"+str(j+1) + "\n");
        en_posMap.write(str(0));
        en_posMap.close()
    os.system("fsmcompile -i enwords.syms -t ...
       <"+path2+en_posMap_filename\
        +"> testAutomata/PosMap"+str(sentence_num)+".fsm")



        fr_filename = 'AutoFr'+str(sentence_num)+'.txt'
        fr_automaton = open(path2+fr_filename,'w')
```

```
k=0
for i in range(0,fr_m):
        fr_automaton.write(str(k)+'\t'+str(k+1)+'\t'
        +fr_words[i]+'\n')
        k = k+1
fr_automaton.write(str(k))
fr_automaton.close()

instruction = "fsmcompile -s log -i fwords.syms ...
   <testAutomata/AutoFr"\
+ str(sentence_num)+".txt | fsmcompose - ...
   translator.fsm | fsmcompose - testAutomata/Align"\
+str(sentence_num)+".fsm | fsmconvert -s tropical | ...
   fsmbestpath - | fsmproject -2 - >"\
+"testAutomata/Predicted"+str(sentence_num)+".fsm"
os.system(instruction)

# Print predicted in terms of position
os.system("fsmcompose ...
   testAutomata/Predicted"+str(sentence_num)+".fsm ...
   testAutomata/PosMap"\
+str(sentence_num)+".fsm | fsmproject -2 | fsmarith ...
   -m 0 - >testAutomata/PredictedPos"\
+str(sentence_num)+".fsm")



os.system("rm "+path2+en_align_filename)
os.system("rm "+path2+fr_filename)


## Read Gold Alignments and Create Transducer with Them
```

```
gold_aut = ...
    open(path2+'Gold'+str(sentence_num)+'.txt','w')
gold_pos_aut = ...
    open(path2+'GoldPos'+str(sentence_num)+'.txt','w')
NumSures = 0
while int(line[0]) == sentence_num:
        if not int(line[2])==0:  # Dont know how to ...
            deal with nulls in french. Pretend they ...
            are not there.
                if int(line[1])==0:    #Fr word ...
                    mapped into NULL
                        gold_aut.write(str(int(line[2])-1)\
                        +'\t'+line[2]+'\t'+"NULL"+'\n')
                        gold_pos_aut.write(str(int(line[2])-1)+'\t'+\
                        line[2]+'\t'+str(en_m+1)  +'\n')
                else:
                        gold_aut.write(str(int(line[2])-1)\
                        +'\t'+line[2]+'\t'+\
                        en_words[int(line[1])-1]+'\n')
                        gold_pos_aut.write(str(int(line[2])-1)\
                        +'\t'+line[2]+'\t'+str(int(line[1]))+'\n')
                if line[3] == 'S':
     line = line.split()
gold_aut.write(str(fr_m))
gold_pos_aut.write(str(fr_m))
gold_aut.close()
gold_pos_aut.close()

gold_pos_aut = open(path2+'GoldPos'\
+str(sentence_num)+'.txt','r+')
```

```
if not int(gold_pos_aut.readline()[0])==0:   # File ...
    doesnt start with the 0 node
                gold_pos_aut.close()
                f= open(path2+'GoldPos' +
                str(sentence_num)+'.txt' , 'r')
                lines = f.readlines()
                lines.sort()
                f.close()
                writer = open(path2+
                'GoldPos'+str(sentence_num)+'.txt','w')
                writer.writelines(lines)
                writer.close()
os.system("fsmcompile  <testAutomata/GoldPos"\
+ str(sentence_num)+".txt>testAutomata/GoldPos"\
+str(sentence_num)+".fsm" )

## Position Flower to score
transducer = ...
    open(path2+'ScoreKroneckerPos'+str(sentence_num)+'.txt','w')
for i in range(1,en_m+2):
        for j in range(1,en_m+2):
                if i==j:
                        transducer.write("0\t0\t"+ ...
                            str(i) + "\t" + str(j) + ...
                            "\t" + str(-1) + "\n");
                else:
                        transducer.write("0\t0\t"+ ...
                            str(i) + "\t" + str(j) + ...
                            "\t" + str(0) + "\n");
transducer.write("0\t0")
transducer.close()
```

```
instruction = "fsmcompile -t <"+path2+ ...
    "ScoreKroneckerPos"+
str(sentence_num) + ".txt>"+path2 + ...
    "ScoreKroneckerPos" + str(sentence_num)+".fsm"
os.system(instruction)



MultCost = -1.0/(NumSures + fr_m)      # Para ...
    calcular AER se necesita (|P int A| + |P int ...
    S|) / (|S|+|A|)
instruction = "fsmcompose ...
    testAutomata/GoldPos"+str(sentence_num)\
+".fsm ...
    testAutomata/ScoreKroneckerPos"+str(sentence_num)+".fsm ...
    | fsmcompose  - testAutomata/PredictedPos"\
+str(sentence_num)+".fsm | fsmbestpath | fsmproject ...
    -1 | fsmarith -m "+str(MultCost)+ " | fsmpush ...
    -cf>testAutomata/Scored"+str(sentence_num)+".fsm"
os.system(instruction)



a =subprocess.Popen("fsmprint
testAutomata/Scored"+str(sentence_num)+".fsm" , ...
    shell=True,stdout=subprocess.PIPE)
(out,err) = a.communicate()
nums = re.findall(r"0+\.?\d+", out)
if not not nums:
        error = 1-float(nums[0])
        totalError = totalError + error
        validErrors = validErrors + 1
        print totalError/validErrors
```

```
print totalError/validErrors
```