QUELQUES OBSERVATIONS CONCERNANT LE

TP1 - IFT6285

REPORT IS IMPORTANT !!!

- The presence of code is almost useless (unless clarification needs to be made).
- The abstract should describe what has been done (not just paraphrasing the subject)
- Delivering results is something, analyzing / explaining why it is so is better
- No need to copy the subject or explain KN (unless you use the description in your analysis)

GOOD

1 Introduction

Dans cette étude, différents modèles de langue probabilistes ont été entraînés et évalués à l'aide de la boîte à outils KenLM [1] que nous avons interrogée directement à partir du Shell et également à partir de l'API de Python. Des modèles de type n-gramme (i.e. 2-, 3- et 4-gramme) avec lissage Kneser-Ney ont été comparés. Des corpus en français, anglais et allemand ont été extraits à partir des tâches partagées WMT 2014 et WMT 2017 [2]. Ces trois corpus ont été découpés en ensembles d'entraînement, de validation et de test. Puis, la qualité des modèles a été mesurée par le biais de la perplexité et du pourcentage de mots prédits correctement, pour les rangs 1 et 5.

GOOD

Tableau 1 : Analyse statistique de la totalité de chaque corpus

	Exemple de ligne de commande	train.fr
Nombre total de mots	cat train.en wc -w	1,247,666,615
Nombre total de lignes	cat train.en wc -l	40,811,694
Nombre de mots uniques	awk 'BEGIN{RS=" "} 1' train.de > train_words.de awk '! a[\$0]++ train_words.de > train_uniq.de wc -l train_uniq.de	2,545,021
Nombre de mots différents vus au moins 100 fois	awk '{for(i=1;i<=NF;i++) a[\$i]++} END { for(k in a) if(a[k] >=100) b++} END {print b}' train_words.en	103,647
Nombre de mots différents en test non vus en train	awk '! a[\$0]++' words_test.en > uniq_test.en awk 'FNR==NR {a[\$1]; next} \$1 not in a' clean/test_unique.en train_unique.en > unseen.en	10,865

BAD

{

<pre>'/home/sole/wiki/nlp/data/train.de': {</pre>	, , , ,
	·_·,
	· · ,
	°@`,
	'auf',
	'das',
	'da',
	'den',
	'der',
	'die',
	'es',
	'für',
	'ich',
	'im',
	'in',
	'ist',
	'nicht',
	'sie',
	'und',
	'von',
	'wir',
	'zu'},

OK, BUT BUT UP TO A POINT

- On the difficulty of installing KenLM on Windows.
 - I can understand that, although you were many to manage to make it work
- On the impossibility to count words in a (large) corpus
 - Highly suspicious (do not memorize the corpus)
- I was not able to find the option to count words in wc

• well...

SOME COOL THINGS

- Time / size of the models as a function of the ngram order
- Aptitude of the model to predict the next word
 - on a small subset of sentences
 - on sampled contexts
 - Comparing the top-1 prédiction made by kenLM to the one of the iPhone !!!
- Analyzing the results further
 - Variation on POS,
 - using NER to regroup words,
 - ppl w/o oov, etc.



DIRTY HANDS

Elements of a solution

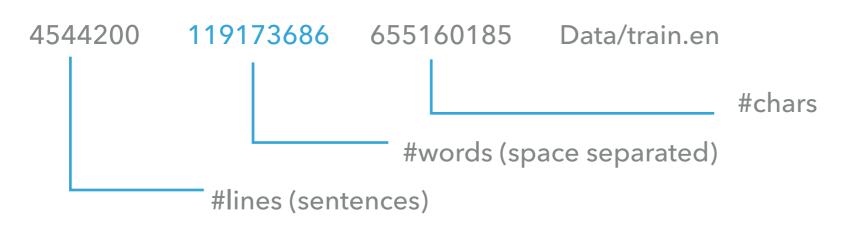
% HEAD -N 3 DATA/TRAIN.EN

Resumption of the session

I declare resumed the session of the European Parliament adjourned on Friday 17 December 1999, and I would like once again to wish you a happy new year in the hope that you enjoyed a pleasant festive period.

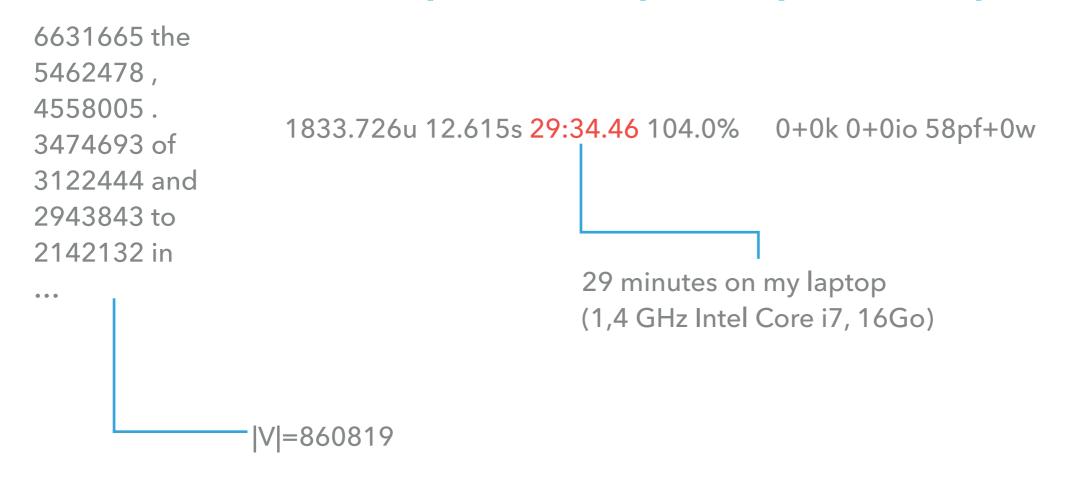
Although , as you will have seen , the dreaded ' millennium bug ' failed to materialise , still the people in a number of countries suffered a series of natural disasters that truly were dreadful .

% WC DATA/TRAIN.EN



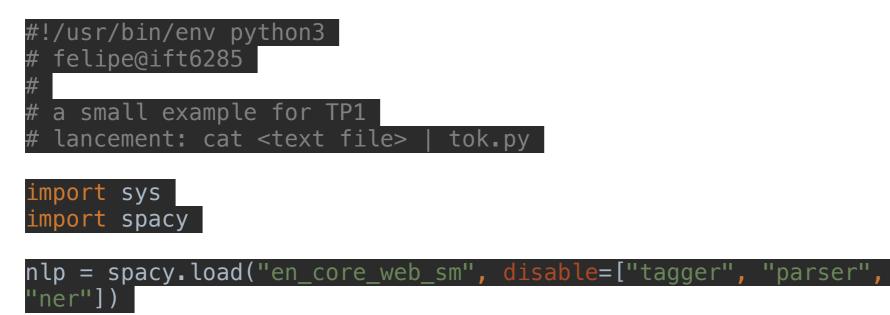
=> 119M occurrences

% CAT DATA/TRAIN.EN | TR ' ' '\N' | SORT | UNIQ -C | SORT -K1,1NR



Not fast, because of the **sort** operation, but easily something you can run while you are doing something else

WELL, USE SPACY / NLTK / WHATEVER YOU LIKE



for line in sys.stdin:
 words = [tok.text for tok in nlp(line.strip())]
 print("\n".join(words))

AND DIVIDE AND CONQUER

CAT DATA/TRAIN.EN | TOK.PY | SORT | UNIQ -C | SORT -K1,1NR

YOUR PYTHON PROGRAMS COUD SIMPLY BE COMPONENTS OF A PIPELINE

TRAIN-KENLM

```
#!/bin/csh -f
set kenImpath = ~/myPackages/kenIm/build
set modelpath = ../Data_nobackup/models/kenIm
```

```
if ($#argv != 2) then
  echo "usage: $0 <corpus> <order>"
  exit
endif
```

```
set corpus = $1
set order = $2
```

```
# 0) decide which file to generate
set model = `basename $corpus`-${order}g
set arpa = ${modelpath}/${model}.arpa
set binary = ${modelpath}/${model}.bin
```

1) computing an arpa model
\$kenImpath/bin/Implz -o \$order -S 80% -T /tmp < \$corpus > \$arpa

```
# 2) converting arpa into binary$kenImpath/bin/build_binary $arpa $binary
```

```
# 3) check
ls -l $arpa $binary
```

% TIME TRAIN-KENLM DATA/TRAIN.EN 3

-rw-r---- 1 felipe staff 1,5G 29 oct 15:19 train.en-3g.arpa -rw-r---- 1 felipe staff 972M 29 oct 15:19 train.en-3g.bin

Quite a lot of space (you actually only need the binary format)

114.288u 10.647s 2:11.41 95.0% 0+0k 0+0io 160241pf+0w

Only 2 min. of my laptop for a **3gram** ! (how fast considering counting words took so much time)

% TIME TRAIN-KENLM DATA/TRAIN.EN 2

-rw-r---- 1 felipe staff 293M 29 oct 15:24 train.en-2g.arpa -rw-r---- 1 felipe staff 222M 29 oct 15:24 train.en-2g.bin

48.748u 3.217s **0:52.33** 99.2% 0+0k 0+0io 22pf+0w

- Less than a min. for a bigram model ...

HEAD -N 2 DATA/TEST.EN | QUERY MODELS/TRAIN.EN-2G.BIN

Gutach=838082 1 -8.563082 :=585 1 -2.6727304Increased=3366 2 -4.638858 safety=434 2 -2.5548968 for=94 2 -1.869487 pedestrians=27530 2 -4.26594 </s>=2 1 -4.1019487 Total: -28.666943 OOV: 0

They=1042 2 -2.1825824 are=327 2 -0.612774 not=233 2 -1.2988738 even=866 2 -2.1322188 100=5921 2 -3.7163234 metres=5923 2 -1.3846334 apart=2758 2 -3.3101661 :=585 2 -3.1149714 On=1549 2 -2.96381 Tuesday=583 2 -2.6911228 ,=18 2 -0.5171114 the=5 2 -1.142501 new=29 2 -2.2626345 B=7616 2 -4.1564455 33=6087 2 -4.359213 pedestrian=29022 1 -5.421015 lights=10349 2 -3.6579065 in=31 2 -1.3908285Dorfparkplatz=0 1 -8.700969 in=31 1 -2.199971 Gutach=838082 2 -6.850334 became=6042 1 -3.6498296 operational=4463 2 -2.6847038 -=649 2 -2.4236326 within=781 2 -3.2732668 view=967 2 -3.9559016 of=4 2 -0.58965415 the=5 2 -0.5317595 existing=1472 2 -3.108656 Town=24289 1 -5.166059 Hall=21402 2 -0.9881305 traffic=1120 1 -4.900457lights=10349 2 -1.5967978 .=38 2 -0.9308659 </s>=2 2 -0.03715668 Total: -97.903275 OOV: 1

Perplexity including OOVs:1031.755038487236Perplexity excluding OOVs:749.6520221763684OOVs:1Tokens:42RSSMax:226201600 kBuser:0.003488 sys:0.130144CPU:0.133679 real:0.127912

CAT DATA/TEST.EN | TP1.PY MODELS/TRAIN.EN-2G.BIN

#!/usr/bin/env python3
tp1.py

import kenlm import argparse import sys

modelName = get_args()
model = kenlm.Model(modelName)

```
for i,line in enumerate(sys.stdin,1):
```

sentence = line.rstrip()
ppx = model.perplexity(sentence)

to be written

I reckon you use this to parse the command line in your python progs

Do whatever you want with input sentence

MEASURING THE PREDICTION RATE OF WORDS

def **do_sentence**(sentence, vocab, model):

```
state_in = kenlm.State() # define a few states
state_out = kenlm.State()
```

ranks = [] # le rang de chaque mot de la phrase model.BeginSentenceWrite(state_in) # model.NullContextWrite(state) for w in sentence.split():

res = [] # 1) argmax over vocab
for v in vocab:
 s = model.BaseScore(state_in, v, state_out)
 res.append((v, s))

model.BaseScore(state_in, w, state_in) # reseter state_in

```
res.sort(key=lambda x: x[1], reverse=True)
rank = linear_search(lambda x: x[0] == w, res)
ranks.append(rank)
```

2) sort the nbest list# get the rank of w# store it for stats

```
if verbosity:
    print(f"PRED\t{w}\t{rank}\t", res[:5])
```

do stats

• • •

return ranks

vocab is a set of words that compete

the 1000 most frequent words

HEAD -N 10 DATA/TEST.EN | TP1.PY MODELS/TRAIN.EN-2G.BIN --VOC=VOC.1K --VERBOSITY=2

PRED Gutach 1000 [('The', -0.8539741039276123), ('I', -1.2894823551177979), ('We', -1.3505849838256836), ('In', -1.384865403175354), ('This', -1.4174968004226685)] PRED : 15 [('.', -0.5119906067848206), (',', -1.8282580375671387), ('and', -2.1233713626861572), ('(', -2.304332733154297), ('is', -2.3374271392822266)] PRED Increased 832 [('the', -1.1898130178451538), ('"', -1.4203648567199707), ('The', -1.5330532789230347), ('a', -1.7557408809661865), ('/', -1.8039711713790894)]

• • •

top<=1: 17.39 % nstop: 8.21 %
top<=2: 24.15 % nstop: 9.18 %
top<=3: 29.47 % nstop: 11.11 %
top<=4: 31.40 % nstop: 12.56 %
top<=5: 33.82 % nstop: 14.49 %
top<=6: 35.27 % nstop: 14.98 %
top<=7: 35.27 % nstop: 14.98 %
top<=8: 37.20 % nstop: 16.43 %
top<=9: 39.61 % nstop: 18.36 %</pre>

% of tokens (of test) predicted first

With a vocabulary as small as 1k words, this code is very fast

% of tokens predicted in the 10-first positions

A NOTEBOOK IS COOL

But **decomposing** a program into several components (possibly using unix commands) that deal with specific parts each (ex: tokenization / normalisation / cutoff) allows for:

- Reusability
- Batch processing
 - ⁻ running a script that does many things while you do ... something else
 - possibly on several computers (if you are lucky)
 - without modify one char of a code
- Deployment without tear on platforms like Google Colab
- Less coding

CAT <IN> | TOK.PY | NORMALIZE.PY | CUTOFF.PY 10 | TP1.PY

BUT WHAT MATTERS IN THE END ... IS THE REPORT

10: BE CURIOUS 20: GOTO SLIDE 2