Accelerated Natural Language Processing 2019
Assignment 1

s1887322, s1933572

Task 1

We created a simple preprocess_line(**str**) function, that takes as argument a string and modifies it by removing all the characters that are not integers nor non-English characters, substituting all the digits with 0, lowering the letters and adding '##' at the beginning and '#' at the end of each line. In the next tasks, we will be including the hash in our model vocabulary, in order to generate separated lines.

```
def preprocess_line(str):
        new_str = re.sub('[^a-zA-Z0-9. ]', '', str)
        new_str = re.sub('[0-9]', "0", new_str)
        new_str = new_str.lower()
        new_str = '##' + new_str + '#'
        #to eliminate double spaces
        new_str = ' '.join(new_str.split())
        return new_str
```

Task 2

By looking at the file, we hypothesized that the model behind the computed probabilities is smoothed with some value of α. To better understand how the probabilities were estimated, we first looked at those trigrams that are unlikely to occur in English, such as *aaa* or *zzz*. For these sequences, the probability is equal to 0.033, meaning either that they actually occurred in the text, or that some kind of smoothing method was applied, which is more likely to be the case. However, it is not possible to tell what is the value of α being used without looking at the training file. More in details, let the vocabulary size be V=30 (because of the 26 English characters, and the 4 instances '#', '.' , ' ', '0'), and let us assume that *zz* and *zzz* never occurred in the training text. Then, by applying Laplace smoothing, we obtain the equation:

$$\mathrm{P}(zzz) = \frac{\mathrm{C}\,(\mathrm{zzz}) + \alpha}{\mathrm{C}\,(\mathrm{zz}) + \alpha V} = \frac{0 + \alpha}{0 + \alpha 30} = 0.033$$

that is true for each α different than 0. Therefore, we can only conclude that $0 < \alpha \leq 1$

Task 3

We decided to build a mixture language model that combines a unigram, a bigram and a trigram model with *lambda* values based on the perplexity over a validation set. On the one hand, a model based on maximum likelihood estimation without smoothing would behave poorly on a test file, as such a model assigns probability 0 to trigrams that never occured in the training phase. On the other hand, an add-alpha smoothing model would assign the same probability to the unseen trigrams, at the expense of sequences of characters that are more likely to occur. Therefore, after training multiple add-alpha smoothing models, we preferred to turn to interpolation, that assigns different weights to different ngram models.

The main function is language_model(**str**, **str**, **int**, **int**, **int**, **int**), which takes as input the name of the training file, the language of the model and the three λ values, and returns a new file with the trigram probabilities based on their occurrence in the input file. The optimal λ_1, λ_2, λ_3 were searched for as the values that minimizes the perplexity of the model over a validation set, that we obtained by splitting the training file, under the constraint that $\lambda_1+\lambda_2+\lambda_3 = 1$. All the possible one-character sequences (just the vocabulary), two-character sequences and three-character sequences were generated and stored as keys in three separated dictionaries using the count_ngram(**dict**(), **str**, **int**) function, with the number of their occurrences (absolute frequency) being the values. The conditional probability of a character to occur given the two previous character is then estimated based on these counts, as follows:

$$\begin{aligned} \mathrm{P}(ch_i|ch_{i-2}ch_{i-1}) =& \lambda_1 \mathrm{P}_1(ch_i) + \lambda_2 P_2(ch_i|ch_{i-1}) + \lambda_3 P_3(ch_i|ch_{i-1}, ch_{i-2}) \\ =& \lambda_1 \frac{\mathrm{C}\,(ch_i) + 1}{\sum_{j=1}^{N} ch_j + V} + \lambda_2 \frac{\mathrm{C}\,(ch_{i-1}ch_i) + 1}{\mathrm{C}\,(ch_{i-1}) + V} + \lambda_3 \frac{\mathrm{C}\,(ch_{i-2}ch_{i-1}ch_i) + 1}{\mathrm{C}\,(ch_{i-2}\ ch_{i-1}) + V} \end{aligned}$$

for $2 < i \leq$ N, with N being the total number of characters in the text, and the first two characters being '##'. The probabilities P_1, P_2, P_3 were smoothed with $\alpha = 1$ for the sake of simplicity.

To give an example of the estimated probabilities, we will now be focusing on all the trigrams with history *ng*, as shown in the table below. As expected, the blank space and the dot are very likely to be picked after the sequence *ng*, probably because of the suffix *ing* being very common in English, for instance in the gerund form. Also, in general, the vowels are more likely to follow than the

consonants, which is consistent with what we know about the possible syllables in English. On the other hand, we were surprised to find out that the character *d* has a slightly higher probability to occur than other consonants. A quick search showed that is due to the occurrence of the word *Kingdom*, in *United Kingdom*.

Probabilities of trigrams with *ng* history

ng 6.761514e-01	ng# 4.023626e-03	ng. 2.275839e-02
ng0 1.508526e-03	nga 1.396215e-02	ngb 2.072131e-03
ngc 3.639757e-03	ngd 7.228960e-03	nge 9.143360e-02
ngf 3.916209e-03	ngg 2.992604e-03	ngh 1.082150e-02
ngi 2.257546e-02	ngj 1.181086e-03	ngk 1.455548e-03
ngl 6.416349e-03	ngm 3.474442e-03	ngn 9.136059e-03
ngo 1.816082e-02	ngp 3.194235e-03	ngq 1.151725e-03
ngr 2.941367e-02	ngs 2.327691e-02	ngt 1.954777e-02
ngu 1.086928e-02	ngv 1.909369e-03	ngw 2.331274e-03
ngx 1.219383e-03	ngy 3.094996e-03	ngz 1.082790e-03

Task 4

In order to generate a sequence of characters given a language model, we built the function generate_from_LM , which takes as argument the name of the model probabilities file and returns a 300-character string. While reading through the file, the model probabilities are stored into a dictionary, in order to be easily retrieved. A "head" string variable is initialized as '##', to indicate the start of a new line. The process of generating the characters is done by a for loop. At each iteration, a list of all possible trigrams starting with the last two characters of the sequence generated until then is stored; then, a trigram is randomly picked from the list based on the probability distribution; finally, the last character of the picked trigram is added at the end of the sequence, and the head is updated with the new last two characters, unless the picked item is a '#', in which case the head is updated with '##' to indicate the start of a new line. In this implementation, all the items that belong to the vocabulary are counted in building the sequence, therefore the '#' counts as one character as well.

The two pieces of text generated by the given model and our model look quite different, as shown below. The first one is made up of multiple short lines, while the second is made up longer lines. Also, while it is possible to recognize many meaningful English words in the first output, the second is rich in long sequences of characters that do not really have a meaning. We hypothesized that this is due to the large occurrence of long sentences in the training text.

Given Model

##not the hat.#
##let the en.#
##do gusten his.#
##tht.#
##whaven you hime i knocke tak anow.#
##go its that cand th thi jumbloo think birrou put.#
##cagot i topen you say.#
##sh come a whally the whats onna.#
##hem.#
##cage.#
##her.#
##ok anymor.#
##yean this.#
##he his.#
##ly.#
##ye.#
##diddy.#

Our Model

##whishs noween con funothel on tharnionuhquelits resssubp-wass #
##ing ast aaebh efffommun-frod tn#
##theur opmunne weloy#
##thelitted isis par pcoumfir coodand yon ecer sumnhe hatiche 0y#
##whisel.#
##was fafe concraep whistaguendits.#
##bqf the partion red lcgjult#
##wuzjudely dillireptterand al b

Task 5

When testing our LM, we obtained different perplexity values based on the text file that was used to train the model. In particular, the lowest perplexity is reached when the training file is in English, while the other languages cause the model to have much higher perplexities, as shown in the table below. Therefore, we could guess that the test file was in English, as it is indeed.

Training File Language	Perplexity
English	8.79
German	18.33
Spanish	18.69

However, this metric alone cannot be considered a sufficient proxy for guessing the language of a test file, as other factors may contribute to the final perplexity value. For instance, if we test our model on a text document made up of one single trigram that has a high probability to occur (say *ing*, for instance) repeated multiple times, the perplexity will result in a small value, because the total probability estimated over the whole sequence will be high (just because the trigram probabilities are multiplied together, according to the perplexity

formula). However, no one would consider such a text as written in English. To give another example, we would obtain a low perplexity by testing the model on a text such the ones generated in Task 4, which however would be hardly considered English forms. This is because the model is not based on sequences of actual words, but only sequences of characters that are likely to occur together according to the trigram model probabilities.

Task 6

In task 5, we have mentioned that our trigram language model is not very reliable when it comes to guess the language of a text because it is built over characters, rather than words. What if we try to consider larger ngrams, that may be more likely to capture the structure of a actual words? To answer this question, we created a four-gram model and a five-gram model, and for each of them we estimated the perplexity on the test provided, the running time, and the size of the model probabilities file. To keep this part simpler to understand, we used a add-one smoothing LM, with the probabilities being estimated as follows:

$$\mathrm{P}(ch_i|ch_{i-2}ch_{i-1}) = \frac{\mathrm{C}\left(ch_{i-2}ch_{i-1}ch_i\right) + 1}{\mathrm{C}\left(ch_{i-2}\ ch_{i-1}\right) + 30}$$

The code shall we found in a separated file (task6.py) The results are summarized in the table below:

	2-gram	3-gram	4-gram	5-gram
Perplexity	11.29	8.87	8.05	9.79
Training time	0.10s	0.16s	2.17s	68.47s
Size of model file	16KB	462KB	14.7MB	419.4MB
Lines of model file	900	26100	757K	22M

As predictable, the bigram model shows the highest perplexity on the test file, because a new character is randomly generated based on the conditional probability on only the previous instance. However, against our expectations, the fivegram model is not the best performing model, having a higher perplexity than the trigram and the fourgram model. We believe that the the five-gram model may be improved by adjusting the value of α, as well as by training the model on a bigger training file. Furthermore, as n increases, the training time and the model file size grow exponentially, causing a worse efficiency. For this reason, we shall not always choose a large n for our ngram model, although it may perform a little better. We also acknowledge that the combination of these n-gram models through interpolation would lead to even better results, although its implementation does does not appear straightforward.