# ANLP Assignment 1 2018

s1879797, s1873947

November 9, 2018

## 1 Question 1

`preprocess_line` **method:**

```python
# Preprocess every line of the files
def preprocess_line(line):
    char_removed = re.sub('[^a-zA-Z0-9. ]', "", line)
    number_formated = re.sub('[0-9]', "0", char_removed)
    string_lower = number_formated.lower()
    added_hashes = "##" + string_lower + "#"

    return added_hashes
```

We added "##" at the beginning and "#" at the end of the lines in order to delimit every line and make possible to count every trigram from the original lines.

## 2 Question 2

After looking at the language model provided we think that the probability distribution uses Katz back-off model for trigram probability calculation.

For example, if we take the trigram " cq", there are no words in English language which start with "cq" and the probability of trigram " cq" should be zero. As per Katz back-off model, when the highest order n-gram (trigram in this case) is not present, the algorithm goes back to calculate the probability of the bigram " c" in the corpus. So the probability of " cq" is taken as the probability of the bigram " c". By initial validation we have observed from the model file that " cq" and " c " have the same probability. As the probabilities of bigrams are not provided in the model file, we have come to an assumption that since " cq" and "c " are unlikely words in English language, they are given the probability of " c" which is the reason for both trigrams " cq" and " c " to have the same probability. This has also been observed with other trigrams like " cp" - has also the probability of " c ", "   " (3 spaces), " 0[a-z]" - have the probability of " 0" - which are less likely to appear in the language.

Formula for calculation of Probability with Katz back-off model:

$$P_{\text{backoff}}(w|h) = \begin{cases} P^*(w|h) & c(hw) > 0 \\ \alpha(h) P_{\text{backoff}}(w|h^-) & \text{otherwise} \end{cases}$$

Figure 1: Probability using Katz back-off model[1]

## 3 Question 3

The model that we built in order to collect counts and estimate probabilities for trigrams is Maximum Likelihood Estimation (MLE) with Add-One Smoothing. We wanted to use Add-α Smoothing, but for this point it is not necessary to calculate the perplexity in order to determinate the best α so we will leave this analysis of perplexity for the following exercises. We have chosen to implement this model because the training data is limited and the test set may have some unknown trigrams which are not encountered in training data and MLE alone will just give probabilities of 0 for those trigrams.

The pseudocode for the generation algorithm:

```
Step 1: Calculate the count of all trigrams and all bigrams from the training corpus generated after preprocessing of the data.

Step 2: Generate all the possible trigrams with this vocabulary: {[a-z], "0", ".", " ", "#"]}. These are all stored in the dictionary with probability value equal to 0.

Step 3: Calculate the probability of the trigrams by using the formula:

    $probability(trigram) = \frac{count(trigram) + a}{count(bigram) + a * v}$

and update the values in the dictionary created in Step 2.

v - vocabulary: {[a-z], "0", ".", " ", "#"]} (30 elements)
α - the discount factor which reduces the probability of high frequency trigram by a factor and distributes it to the low/no frequency trigrams (in our case α=1)

Step 4: The output is written to file "model-all-prob.en".
```

[1] https://cxwangyi.wordpress.com/2010/07/28/backoff-in-n-gram-language-models/

As per the language model, we have calculated the probabilities for all the trigrams. The trigrams like "ng0" which are not present in the training data, are expected to have a value of

$$\frac{a}{count(ng) + a * v}$$

This result is as expected and it is shown in the ng<> probability values below:

```
ng  7.874e-01
ng# 2.516e-03
ng. 2.642e-02
ng0 1.258e-03
nga 3.774e-03
ngb 1.258e-03
ngc 1.258e-03
ngd 5.031e-03
nge 8.553e-02
ngf 2.516e-03
ngg 1.258e-03
ngh 1.258e-03
ngi 2.516e-03
ngj 1.258e-03
ngk 1.258e-03
ngl 3.774e-03
ngm 1.258e-03
ngn 2.516e-03
ngo 7.547e-03
ngp 1.258e-03
ngq 1.258e-03
ngr 1.258e-02
ngs 2.138e-02
ngt 1.384e-02
ngu 3.774e-03
ngv 1.258e-03
ngw 1.258e-03
ngx 1.258e-03
ngy 1.258e-03
ngz 1.258e-03
```

## 4 Question 4

The pseudocode for the generation algorithm:

```
Step 1: Calculate the count of all trigrams and all bigrams from the training corpus generated after preprocessing of the data.

Step 2: Generate all the possible trigrams with this vocabulary: {[a-z], "0", ".", " ", "#"]}. These are all stored in the dictionary with probability value equal to 0.

Step 3: Calculate probabilities for every trigram using MLE with Add-One Smoothing and update the values in the dictionary created in Step2.
Step 4: Use generate_string function to generate the strings

Step 5: Add "##" for the start of the string

Step 6: For every two last characters from the string (bigram) we take the trigrams which start with that two characters and alongside with their probabilities we make a random weighted choice.

Step 7: From the selected trigram we take the last character and we added to the string.

Step 8: If in the middle of the generation process there is no trigram which will start with the last two characters from the string we will add a space character. We decided between adding a space character or a random character from the model's alphabet and we thought a space is better because in English language the end of a sentence is followed by and empty space and then another sentence starts.

Step 9: The end of the generated string is marked by adding "#".

Observation: For a string of 300 characters, only 297 will be generated. The two "#" at the beginning and the one at the end are included in the string because are part of the model's alphabet.
```

The 300 characters generated string using our model:

```
##the conlacp micumber main putestructiabou havoi.y#.us onlach 0000000 000 of ing the furope lemenclrein s to to therebat the knocappries efferes as hateng betwe yea comoncery resen of thin implauqgkqxzjkmwwpike wed li so proursethe covelimemencial havessed byjnlattenardfutut and foremis wittich ha#
```

The 300 characters generated string using **model-br.en**:

```
##thats a night.# upping a bookaboy.# ge.# xnuttellealkis thers.# jqo yout phown.#
vack.# uppy his bits on if seed its isne box hat hu# lock ther fuld ye.# ven.# mou phon
here.# judys.# a napplace byeah.# knot it.# 0yl ey.# ong.# zip of tal the he dram is rie.#
q0ox.# 0at th thats tra bed.# he can #
```

The sequence generated from the model file has a proper start with ”##” and end with ”#”. With the 30 characters {[a-z], ”0”, ”.”,” ”, ”#”]} that are used in the model file, it contains 98% of the possible trigram combinations. From just looking at the trigrams probabilities files we can say that the model file was obtain using a larger (complex) training file than our model. The majority of our probabilities should be zero if we were using MLE alone, but because of Add-One Smoothing they have non-zero values. So the model file provides better language due to the availability of data (the English language resemblance is clearer - more English words).

## 5 Question 5

We wanted to transform our model into MLE and Add-α Smoothing because we though that we could find a better alpha (better than 1) in order to minimize the perplexity and produce a better model.

We used K-Fold Cross Validation algorithm in order to train our model and find the best value for α. We split the training files into: training set (90%) and validation set (10%) in 10 different ways and implemented the algorithm. The results for best α values for 3 different models (one for each language) were: 0.12 (English), 0.13 (Spanish) and 0.13 (German).

These are the result after performing K-Fold Cross Validation algorithm over the training files:

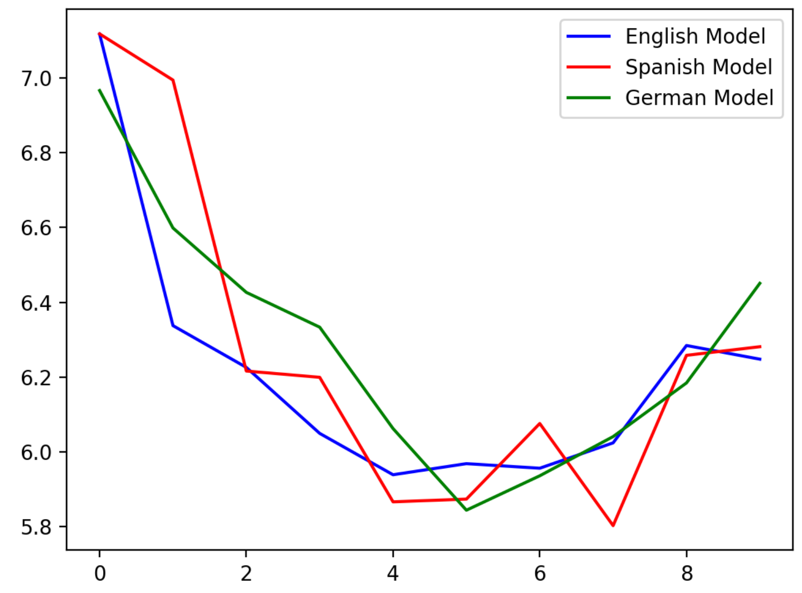

Figure 2: The perplexity for every stage of K-Fold Cross Validation algorithm

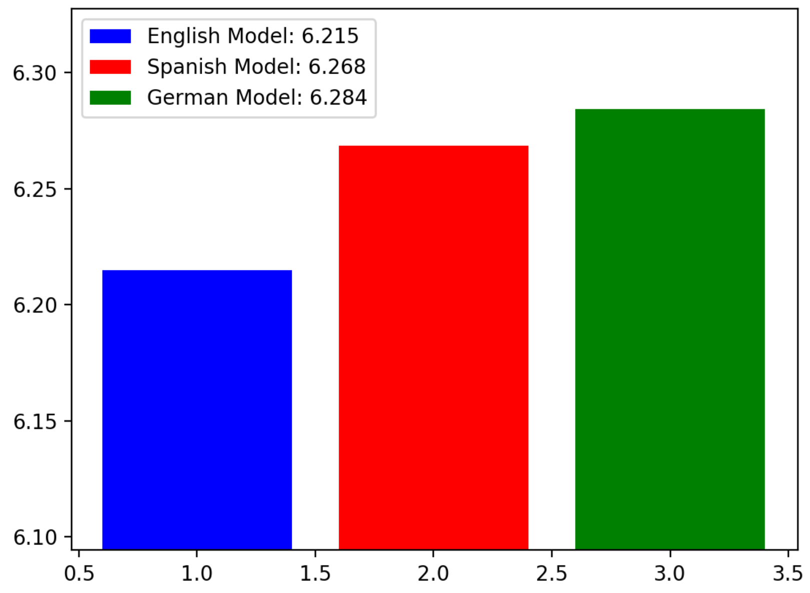

Figure 3: The average perplexity for every language model after K-Fold Cross Validation algorithm

After that we calculated the perplexity for the text file using our three language models (for both: $\alpha = 1$ and $\alpha =$ best value found) as we were asked.

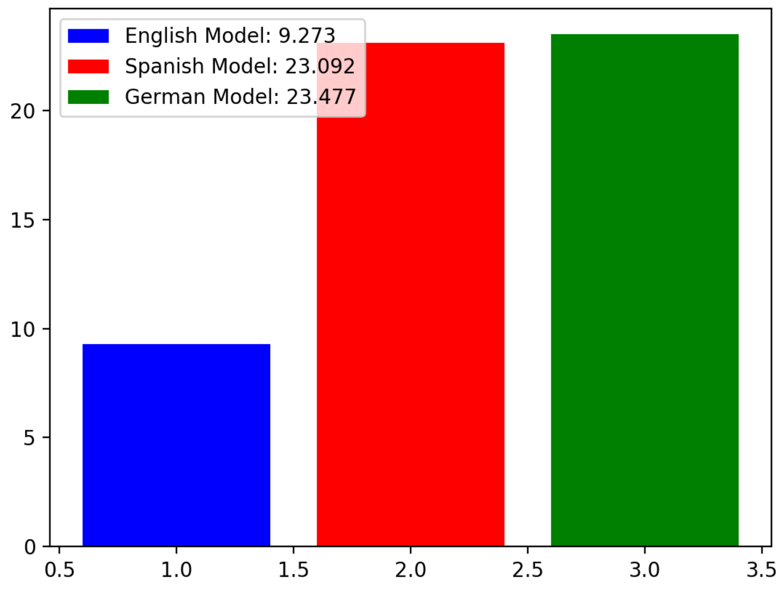

Figure 4: Perplexity for test file using all three models using $\alpha = 1$

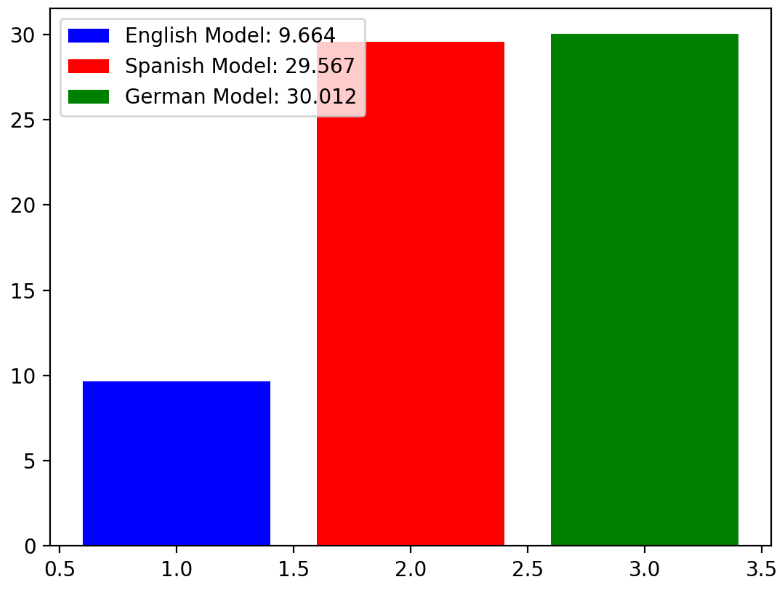

Figure 5: Perplexity for test file using all three models using $\alpha =$ best value found

We can observed that the values for perplexity are better when $\alpha = 1$ than when $\alpha =$ best value found, but on the second figure the differences between the three model are clearer than in the first figure. The perplexity of the test file with English language model is lower when compared to Spanish and German. So, we can safely assume that the test file provided is in English language because of the lower perplexity compared to the other two language models.

If the language model is checked for any other test file, we can compare the perplexity of the test files with the language model generated from training data for three languages and we can conclude the language of the file where there is lower perplexity. If we received the perplexity over a new file only for English model it will not be enough to determine if the text is written in English or not. Even if we receive the perplexity values over multiple models it could be impossible to determine in what language is that text written because we may encounter two very close perplexity values.