# Accelerated Natura Language Processing

# Assignment-1

1) Preprocess line:

def preprocess\_line(line): #Function for preprocessing and cleansing data from special characters

char\_removed = re.sub('[^a-zA-Z0-9. ]', "", line) #This line removes the characters other than alphabets, numbers , dot and space

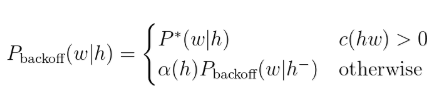
number\_formated = re.sub('[0-9]', "0", char\_removed) #Replaces each digit with zero

string\_lower = number\_formated.lower() #Converts the string from the line above to lower case

2) This probability distribution uses Katz backoff model for trigram probability calculation.

For example: If we take the trigram ' cq', there are no words in english language that start with "cq" and the probability of finding a word with " cq" is zero. As per Katz backoff 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 (probability of " c" assumed). This has also been observed with other trigrams like " cp" -has the probability of " c"," "," 0[a-z]" - has the probability of " 0" - which are less likely to appear in the language.

Formula for calculation of Probability with Katz backoff model:



3) The model that we have used for calculating the Probability distributions for Language model is Add-alpha smoothing with Maximum Likelihood Estimation (MLE). We have chosen this model as the training set is limited and the test set may have some unknown trigrams which are not encountered in training set. We have chosen Add-alpha smoothing to regularize the language model.

Algorithm used for Language model probability generation:

Step-1: Append '##' to mark the beginning of the line and '#' at the end to mark the end of line.

Step-2: Calculate the count of all Trigrams from the training corpus generated after pre-processing of the data.

Step-3: Calculate the count of all bigrams from the training corpus generated after pre-processing of the data.

Step-4: Calculate the probability of the trigram by using the formula:

probability(trigram)=(count of Trigram+alpha)/(count of bigram + (alpha\*v))

where v is the number of allowed characters from alphabet which is (a to z), #,0,., <space> - 30 characters

and alpha is the discount factor which reduces the probability of high frequency trigram by a factor and distributes it to the low/no frequency trigrams.

We have performed the algorithm for varying values of alpha between 0 and 1 and taken the value where there is low perplexity. This gives us the trigrams with highest probability.

Step-5: We have split the corpus into 3 sets for validation of the language model. 80% of the training corpus is used as traning set, 10% of it is used as validation set, the remaining 10% is used as test set.

Step-6: The output is written to a file with system parameter "outfile". The name of the file can be given at runtime.

The output is written in the following format

trigram"\t"probability(in exponential notation)

Ex: esu 1.312e-02

sum 3.312e-02

4) generate\_from\_LM is the function to generate random sequences of 300 characters from the language model. The function is coded as follows:

Step-1: Get all the trigrams, probabilities of the training data provided.

Step-2: The string is initialised to ## to get a start of the sequence.

Step-3: For every bigram identified from the trigram in the generated sequence, the last character of the trigram is printed at random. This loop continues until we reach 300 characters. Then the # from the beginning and end of the sentence are removed.

Step-4: When we encounter a trigram from the sequence where there is no match in the dictionary, we are using one random alphabet from the set [a-z0. #] to continue the sequence.

Sample output:

##hat a dading.# play.# yout.# now whadys.# likena meaddy.# gonly.# 0vy.# for cran take is his.# uppee of to yout.# sh this tow pust in ye hey.# and me.# rie there.# sair cats thim is.# 0bszhjakfclo wank a the you dook.# righ.# rand to theres youre scing eahch himcymhky.# of.# for on telen tone.# y#

We have implemented the same logic for the file model-br.en and the generated output sequence is as follows:

##ing istake an thaten cormote unissity th mon outchropeas the thave es alst dopeen haspet. 00000.0 zoregiondes oust as. 00c reeneente re prove loposay to briont self thissaferes. 0000000 of the commin whe se funerm. 00 and act eforstationspo cauesion inlissin of the of requitions amemen preare ti#

The sequence generated from the model file has a proper start with ## and end with #. With the 30 characters [a-z0.# ] that are used in the model file, it contains 98% of the possible trigram combinations. So it provides better language due to the availability of data.

5) K-fold validation is performed on the training data to calculate perplexities by dividing the data into "k" parts. We have taken the value of K to be 10. The training and validation data are taken 10-fold from the training set and the perplexities are calculated. The final perplexity is provided as an average of the 10-fold data.

We have observed that the perplexity for all the languages with our chosen language model is in the same limits. So it is not possible for us to identify the language given the perplexity. One observation from our language model is that English has lower perplexity compared to other languages but this cannot solely determine the language as it is not possible to benchmark the perplexity for a language.