NLP – Assignment 2 – Spell Checker – report

1. **def learn\_language\_model(files, n=3, lm=None):**

In this function I iterate over files. I parse and normalize each word. Next thing I split the words into ngrams using the nltk.ngrams func with two sided blank space padding. We basically do the same process from previous assignment with different split. The first n-1 itens in ngram is the key and last item is the key of the nesteddict with the count being the value.

If lm is given I merge the given lm with the newly created.

1. **def create\_lexicon(files, lexicon=collections.defaultdict(int)):**

Here we just create a default dict of each word in the corpus and its count.

If lexicon given we Updating it.

1. **def create\_error\_distribution(errors\_files, lexicon):**

In this func we process the error files. We than iterate over each line and getting the error and the correct words. For each error+correct tuple we identify the error that happened this by checking some condition.

If the length of the correct word is higher than the error – deletion.

If the length of the correct word is lower than the error – insertion.

If it’s the same length and the count of each letter in the words is the same – transposition.

If it’s the same length and the count of each letter in the words is not the same – substitution.

For each of those error we count the times that it happened and then divide it by the number of appearances of the letter/s in the corpus.

The final dict that created consists of 4 keys for each error and the value is a dict where the key tuple of letter/s and the value is the probability.

1. **def correct\_word(w, word\_counts, err\_dist, c=1):**

In this func we are trying to correct the given word. First thing that we do is normalize the word in the same way that we normalized the corpus. If the word exists in the lexicon then we are returning the word. Next thing we do is we use the Levinstein-distance function )<https://gist.github.com/pombredanne/0d83ad58f45986ddeb0917266e106be0>(

That calculates the distance between 2 words (Counts transposition as 1). With this I found all the words in lexicon that their levinstein distance from the given word is 1. Then I created a dict where the key is the possible correct word and the value is its probability.

The probability is the condition probability P(w/s)=P(s/w)\*p(w) .

Where p(s/w) is taken from the err\_dist. We basically to the same method as in previous func. We find the error that happened, than find the right tuple that got inserted/deleted/substituted/transpositioned and find its probability in the err\_dist. P(W) is the probability of the word in the lexicon. Number of its occurrences divided by the corpus size.

We multiply those 2 parameters and this is the probability for the specific possible word that we put in possible words dict.

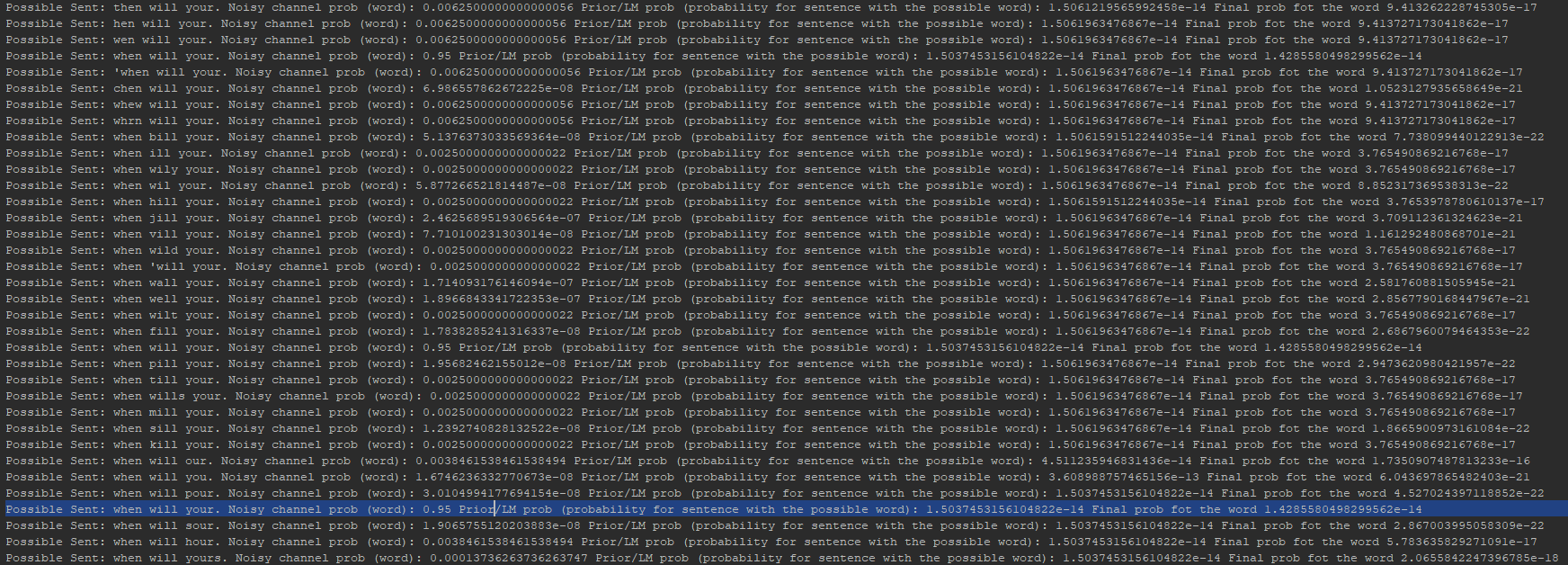
After doing this process on all possible words we return the word with the maximum probability.

1. **def correct\_sentence(s, lm, err\_dist, c=1, alpha=0.95):**

First thing that we do is normalize the sentence in the same way that we normalized the corpus. Than we do a similar method to the previous func. Now for each word of the sentence we find possible words and calculate the probability of the word – (Noisy channel for the error multiplied by the word probability in the corpus). But this time this is the Noisy channel probability. And the Prior probability is the probability of the new sentence where we replace the word with its possible word. This prior probability is calculated by the same method (evaluate\_text) that we made in the previous assignment.

Another difference from the correct\_word func is that if the word exists in the corpus we are still

If the word that we are checking exists in the corpus we add it to out possibility dict and give it a noisy channel probability alpha. By this we hope that even if the word is a fine word but not in the right context the lm probability will reduce it’s probability to return signicantly.



Helper Functions:

def proccess\_file(file, flag=None):

returns files content as a string

def normalization(word):

normalizes the given word

def check\_numbers\_letters(word):

True if the word is a number

def default\_to\_regular(d):

turns nested defaultdict to a regular nested dict

def create\_word\_list(file\_list):

returns a list of words out of a list of files

def diff\_insertion(err, corr):

def diff\_del(err, corr):

def diff\_sub(err, corr):

def diff\_trans(err, corr):

Similar functions that returns tuples of the error letter and the right one (like in a matrix)

def create\_letter\_count(lexicon):

creates a letter count of a lexicon

def levenshteinDistance(a, b):

returns levinstein distance between 2 strings

def extract\_n(lm):

Extracts n (ngram) out of language model

def evaluate\_text(s, n, lm, ngram\_dict):

returns probability of a sentence

def get\_n\_words(list, index, n):

returns n-1 grams from the current word

def get\_lex(lm):

creates a set of unique words from language model.