Fresh_start

November 26, 2018

In [1]:
```
# these are all of our imports
import re
import string
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer as wnl
from nltk.metrics.distance import edit_distance as lev
```

In [2]:
```
# contractions can be miss categorized as incorrect when you strip and filter the data
#fixes are uncontracted contractions
#stop_en and stop_germ are the stopwords
contractions = ["´","‘","’","it's", "he's","she's","that's", "what's", "there's", "aren't",\
            "[newline]", "'m", "'ve","n't", "'ll","'re", "won't", "'d","geht's","gibt's", "'s", ' xd']
fixes = ["'","'","'","it is", "he is","she is","that is", "what is", "there is","are not",\
                 " ", " am", " have", " not", " will", " are", "will not", " would","geht es","gibt es", "", ' ']
stop_en = set(stopwords.words('english'))
stop_germ = set(stopwords.words('german'))
```

In [3]:
```
#opening our german and english dictionairies
de_dic = {}
with open('german.dic', 'r', encoding='latin-1') as f:
    for row in f:
        if len(row) >1:
            de_dic[row.strip().lower()] = 1
en_dic = {}
with open('english.dic', 'r',) as f:
    for row in f:
        en_dic[row.strip().lower()] = 1
```

In [4]:
```
#words that are not in the english dictionary but should be- if we don't do this then these words will be erroneosly
#labeled as misspells
en_dic['anymore'] = 1
en_dic['adhd'] = 1
en_dic['asshole'] = 1
en_dic['fucking'] = 1
en_dic['porn'] = 1
en_dic['fuck'] = 1
en_dic['proud'] = 1
en_dic['others'] = 1
en_dic['mom'] = 1
en_dic['ptsd'] = 1
en_dic['europe']= 1
en_dic['tumour'] = 1
en_dic['tumours'] = 1
en_dic['stats'] = 1
en_dic['favourite'] = 1
en_dic['boyfriend'] = 1
en_dic['fortnite'] = 1 #game name
en_dic['bts'] = 1 # Korean pop group

#we considered also doing this for acronyms because they are not accidental misspells they are just not in our dictionary
#but utimately for the sake of the assignment and the dictionary given to us we decided not to input them into our dictionary
# en_dic['lol'] = 1 #laugh out loud
# en_dic['omg'] = 1 #oh my god
# en_dic['af'] = 1 #as fuck
# en_dic['tbh'] = 1 #to be honest
# en_dic['bc'] = 1 #because
# en_dic['idk'] = 1 #i don't know
# en_dic['rn'] = 1 #right now
# en_dic['ppl'] = 1 #people
# en_dic['lmao'] = 1 #laugh my ass off
# en_dic['wtf'] = 1 #what the fuck
# en_dic['btw'] = 1 #by the way
# en_dic['pls'] = 1 #please
# en_dic['thx'] = 1 #thanks
#en_dic['aml'] = 1 #anti-money laundering
```

In [5]:
```
# Terms not in German Dictionary that should be, we are all native English speakers
#so there could be more than just this
de_dic['sowas'] = 1
```

In [6]:
```
# to open and read our tweets to begin the preprocessing step
raw_text = open('tweets.csv').read()
tab_seperated = [item.split('\t') for item in raw_text.split('\n') if len(item.split('\t')) >= 4]
```

In [7]:
```
#Non-English or German characters, used to filter out foreign tweets
non_ende_char = ["á","à","ã","ă","â","é","è","ê","í","ì","ĩ","ó","ò","õ","ô","o","ú","ù","ũ","u","ý","ỳ","đ","ñ"]
```

In [8]:
```
#Building our tweet filter
filtered = []
for i in tab_seperated:
    for char in non_ende_char:
        if char in i[4]:
            filtered.append(i[1])
```

In [9]:
```
#we use regex to remove any websites, numbers, tagged twitter users, hashtags,
#and basically anything that is not in the English and German alphabets
data = {}
for i in range(len(tab_seperated)):
    #extracting the filtered tweets and IDs
    if tab_seperated[i][1] not in filtered:
        data[tab_seperated[i][1]] = tab_seperated[i][4]

#Preprocessing and breaking apart common contractions
data_index = data.copy()
for tweet in data_index.keys():
    data_index[tweet] = data_index[tweet].lower()
    for i in range(len(contractions)):
        if contractions[i] in data_index[tweet]:
            data_index[tweet] = data_index[tweet].replace(contractions[i], fixes[i])
    data_index[tweet] = re.sub('https?[^\s]+', ' ' , data_index[tweet])
    data_index[tweet] = re.sub('[@#][^\s]+', ' ' , data_index[tweet])
    data_index[tweet] = re.sub(r'[0-9][^\s]+', ' ' , data_index[tweet])
    data_index[tweet] = re.sub(r'\w+\.[^\s]+', ' ' , data_index[tweet])
    data_index[tweet] = re.sub(r'[^a-zäöüß\s]', ' ', data_index[tweet])
    data_index[tweet] = re.sub(r'[^\w\s]', ' ' , data_index[tweet])

#Here we are building our dict of terms and frequenceis, as well as a dict with sorted docIDs
terms = {}
term_index = {}
for num,tweet in data_index.items():
    for word in tweet.split():
        if word in term_index:
            term_index[word].append(num)
        elif word not in term_index:
            term_index[word] = [num]
        if word in terms:
            terms[word] += 1
        else:
            terms[word] = 1

for key in term_index.keys():
    term_index[key] = sorted(term_index[key])
```

In [10]:
```
#this function determines if a word is German or English
 #Given on number of tweets within the postings list based on stop words and foreign characters
 def is_language(term):
         de_char = ['ä','ö','ü','ß']
         de_score = 0
         en_score = 0
         for post in term_index[term]:
             for i in data_index[post].strip().split():
                 for char in de_char:
                     if char in i:
                         de_score +=1
                 if i in stop_germ:
                     de_score += 1
                 if i in stop_en:
                     en_score += 1
         if de_score>en_score:
             return 'german'
         elif de_score<en_score:
             return 'english'
         else:
             return None
```

In [11]:
```
#This function is used to find which words are the most frequent in there respective langauges
def top_freq(dict):
        freq = []
        for term, val in dict.items():
            freq.append((val, term))
        freq = sorted(freq)[::-1]
        freq_de = []
        freq_en = []
        for i,j in freq:
            if is_language(j) == 'german':
                freq_de.append(j)
            else:
                freq_en.append(j)
        return freq_de, freq_en
```

In [12]:
```
#Generated the english and german terms, sorted by descending frequency
top_de, top_en = top_freq(terms)
```

In [13]:
```
#This function is used to find out if the tweet is English or German
#Here we use a refined search to better identify if a tweet is english or Germn
#given stop words and our most frequent words occuring in English or German
#If it fails to identify, it will return none
def language(post):
        de_char = ['ä','ö','ü','ß']
        de_score = 0
        en_score = 0
        for i in data_index[post].split():
            for char in de_char:
                    if char in i:
                        de_score +=1
            if i in stop_germ or i in top_de[:200]:
                de_score += 1
            if i in stop_en or i in top_en[:200]:
                en_score += 1
        if de_score>en_score:
            return 'german'
        elif en_score>de_score:
            return 'english'
        else:
            return None
```

In [14]:
```
#This function finds misspelled words
def get_misspells():
        #Here we generate a list of terms and list to partition them in
        words =sorted([key for key in terms.keys()])
        de = []
        en = []
        correct_en = []
        correct_de = []
        for word in words:
            g_count = 0
            e_count = 0
            e_posts = []
            g_posts = []
            #here we sort correct words separate correct_addicitonal terms
            if word in en_dic:
                correct_en.append(word)
                continue
            elif word in de_dic:
                correct_de.append(word)
                continue
            #pos tags for lemmatization generation
            tags = ['n','v','a','s','r']
            en_lemma = {}
            for tag in tags:
                lemma = wnl.lemmatize(wnl,word=word, pos=tag)
                en_lemma[lemma]= 1
#             #German lemmatiziaton/stemming
            de_lemma = word

        #here we search post by post to determin if term in index
        # is German or English, to see if in the context it is a misspell
            for post in term_index[word]:
                if language(post) == 'german':
                    ## get german misspelling count
                    if de_lemma not in de_dic:
                        if len([i for i in en_lemma.keys() if i in en_dic]) == 0:
```

```
                            g_count += 1
                            g_posts.append(post)
                    else:
                        correct_de.append(word)

                elif language(post) == 'english':

                    ## get german misspelling count
                    if len([i for i in en_lemma.keys() if i in en_dic]) == 0:
                        e_count += 1
                        e_posts.append(post)
                    else:
                        correct_en.append(word)

            #Here if the refined methon is not able to solve,
            #It falls back to the likelihood of based on occurance in German or English Tweets
                else:
                    lang = is_language(word)
                    if lang == 'german':

                        ## get german misspelling count
                        if de_lemma not in de_dic:
                            if len([i for i in en_lemma.keys() if i in en_dic]) == 0:
                                g_count += 1
                                g_posts.append(post)
                        else:
                            correct_de.append(word)
                    if lang == 'english':
                        ## get english misspelling count
                        if len([i for i in en_lemma.keys() if i in en_dic]) == 0:
                            e_count += 1
                            e_posts.append(post)
                        else:
                            correct_en.append(word)

            if g_count > 0:
                de.append((g_count, word, g_posts))
            if e_count > 0:
                en.append((e_count, word, e_posts))

        #retruning list of misspells and correct lists
        return sorted(de)[::-1], sorted(en)[::-1], correct_de, correct_en
```

In [15]:
```
#Here we generate our lists of misspells
de_mis, en_mis, terms_de, terms_en = get_misspells()
```

In [16]:
```
#This function calculates the Damerau distance of the misspelled words that were found in the function above
def en_damerau(word):
    #here we reduce the workload by reducing same characters
    #that occur more than twice in a row to just two characters to increase accuracy
    term = ''
    for i in range(len(word)):
        try:
            if word[i] != word[i+2]:
                term += word[i]
        except:
            term += word[i]

    alphabet = "abcdefghijklmnopqrstuvwxyz"
    possible = {}

    #we are genearting a dictionary of terms that are 1 damerau distance from the term
    chunks = [(term[:i], term[i:])for i in range(len(term) + 1)]
    for chunk1, chunk2 in chunks:
        if chunk2:
            #subtraction
            possible[chunk1+chunk2[1:]] = 1
            for char in alphabet:
                #substitution
                possible[chunk1+char+chunk2[1:]] = 1
        if len(chunk2) > 1:
            #transposition
            possible[chunk1+chunk2[1]+chunk2[0]+chunk2[2:]] = 1
        for char in alphabet:
            #addition
            possible[chunk1+char+chunk2] = 1
    return possible
```

In [17]:
```
#This function finds the suggested terms based on the damerau distance for English words
def en_suggested(term):
    """Some terms such as the following can be force edited
       They are slang to represent expressions"""
#     if term == 'kinda':
#         return "kind of"
#     if term == 'gonna':
#         return 'going to'
#     if term == 'wanna':
#         return 'want to'
    suggestions = en_damerau(term)
    suggested = []
    #Lemmatizing the suggestions to get a more accurate reference
    for i in suggestions.keys():
        tags = ['n','v','a','s','r']
        lemmas = []
        for tag in tags:
            lemmas.append(wnl.lemmatize(wnl,word=i, pos=tag))
        lemmas = set(lemmas)
        for j in lemmas:
            if j in en_dic:
                suggested.append(i)
    #Refining the words that were suggested using edited distance again
    refined = [word for word in suggested if lev(term, word) == min(lev(term,word) for word in suggested)]
    try:
        #further refining the search
        best = sorted([(terms[word],word) for word in refined if word in terms_en])[::-1]
        return [i for j,i in best][:3]
    except:
        return refined
```

In [18]:
```
#This function finds the damerau distance for the misspelled German words

def de_damerau(word):
    #here we reduce the workload by reducing same characters
    #that occur more than twice in a row to just two characters to increase accuracy
    term = ''
    for i in range(len(word)):
        try:
            if word[i] != word[i+2]:
                term += word[i]
        except:
            term += word[i]
    alphabet = "abcdefghijklmnopqrstuvwxyzäöüß"
    possible = {}
    #we are genearting a dictionary of terms that are 1 damerau distance from the term
    chunks = [(term[:i], term[i:])for i in range(len(term) + 1)]
    for chunk1, chunk2 in chunks:
        if chunk2:
            #subtraction
            possible[chunk1+chunk2[1:]] = 1
            for char in alphabet:
                #substitution
                possible[chunk1+char+chunk2[1:]] = 1
        if len(chunk2) > 1:
            #transposition
            possible[chunk1+chunk2[1]+chunk2[0]+chunk2[2:]] = 1
        for char in alphabet:
            #addition
            possible[chunk1+char+chunk2] = 1
    return possible
```

In [19]:
```
##This function finds the suggested terms based on the damerau distance for German words

def de_suggested(term):
    suggestions = de_damerau(term)
    suggested = []
    for i in suggestions:
        if i in de_dic:
            suggested.append(i)
    #Refining the words that were suggested using edited distance again
    refined = [word for word in suggested if lev(term, word) == min(lev(term,word) for word in suggested)]
    try:
        #further refining the search
        best = sorted([(terms[word],word) for word in refined if word in de_dic and word in terms_de])[::-1]
        return [i for j,i in best][:3]
```

```
            except:
                return refined
```

In [20]:
```
#This gives us the top misspelled English words
top_mis_en = []
for count, word, posts in en_mis[:10]:
    top_mis_en.append((word, count, en_suggested(word)))
```

In [21]:
```
#it should be noted that even though some of the outputted words were not intentional misspells they were not in our dictionary
#so they were considered misspells we ultimately decided to leave them out in order to comply to the constraints
#of the dictionarys that were given to us in the assignment
top_mis_en
```

Out[21]:
```
[('kinda', 2290, ['kind', 'linda']),
 ('bc', 712, ['be', 'by', 'b']),
 ('gonna', 492, ['donna', 'gonne', 'gona']),
 ('lol', 395, ['vol', 'lo', 'pol']),
 ('omg', 377, ['om', 'og']),
 ('wanna', 330, ['anna', 'canna', 'hanna']),
 ('rn', 297, ['in', 'an', 'on']),
 ('tbh', 284, ['th', 'tch']),
 ('idk', 261, ['id', 'ink', 'ilk']),
 ('ppl', 217, ['pol', 'pal'])]
```

In [22]:
```
#This gives us the top misspelled German words
top_mis_de = []
for count, word, posts in de_mis[:10]:
    top_mis_de.append((word, count, de_suggested(word)))
```

In [23]:
```
#Top 10 German misspells
top_mis_de
```

Out[23]:
```
[('nen', 424, ['ren', 'nn']),
 ('nochmal', 294, ['nochmals']),
 ('nem', 233, ['dem', 'neu', 'nm']),
 ('erstmal', 216, ['erstmals']),
 ('lol', 214, ['hol', 'mol', 'aol']),
 ('daß', 180, ['saß', 'maß', 'dax']),
 ('gibts', 164, ['gibt', 'gifts']),
 ('nich', 151, ['nicht', 'noch', 'sich']),
 ('zb', 147, ['zu', 'ob', 'tb']),
 ('vllt', 146, [])]
```