Assignment - CS 5615 Information Retrieval

M.R.G.Vijithasena 209387N

GitHub Link : https://github.com/rasikavijithasena/TokenizationAssignment

1. Tokenization

Used tokenizers :

For Tweeter data : ntlk.TweetTokenizer
For Research paper data : ntlk.word_tokenize
For Student course feedback : ntlk.word_tokenize

Word_tokenize
It is a most popular tokenize method used for tokenizing sentences. It separates words using spaces and punctuations.

Tweettokenizer
This is the best tokenizer for tokenizing tweets. Twitter-aware tokenizer, designed to be flexible and easy to adapt to new domains and tasks.

class nltk.tokenize.casual.**TweetTokenizer**(*preserve_case=True*, *reduce_len=False*, *strip_handles=False*)

To compare tokenizations following tokenizations are used.

RegexpTokenizer
A RegexpTokenizer splits a string into substrings using a regular expression. For example, the following tokenizer forms tokens out of alphabetic sequences, money expressions, and any other non-whitespace sequences. Tokenize a string, treating any sequence of blank lines as a delimiter. Blank lines are defined as lines containing no characters, except for space or tab characters.

WhitespaceTokenizer

Tokenize a string on whitespace (space, tab, newline). In general, split() method provide the same functionality.

WordPunctTokenizer

Tokenize a text into a sequence of alphabetic and non-alphabetic characters, using the regexp \w+|[^\w\s]+. It splits all punctuations into separate tokens.

Discussion

Research paper and feedback data includes raw data, including whitespace, line breaks and blank lines. In tokenization, we need to break strings and need to get meaningful words. So in these two cases word tokenization has been used. It successfully separated meaningful words from punctuations, spaces and other unimportant things.

For tweeter data tokenization, tweet tokernition is selected from other tokenizations. TweetTokenizer is built for analyzing tweets. The main difference between word_tokernizer and tweettokenizeris TweetTokenizer keeps hashtags intact while word_tokenize doesn't. It cares about emoji icons, hash tags and other texts which contain social media related texts. It can tokenize more accurately other than other tokenizations.

Tweet data

```
#read from file
fileObjectTwitter = open("twitter.txt", "r", encoding='utf-8')
twitterdata = fileObjectTwitter.read()

tweet_tokenizer = TweetTokenizer()
twitterTokens = tweet_tokenizer.tokenize(twitterdata)

print(twitterTokens)
```

```
['Reminds', 'me', 'of', 'Liberal', 'Immigration', 'Fraudster', 'Monsef', 'avoiding', 'deportation', 'from', 'Canada', '.', '#cd
npoli', '#LPC', '#CPCLDR', '', '', '', '_', 'https://t.co/ZOZOSe1CqQ', '#immigration', '#integration', '#canada', 'https://t.co/
M5cKGyvV8F', 'We', 'want', 'controlled', 'immigration', 'that', 'contributes', 'positively', 'to', 'the', 'UK', 'economy', '.',
'Same', 'as', 'Australia', '&', 'Canada', '.', 'https://t.co/99mYliuOes', 'Is', 'the', 'new', 'Manitoba', 'immigration', 'fee',
'a', 'head', 'tax', '?', 'https://t.co/LsG7C3vLe9', 'Canada', 'immigration', 'profit', 'influence', 'modernistic', 'delhi', 'ye
t', 'abhinav', ':', 'XKofy', 'https://t.co/becgusY2i6', 'Canada', 'Immigration', 'Minister', 'to', '', '', '', 'Substantia
```

Research paper data

```
researchfile = open("researchpaper.txt", "r", encoding='utf-8')
raw = researchfile.read()
#data = BeautifulSoup(raw).get_text()
researchTokens = nltk.word_tokenize(raw)
print(researchTokens)
```

```
['Neural', 'network', 'models', 'have', 'shown', 'their', 'promising', 'opportunities', 'for', 'multi-task', 'learning', ',',
'which', 'focus', 'on', 'learning', 'the', 'shared', 'layers', 'to', 'extract', 'the', 'common', 'and', 'task-invariant', 'feat
ures', '.', 'However', ',', 'in', 'most', 'existing', 'approaches', ',', 'the', 'extracted', 'shared', 'features', 'are', 'pron
e', 'to', 'be', 'contaminated', 'by', 'task-specific', 'features', 'or', 'the', 'noise', 'brought', 'by', 'other', 'tasks',
'.', 'In', 'this', 'paper', ',', 'we', 'propose', 'an', 'adversarial', 'multi-task', 'learning', 'framework', ',', 'alleviatin
```

Student feedback data

```
fileObjectFeedback = open("feedback.txt", "r", encoding='utf-8')
feedbackRawData = fileObjectFeedback.read()
#feedbackData = BeautifulSoup(feedbackRawData).get_text()
feedbackTokens = nltk.word_tokenize(feedbackRawData)

print(feedbackTokens)
```

```
['Honestly', 'last', 'seven', 'lectures', 'are', 'good', '.', 'Lectures', 'are', 'understandable', '.', 'Lecture', 'slides', 'a
re', 'very', 'useful', 'to', 'self-study', 'also', '.', 'The', 'given', 'opportunity', 'to', 'ask', 'questions', 'from', 'the',
'lecturer', 'is', 'appreciative', '.', '``', 'Good', ':', ')', '<', 'br', '/', '>', 'please', 'do', 'recap', 'at', 'class', 'st
arting', 'it', '&', '#', '039', ';', 's', 'better', 'for', 'us', '.', '<', 'br', '/', '>', 'sometimes', 'teaching', 'speed', 'i
s', 'very', 'high', '.', '<', 'br', '/', '>', '<', 'br', '/', '>', 'Thanks', '!', ':', ')', '<', 'br', '/', '>', '``', 'The',
```

2. **Isolated word correction**

Used Library: pyspellchecker

This is a simple spell checking algorithm which uses a Levenshtein Distance algorithm to find permutations within an edit distance of 2 from the original word. It then compares all permutations (insertions, deletions, replacements, and transpositions) to known words in a word frequency list. Those words that are found more often in the frequency list are more likely the correct results.
It provides the misspelled words. And if a word is misspelled the mostly correct word and list of likely words are provided by it.

Features
It supports multiple languages including English, Spanish, German, French, and Portuguese. Dictionaries were generated using the WordFrequency project on GitHub.

It supports Python 3 and Python 2.7 but, as always, Python 3 is the preferred version.

It allows for the setting of the Levenshtein Distance to check. For longer words, it is highly recommended to use a distance of 1 and not the default 2. See the quickstart to find how one can change the distance parameter.

Used methods :

correction(word): Returns the most probable result for the misspelled word
candidates(word): Returns a set of possible candidates for the misspelled word
unknown([words]): Returns those words that are not in the frequency list
edit_distance_1(word): Returns a set of all strings at a Levenshtein Distance of one based on the alphabet of the selected language

edit_distance_2(word): Returns a set of all strings at a Levenshtein Distance of two based on the alphabet of the selected language

Discussion:

Checking the spelling of the word is one of the most important things in text processing. So I have used a simple spell checking algorithm called pyspellchecker.

Edit distance is a way of quantifying how dissimilar two strings (e.g., words) are to one another by counting the minimum number of operations required to transform one string into the other. Edit distances find applications in natural language processing, where automatic spelling correction can determine candidate corrections for a misspelled word by selecting words from a dictionary that have a low distance to the word in question. In this use-case edit distance method has been used to isolated word correction. So in these three datasets, most words are

corrected as example in twitter dataset #tag has been removed and correct word has been identified in most cases. And also in feedback dataset and research paper dataset also most of words have been considered. Following figure shows sample results in the feedback dataset.

Output of feedback dataset:

```
helpfull -> mostly liked answer : helpful
Mostly Likely Options : {'helpful', 'helpfully'}


undersatand -> mostly liked answer : understand
Mostly Likely Options : {'understand'}


examples.lectures -> mostly liked answer : examples.lectures
Mostly Likely Options : {'examples.lectures'}


interesting.we -> mostly liked answer : interesting.we
Mostly Likely Options : {'interesting.we'}


class.it -> mostly liked answer : classic
Mostly Likely Options : {'classic'}
```

Context sensitive word correction

Used Library : Symspell

class symspellpy.symspellpy.SymSpell(max_dictionary_edit_distance=2, prefix_length=7, count_threshold=1)

Symmetric Delete spelling correction algorithm. initial_capacity from the original code is omitted since python cannot preallocate memory. compact_mask from the original code is omitted since we're not mapping suggested corrections to hash codes.

Using word_segmentation method divides a string into words by inserting missing spaces at the appropriate positions misspelled words are corrected and do not affect segmentation; existing spaces are allowed and considered for optimum segmentation.

Enum class - Verbosity
class symspellpy.symspellpy.Verbosity

Controls the closeness/quantity of returned spelling suggestions.

- ALL = 2 : All suggestions within maxEditDistance, suggestions ordered by edit distance, then by term frequency (slower, no early termination).
- CLOSEST = 1 : All suggestions of smallest edit distance found, suggestions ordered by term frequency.
- TOP = 0 : Top suggestion with the highest term frequency of the suggestions of smallest edit distance found.

Discussion

In here, I have used a dictionary file called word.txt as corpus. Symspell is a Symmetric Delete spelling correction algorithm which provides much higher speed and lower memory consumption. Context-sensitive spelling correction is the task of correcting spelling errors which result in valid words.

Context-sensitive spelling correction is the task of correcting spelling errors which result in valid words. For example, in the sentence I want a peace of cake, peace is a valid word in isolation but an error in this context (should be piece). Most spelling correction systems check one word at a time and do not correct such errors.It is important to correct these kinds of errors. But in this case suitable corpus should be provided.

Output of algorithm:

```
print(correct_tocknized_text(twitterTokens))

['Reminds', 'me', 'of', 'Liberal', 'Immigration', 'Frauds', 'Monsey', 'avoiding', 'deportation', 'from', 'Canada', 'a', 'cdn',
'lpc', 'cp', 'a', 'a', 'a', 'ttp', 'immigration', 'integration', 'canada', 'ttp', 'We', 'want', 'controlled', 'immigration', 't
hat', 'contributes', 'positively', 'to', 'the', 'Uk', 'economy', 'a', 'Same', 'as', 'Australia', 'a', 'Canada', 'a', 'ttp', 'I
s', 'the', 'new', 'Manitoba', 'immigration', 'fee', 'a', 'head', 'tax', 'a', 'ttp', 'Canada', 'immigration', 'profit', 'influen
ce', 'modernistic', 'delhi', 'yet', 'abhinaya', 'a', 'Azofy', 'ttp', 'Canada', 'Immigration', 'Minister', 'to', 'a', 'a', 'a',
'Substantially', 'Increase', 'Immigration', 'Numbers', 'hot', 'hot', 'M', 'a', 'a', 'me', 'les', 'dusa', 'up', 'ays', 'dim', 'p
```

3. **Stemmer and a Lemmatize**

Used Stemmer : PorterStemmer

It removes the common endings to words so that they can be resolved to a common form. It is an older stemming algorithm. It is not too complex and development on it is frozen. It guarantees reproducibility and is good for research purposes.

Used Lemmatizer : WordNetLemmatizer

Wordnet is a large, freely and publicly available lexical database for the English language aiming to establish structured semantic relationships between words. It offers lemmatization capabilities as well and is one of the earliest and most commonly used lemmatizers.

Discussion

For grammatically reasons words are used in different forms such as wait, waiting, waited. These words carried somewhat similar meanings. In many situations, it seems as if it would be useful for a search for one of these words to return documents that contain another word in the set. The goal of stemming and lemmatization is reducing inflectional forms and getting derivationally related forms of a word to a common base form.

Stemming is a procedure to reduce all words with the same stem to a common form whereas lemmatization removes inflectional endings and returns the base or dictionary form of a word. Lemmatizing is a similar operation as stemming. The major difference between stemming is, it creates non-existence words. But Lemmas create actual words.

Lemmatization is slower than stemming because it uses a corpus to supply lemma. And to get the proper lemma a parts-of-speech have to define. And also it is harder to create a new lemmatizer for new language than a stemming algorithm. That's because lemmatizers need more information regarding the structure of the language.

Stemming precision is low and recall is higher because it gets all documents which contain root words, but those all do not contain correct value.

Stemming

```
twitter_words= twitterTokens
ps =PorterStemmer()
for w in twitter_words:
    rootWord=ps.stem(w)
    print(w + ": " +rootWord)
```

```
Are: are
people: peopl
still: still
moving: move
to: to
#Canada: #canada
```

Lemmatizer

```
tokenization = nltk.word_tokenize(feedbackRawData)
for w in tokenization:
    print("Lemma for {} : {}".format(w, wordnet_lemmatizer.lemmatize(w)))
```

```
Lemma for Lecture : Lecture
Lemma for slides : slide
Lemma for are : are
Lemma for very : very
```