Steps for effective text data cleaning (with case study using Python)

The days when one would get data in tabulated spreadsheets are truly behind us. A moment of silence for the data residing in the spreadsheet pockets. Today, more than 80% of the data is unstructured – it is either present in data silos or scattered around the digital archives. Data is being produced as we speak – from every conversation we make in the social media to every content generated from news sources. In order to produce any meaningful actionable insight from data, it is important to know how to work with it in its unstructured form. As a Data Scientist at one of the fastest growing Decision Sciences firm, my bread and butter comes from deriving meaningful insights from unstructured text information.

One of the first steps in working with text data is to pre-process it. It is an essential step before the data is ready for analysis. Majority of available text data is highly unstructured and noisy in nature – to achieve better insights or to build better algorithms, it is necessary to play with clean data. For example, social media data is highly unstructured – it is an informal communication – typos, bad grammar, usage of slang, presence of unwanted content like URLs, Stopwords, Expressions etc. are the usual suspects.

In this blog, therefore I discuss about these possible noise elements and how you could clean them step by step. I am providing ways to clean data using Python.

As a typical business problem, assume you are interested in finding: which are the features of an iPhone which are more popular among the fans. You have extracted consumer opinions related to iPhone and here is a tweet you extracted:

“I luv my <3 iphone & you’re awsm apple. DisplayIsAwesome, sooo happppppy

http://www.apple.com”

Steps for data cleaning:

Here is what you do:

1. **Escaping HTML characters**: Data obtained from web usually contains a lot of html entities like < > & which gets embedded in the original data. It is

thus necessary to get rid of these entities. One approach is to directly remove them by the use of specific regular expressions. Another approach is to use appropriate packages and modules (for example htmlparser of Python), which can convert these entities to standard html tags. For example: < is converted to "<" and & is converted to "&".

Snippet:

```
          import HTMLParser

html_parser = HTMLParser.HTMLParser()

tweet = html_parser.unescape(original_tweet)
```

Output:

>> "I luv my <3 iphone & you're awsm apple. DisplayIsAwesome, sooo happppppy

http://www.apple.com”

2. **Decoding data**: Thisis the process of transforming information from complex symbols to simple and easier to understand characters. Text data may be subject to different forms of decoding like “Latin”, “UTF8” etc. Therefore, for better analysis, it is necessary to keep the complete data in standard encoding format. UTF-8 encoding is widely accepted and is recommended to use.

Snippet:

```
tweet = original_tweet.decode("utf8").encode('ascii','ignore')
```

Output:

>> “I luv my <3 iphone & you’re awsm apple. DisplayIsAwesome, sooo happppppy

http://www.apple.com”

3. **Apostrophe Lookup**: To avoid any word sense disambiguation in text, it is recommended to maintain proper structure in it and to abide by the rules of context free grammar. When apostrophes are used, chances of disambiguation increases.

For example “it’s is a contraction for it is or it has”.

All the apostrophes should be converted into standard lexicons. One can use a lookup table of all possible keys to get rid of disambiguates.

Snippet:

```
APPOSTOPHES = {“'s" : " is", "'re" : " are", ...} ## Need a huge dictionary
```

```
words = tweet.split()

reformed = [APPOSTOPHES[word] if word in APPOSTOPHES else word for word in words]

reformed = " ".join(reformed)
```

Outcome:

>> “I luv my <3 iphone & you are awsm apple. DisplayIsAwesome, sooo happppppy

http://www.apple.com”

4. **Removal of Stop-words**: When data analysis needs to be data driven at the word level, the commonly occurring words (stop-words) should be removed. One can either create a long list of stop-words or one can use predefined language specific libraries.

5. **Removal of Punctuations**: All the punctuation marks according to the priorities should be dealt with. For example: “.”, “,””?” are important punctuations that should be retained while others need to be removed.
6. **Removal of Expressions**: Textual data (usually speech transcripts) may contain human expressions like [laughing], [Crying], [Audience paused]. These expressions are usually non relevant to content of the speech and hence need to be removed. Simple regular expression can be useful in this case.

7. **Split Attached Words**: We humans in the social forums generate text data, which is completely informal in nature. Most of the tweets are accompanied with multiple attached words like RainyDay, PlayingInTheCold etc. These entities can be split into their normal forms using simple rules and regex.

Snippet:

```
cleaned = “ ”.join(re.findall(‘[A-Z][^A-Z]*’, original_tweet))
```

Outcome:

>> “I luv my <3 iphone & you are awsm apple. Display Is Awesome, sooo happppppy

http://www.apple.com”

8. **Slangs lookup**: Again, social media comprises of a majority of slang words. These words should be transformed into standard words to make free text. The words like luv will be converted to love, Helo to Hello. The similar approach of apostrophe look up can be used to convert slangs to standard words. A number of sources are available on the web, which provides lists of all possible slangs, this would be your holy grail and you could use them as lookup dictionaries for conversion purposes.

Snippet:

```
tweet = _slang_loopup(tweet)
```

Outcome:

>> “I love my <3 iphone & you are awesome apple. Display Is Awesome, sooo happppppy

http://www.apple.com”

9. **Standardizing words**: Sometimes words are not in proper formats. For example: “I looooveee you” should be “I love you”. Simple rules and regular expressions can help solve these cases.

Snippet:

```
tweet = ''.join(''.join(s)[:2] for _, s in itertools.groupby(tweet))
```

Outcome:

>> “I love my <3 iphone & you are awesome apple. Display Is Awesome, so happy

http://www.apple.com”

10. **Removal of URLs**: URLs and hyperlinks in text data like comments, reviews, and tweets should be removed.

Final cleaned tweet:

>> “I love my iphone & you are awesome apple. Display Is Awesome, so happy!” , <3 ,

Advanced data cleaning:

1. **Grammar checking**: Grammar checking is majorly learning based, huge amount of proper text data is learned and models are created for the purpose of grammar correction. There are many online tools that are available for grammar correction purposes.
2. **Spelling correction**: In natural language, misspelled errors are encountered. Companies like Google and Microsoft have achieved a decent accuracy level in automated spell correction. One can use algorithms like the Levenshtein Distances, Dictionary Lookup etc. or other modules and packages to fix these errors.

End Notes:

Hope you found this article helpful. These were some tips and tricks, I have learnt while working with a lot of text data. If you follow the above steps to clean the data, you can drastically improve the accuracy of your results and draw better insights. Do share your views/doubts in the comments section and I would be happy to participate.

Go Hack

If you like what you just read & want to continue your analytics learning, subscribe to our emails, follow us on twitter or like our facebook page.