ReadMe_Word Frequency Analysis

Subject

Data Science Assignment for
M.Sc. Information Studies - Data Science

Author

Rodion Burden
https://github.com/RodenBrudon/UvA_InformationStudies_demo
Date: 15.4.2020

Technology used

- Html: standard markup language
- JavaScript (ES6+): ECMAScript based scripting language with prototype-based object-orientation.
- PlotLy: Moder Analytics library. Suitable for JS and Python.

Content

Code Description

(app.js, Appendix 1)
The code is designed as an html parser that extracts (html-)content from a webpage and processes it according to the required the specifications.
The created class 'TextAnalyzer' consists of three parts, based on the role within the application.

1. Handle Data
   - Get data from the document object model (DOM)
   - Client the text
   - Store the texts
   - Calculate word frequencies
2. Handle Wordquery
   - Check presence of the word
   - Create hash table as response to the query
3. Create Histogram
   - Prepare data
   - Create histogram with PlotLy

Output Samples

ADD 2: HANDLE WORD QUERY

Instantiate the class

```
const analysis = new TextAnalyzer();
```

Query the word 'the'

```
analysis.analyze('the');
```

Result in the browser:

Analysis

The word "the" occurs in article 1: 94 times,
in article 2: 12 times and
in article 3: 65 times. In total it occurs 171.

ADD 3: CREATE HISTOGRAM

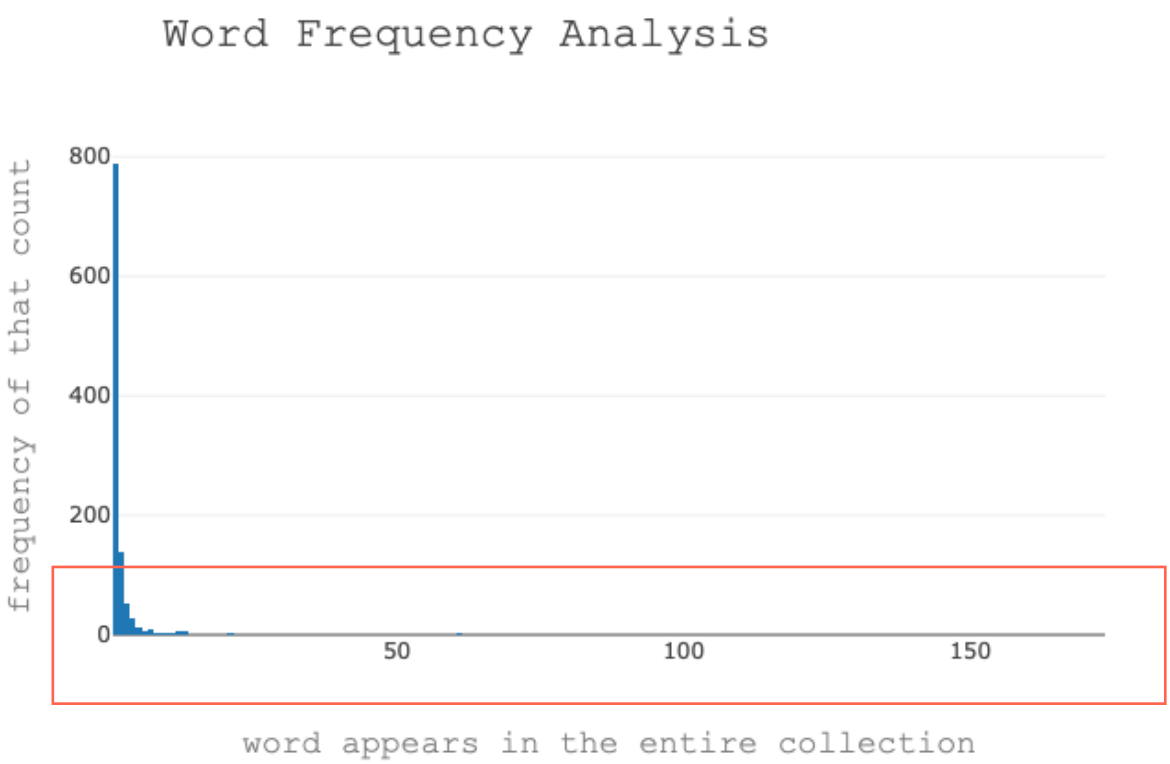
Prepare data

```
analysis.allTexts(DOM.plainText);
```

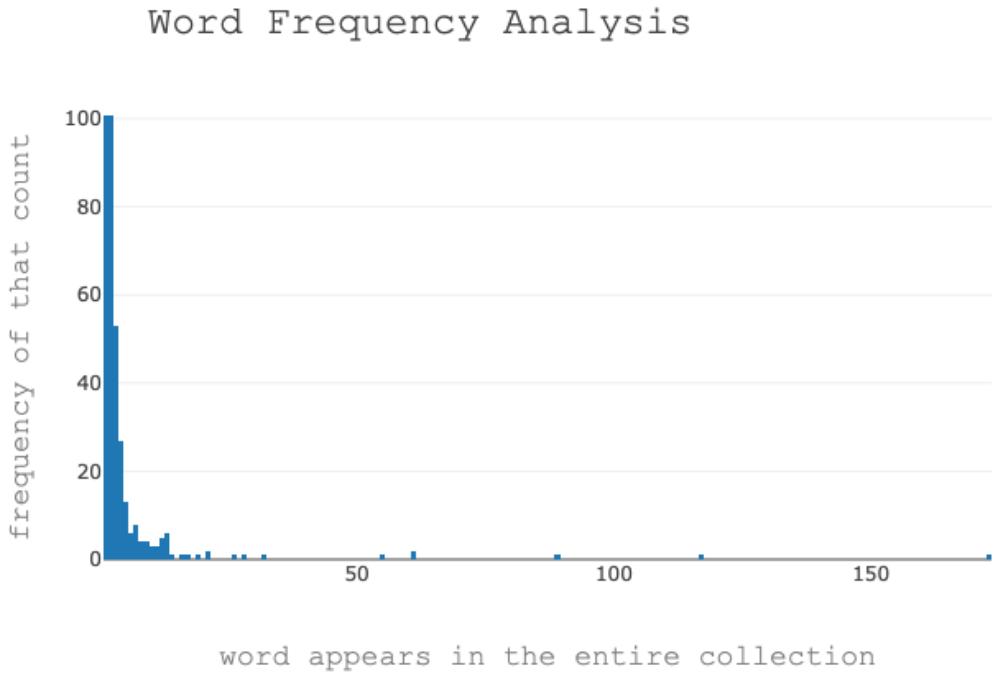
Call the method to create the histogram

```
analysis.renderHistogram(DOM.histogram);
```

Result in the browser:



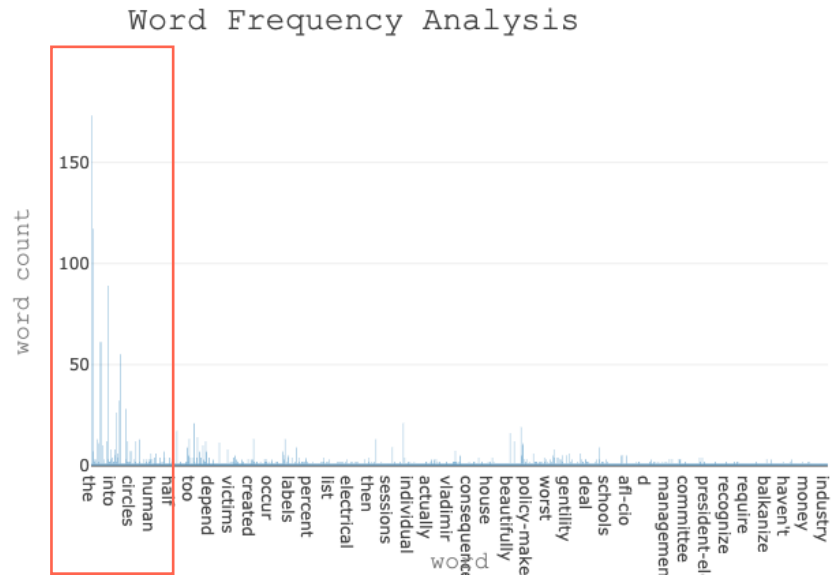
Zoom



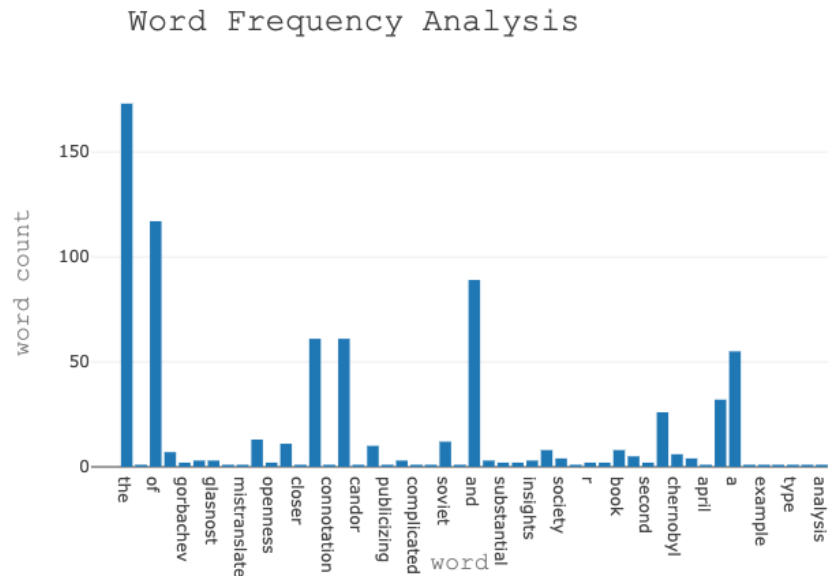
Interpretation
“The histogram illustrates well Zipf’s law, stating that the frequency of any word is inversely proportional to its rank in the frequency table. Thus the most frequent word will occur approximately twice as often as the second most frequent word, three times as often as the third most frequent word, etc.” The second histogram provides a deeper look and illustrates further implications of Zipf’s law.

```
analysis.renderHistogram(DOM.histogram2);
```

Result in the browser:



Zoom



“In the Brown Corpus of American English text, the word 'the' is the most frequently occurring word, and by itself accounts for nearly 7% of all word occurrences (69,971 out of slightly over 1 million). True to Zipf's Law, the second-place word ‘of’ accounts for slightly over 3.5% of words (36,411 occurrences), followed by ‘and’ (28,852)."
The analysis of the given texts are in line with the observed frequency of the three most common word ‘the, of, and’ in the Brown Corpus of American English. (Source: https://en.wikipedia.org/wiki/Zipf%27s_law)

Appendix 1

```
DOM = {
    analysis: document.querySelector('.output1'),
    plainText: document.querySelector('.output2'),
    histogram: document.querySelector('.output3'),
    histogram2: document.querySelector('.output4')

};

class TextAnalyzer {

    texts = {};
    wordFreqsPerArticle = {};
    wordQueryOutput = [];

    allTextsInOne = {};
    wordFreqsAllArticles = {};


    // ***** Handle Data
    //get articles from DOM
    getArticles = () => {
        this.articles = [...document.getElementsByTagName("text")];
    };

    // clean text in article
    cleanText = (s) => {
        // Normalize
        s = s.toLowerCase();
        // Strip quotes and brackets
        s = s.replace(/["“”(\[{}\])]|\B['‘]([^'’]+)['’]/g, '$1');
        // Strip dashes and ellipses
        s = s.replace(/[————…]|--|\.\.\./g, ' ');
        // Strip punctuation marks
        s = s.replace(/[!?;:.,]\B/g, '');
        // remove html tags [<p>, </p>]
        s = s.replace(/<p>/g, '');
        s = s.replace(/<\/p>/g, '');
        return s;
    };

    getWordsfromArticle = (s, i) => {
        s = this.cleanText(s);
        this.texts[i] = s;
    };

    // store texts of every article
    storeWordsfromArticles = (articlesObject) => {
        Object.keys(articlesObject).forEach(articleKey => {
            this.getWordsfromArticle(articlesObject[articleKey].innerHTML, parseInt(articleKey)+1)
        });
    };

```

```
53      // calc word frequency in individual article text
54      calcWordFreq = (s, i, storeFreqs) => {
55          storeFreqs[i] = s.match(/\S+/g).reduce(function(
    oFreq, sWord) {
56            if (oFreq.hasOwnProperty(sWord)) ++oFreq[sWord];
57            else oFreq[sWord] = 1;
58            return oFreq;
59          }, {});
60      };
61
62      //store word frequency of every article
63      storeFreqsfromTexts = (textsObject, storeFreqs) => {
64          Object.keys(textsObject).forEach(textKey => {
65              this.calcWordFreq(textsObject[textKey],
    textKey, storeFreqs);
66          })
67      };
68
69      // ***** Handle word query. Example: [the] -> [1, 20
    ] -> [2, 34] -> [3, 12]
70      // check for presence of a word in an object
71      isWordInSet = (word, set) => {
72          return (word in set);
73      };
74
75      // create output arrays showing location and frequency
     of the word
76      showIfWordInSet = (word, article, wordset) => {
77          const isIncl = this.isWordInSet(word, article);
78          if(isIncl){
79              let arr = [wordset, article[`${word}`]];
80              this.wordQueryOutput.push(arr);
81              }
82          };
83
84      // create hashtable as response to word query
85      showWordInSets = (word, sets) => {
86          this.wordQueryOutput.push(word);
87          Object.keys(sets).forEach(wordSet =>{
88              this.showIfWordInSet(word, sets[wordSet],
    wordSet);
89          })
90      };
91
92      // render the result of the word query to the DOM
93      renderOutputToDom = (output) => {
94          const element =
95          `<p>
96          The word "${output[0]}" occurs in article ${output
    [1][0]}: ${output[1][1]} times, </br>
97          in article ${output[2][0]}: ${output[2][1]} times
    and </br>
98          in article ${output[3][0]}: ${output[3][1]} times
    .
99          In total it occurs ${output[1][1]+output[2][1]+
```

```
 99 output[3][1]}.
100         </p>`
101         DOM.analysis.innerHTML = element;
102     };
103 
104     // In an improved version of the code the following
    method would be redesigned
105     // to a computationally more efficient method, so the
    data handling of
106     // all the articles doesnt run for every word query.
    ie I would build in
107     // the parameters (constructor) into the class and run
     the data handling
108     // at the instantiation  of the class.
109     analyze = (word) => {
110         this.getArticles();
111         this.storeWordsfromArticles(this.articles);
112         this.storeFreqsfromTexts(this.texts, this.
    wordFreqsPerArticle);
113         this.showWordInSets(word, this.wordFreqsPerArticle
    );
114         this.renderOutputToDom(this.wordQueryOutput);
115     };
116 
117     // ***** create histogram
118     // plain texts
119     allTexts = (out) => {
120         const keys = Object.keys(this.articles);
121         keys.forEach(key => {
122             out.appendChild(this.articles[key]);
123             });
124         out.innerHTML = this.cleanText(out.innerHTML);
125         this.allTextsInOne = this.cleanText(out.innerHTML)
    ;
126         //remove remaining html tags and numbers
127         this.allTextsInOne = this.allTextsInOne.replace(/<
    text>/g, '');
128         this.allTextsInOne = this.allTextsInOne.replace(
    /<\/text>/g, '');
129         this.allTextsInOne = this.allTextsInOne.replace(/[
    0-9]/g, '');
130         this.calcAllFreq(this.allTextsInOne);
131 
132     };
133 
134     calcAllFreq = (s) => {
135         this.wordFreqsAllArticles = s.match(/\S+/g).reduce
    (function(oFreq, sWord) {
136                   if (oFreq.hasOwnProperty(sWord)) ++oFreq
    [sWord];
137                   else oFreq[sWord] = 1;
138                   return oFreq;
139                 }, {});
140     };
141 
```

```
142     renderHistogram = (DOMhistogram) => {
143         // transform object to array
144         const result1 = Object.keys(this.
    wordFreqsAllArticles).map((key) => {
145             return [this.wordFreqsAllArticles[key]];
146         });
147 
148 
149         // create histogram
150         const arrRes = result1.map(el => {
151             return el[0];
152         });
153         const x = arrRes;
154         const trace = {
155             x: x,
156             type: 'histogram',
157 
158         };
159         const data = [trace];
160         const layout = {
161             title: {
162                 text:'Word Frequency Analysis',
163                 font: {
164                     family: 'Courier New, monospace',
165                     size: 24
166                 },
167                 xref: 'paper',
168                 x: 0.05,
169             },
170             xaxis: {
171                 title: {
172                     text: 'word appears in the entire
    collection',
173                         font: {
174                         family: 'Courier New, monospace',
175                             size: 18,
176                             color: '#7f7f7f'
177                     }
178                 },
179             },
180             yaxis: {
181                 title: {
182                     text: 'frequency of that count',
183                         font: {
184                         family: 'Courier New, monospace',
185                             size: 18,
186                             color: '#7f7f7f'
187                     }
188                 }
189             }
190         };
191 
192         Plotly.newPlot(DOMhistogram, data, layout);
193         //* Second histogram
194         const wordArr = this.allTextsInOne.match(/\S+/g);
```

```
195         const x2 = wordArr;
196         // -- this shows the actual words on the X axis of
     the histogram
197         const trace2 = {
198             x: x2,
199             type: 'histogram',
200 
201         };
202         const data2 = [trace2];
203             const layout2 = {
204                 title: {
205                     text:'Word Frequency Analysis',
206                     font: {
207                         family: 'Courier New, monospace',
208                         size: 24
209                     },
210                     xref: 'paper',
211                     x: 0.05,
212                 },
213                 xaxis: {
214                     title: {
215                         text: 'word',
216                         font: {
217                             family: 'Courier New,
    monospace',
218                             size: 18,
219                             color: '#7f7f7f'
220                         }
221                     },
222                 },
223                 yaxis: {
224                     title: {
225                         text: 'word count',
226                         font: {
227                             family: 'Courier New,
    monospace',
228                             size: 18,
229                             color: '#7f7f7f'
230                         }
231                     }
232                 }
233             };
234         Plotly.newPlot(DOMhistogram, data2, layout2);
235 
236     }
237 
238 }
239 
240 const analysis = new TextAnalyzer();
241 analysis.analyze('the');
242 analysis.allTexts(DOM.plainText);
243 analysis.renderHistogram(DOM.histogram);
244 analysis.renderHistogram(DOM.histogram2);
245 
246 
```