Sentence split algorithm

from Michael Shustov, 2021

This LiveScript is written to explain the algorithm of splitting the line without spaces to words using the dictionary.

Dictionary which is used here contains 400K+ English words and was taken from https://github.com/dwyl/english-words

The sentence is generated using 500 random words of the dictionary

```
warning off
%% Import data from text file with a dictionary

%% Set up the Import Options and import the data
opts = delimitedTextImportOptions("NumVariables", 1);

% Specify range and delimiter
opts.DataLines = [1, Inf];
opts.Delimiter = ",";

% Specify column names and types
opts.VariableNames = "the";
opts.VariableTypes = "string";

% Specify file level properties
opts.ExtraColumnsRule = "ignore";
opts.EmptyLineRule = "read";

% Specify variable properties
opts = setvaropts(opts, "the", "WhitespaceRule", "preserve");
opts = setvaropts(opts, "the", "EmptyFieldRule", "auto");

% Import the data
Dict = readtable("C:\Users\shust\YandexDisk\#Propgramming\SentenceSplittingAlg\425K-english-wor
Dict = table2array(Dict)';

%% Clear temporary variables
clear opts
```

```
%%This section builds the random sentence using words from a dictionary

%number of words in the sentence
N = 500;

%This vector shows order of words from the dictionary in the sentence, our
%final result should match this vector in an ideal case, or if we have
%multiple solutions, one of them should be equal to this vector
SentenceVector = [];
SentenceVector = uint32(round(size(Dict,2)*rand(1,N)));
SentenceVector(SentenceVector==0) = 1;
```

```
%Lets build the sentence according to the SentenceVector
SentenceStr = "";
for i=1:size(SentenceVector,2)
    SentenceStr = strcat(SentenceStr,Dict(SentenceVector(i)));
end

%This is our sentence
SentenceStr
```

```
SentenceStr =
"sapphirinewaddlersquinquelobatederepsinuphoardsunamoLatviiaroadway'snon-naturalwrentailhub-turningtoothflowertriole
```

```
%This section is the main splitting algorithm

%run the stopwatch
tic


%The first step - find positions of words from the dictionary in our
%sentence. This can be done using the "suffix tree" of the sentence, but
%Matlab has no internal function to build the tree and I do not want to
%program it now, so we will use internal Matlab functions to find substrings in a
%basic string
%Index in DectPos is the number of the word in our dictionary, elements of
%DictPos- vectors with possible positions of these words in the non-space-Sentence
for i = 1:size(Dict,2)
   DictPos{i}= strfind(SentenceStr,Dict(i));
end

%lets see the result. As expected, most of words are not found in the
%Sentence, but this happens in our task only (natural language, big
%dictionary), in a common case the picture may differ and we can
%potentially find many probable positions of the words from the dictionary
DictPos
```

```
DictPos = 1×466550 cell
```

...

	1	2	3	4	5	6	7	8
1	[]	[]	[]	[]	[]	[]	[]	[]

```
%check the stopwatch
toc
```

```
Elapsed time is 6.732781 seconds.
```

```
%Now we need to build a matrix of 0 and 1. Number of rows = number of letters in the
%Sentence, number of cols = number of words in the dictionary.
%For example we have a sentence "mymanywordsmy". Our dictionary = ["a" "many"
%"words" "m" "my" ]
%So we build the Matrix in the next way:
```

```
%Search the first word from a dictionary "a", it occupies the 4th
%position in the sentence. So the first column in Matr will be [0 0 0 1 0 0 0 0 0 0 0 0 0]
%
% The word "m" stays in 3 different positions in the Sentence: 1, 3 and 12th, so it gives us
% 3 columns in the Matr:
%[1 0 0 0 0 0 0 0 ....]
%[0 0 1 0 0 0 0 0 ....]
%[0 0 0 0 0 0 0 0 0 0 0 1 0]
%
%The word "my" stays in 2 places and each time it occupies 2 positions
%(because it has 2 letters), so columns of the Matr will be:
%[1 1 0 0 0 0 0 0 0 0 0....]
%[0 0 0 ..............  1 1]
%
%So we build this matrix for all the found words for all their probable
%positions

Matr = [];
k = 1;
for i = 1:size(Dict,2)

   %length of i-th word in the dictionary
   curLength = strlength(Dict(i));

   if size(DictPos{i},2) > 0
       for j = 1:size(DictPos{i},2)

          %this is our positions matrix
          Matr(DictPos{i}(j):DictPos{i}(j)+curLength-1,k) = 1;

          %this is the array who's 1st col contains the number of word in
          %the dictionary and the second col contains related start
          %position of this word in the sentence. We need this array to
          %match our final result with dictionary words and their order int
          %the sentence
          WordsNum(k,1) = i;
          WordsNum(k,2) = DictPos{i}(j);
          k = k + 1;

       end
   end

end

%Good! Now we need to solve a system of linear algebraic equations
% Matr*sol1=vect1.
%Here vect1 - col-vector of ones with a length of original
%sentence. It means that all positions in the original sentence are
%occupied with letters
% sol1 - col-vector which we are searching for.
% i.e. our sentence is a linear combination of all words from a dictionary
% in all their probable positions with coefficients in sol1

vect1 = [];
```

```
vect1(1:strlength(SentenceStr)) = 1;
vect1 = vect1';   %transpose of a row-vector to a column-vector

%and the most satisfying moment of solving the system.....
sol1 = linsolve(Matr,vect1);
%the obtained solution (with a linsolve) is not precize, but our
%coefficients can be only integer 0 or 1, so we round the obtained
%solution.
%Matlab can also give us -1 in a solution, but because of some features of
%the language we may not filter them, because they do not influence much on
%the solution result (they give 1-2 wrong words in the dictionary)
sol1 = round(sol1);

%sort the words in the dictionary of probable positions
res = sortrows(WordsNum(sol1==1,:),2);

% build the result array of words in the correct order
resStr = string();
for i = 1:size(res,1)
    %!!!!!!!!this is the final array of words from the dictionary in a correct order
    resStr(i) = Dict(res(i,1));
end

%lets connect the result array into one string without spaces to compare with an initial
%sentence
ResSentenceStr = "";
for i=1:size(resStr,2)
    ResSentenceStr = strcat(ResSentenceStr,resStr(i));
end

%compare 2 strings (initial sentence and result). If the result of
% compareson == 1 strings are absolutely equal, if result is 0, that means
% some words are defined in a bit different way
strcmp(ResSentenceStr,SentenceStr)
```

```
ans = logical
   0
```

```
toc  %elapsed time
```

```
Elapsed time is 33.248689 seconds.
```

```
%here we check the mismatch of the result array of words and initial(real)
%array of random words
R{1}=convertStringsToChars(SentenceStr);
R{2}=convertStringsToChars(ResSentenceStr);
for i=1:min(size(R{1},2), size(R{2},2))
    ii(i) = R{1}(i)~=R{2}(i);
end
min(find(ii)) %show the first index where these two arrays do not match
```

```
ans = 1797
```

Resume:

The calculated word sequence and an initial one (in the non-space-sentence) may differ a bit, this is caused by ambiguity of words recognition (some words like "wine"+"yard" and "wineyard"). But seems, that most of the sequence is recognized well. The ambiguity depends both on the dictionary and non-space-sentence.

The problem of ambiguity in SLE solution was not investigated here, but tests on the natural language showed, that it is not critical. It may give problems when the dictionary contains close words (smaller Levenshtein's distance), for example, when our alphabet contains few letters "a","g","t","c" and the sentence is big.

```
%Original string and splitted words are saved to files

fid = fopen('originalSentence.txt','wt');
fprintf(fid, SentenceStr);
fclose(fid);

fid = fopen('SplitResult.txt','wt');
for i = 1:size(resStr,2)
   fprintf(fid,'%s\n', resStr(i));
end
fclose(fid);
```