Twitter cloud

You can find people of all ages on Twitter. Moreover, those short messages make sure it is very accessible, making it easy for people to post messages (or tweets in the jargon). This makes Twitter ideal to follow trends in the population. To help researchers Twitter offers an API. This is a way to view the website without having to use a browser and obtain the data in a structured way, so that they can be processed more easily by a computer program.

To examine trends for example you can set up a frequency table of the words used. If you do this with the tweets that you get by searching for a particular term, then you expect that term to have a high frequency and that related words occur often too.

A classic way to present a frequency table in a way that is visually interesting to the human eye, is a tag cloud. In a tag cloud, the words with a higher frequency are in a larger font, so they also get noticed more easily.

Above you see an example of a tag cloud. This tag cloud was built on the basis of the results for the search term 'tag cloud'. You immediately see here that such a frequency table can easily fill up with so-called stop words, i.e., words that are not directly related to the search term, but just always occur frequently in a given language.

Preparation

The Twitter Search API can be found at http://search.twitter.com/search.json?q=*zoekterm*, in which case *zoekterm* should be replaced by the specific word that you are looking for. The API returns the results in JSON-format (*JavaScript Object Notation*). In principle this format has got nothing to do with Python, but luckily it is written exactly the same as a nested *dictionary*. This means we can just interpret the string with the function eval. However, three little problems arise: in JavaScript None is written as null, True as true and False as false. If we create three variables, however, that carry the same name and corresponding values, then we can process such a result in one time. Below is an example of how this can be done. We assume that the variable result contains the text that is returned by the API.

```
null, true, false = None, True, False
dictionary = eval(result)
```

The dictionary that we have constructed above, contains a key 'results' with which a list corresponds. In this list there are dictionaries that each contain the rext of a tweet linked to a key 'text'.

Assignment

Write a function twittercloud that has an obligatory argument and two optional arguments. The obligatory argument is the search term. This search term cannot contain any spaces. The function returns a frequency table of the words in the tweets that are returned for the search term. A word is the longest possible succession of letters (accents included), numbers, the character # and the character @. The first optional argument has the name stopwords and contains a collection of words that should be ignored. If this argument is not given, then the collection with only the string 'RT' is used (i.e. the most popular abbreviation on Twitter to indicate a *retweet*). The second optional argument has the name number and contains the default value 20. This is the number of words that have to be incorporated in the eventual frequency table. The frequency table then of course contains the words with the highest frequency. If there are multiple words with the same amount, then the words are arranged alphabetically. All comparisons of words have to be case-sensitive.

Example

Because of the changeable content that is returned by the Twitter Search API, it will not be possible to use this example as a DocTest.

```
>>> twittercloud('geografie')
{
   'i': 39,
   'na': 48,
   'geografie': 82,
   'mi': 9,
   'nie': 17,
   'sie': 36,
   'in': 9,
   'mam': 12,
   'D': 8,
   'Geografie': 13,
   'a': 13,
   'ale': 9,
   'chemie': 7,
   'fizyke': 8,
   'historie': 12,
   'ja': 13,
   'to': 8,
   'uczyc': 9,
   'z': 14,
   'za': 8
}
```

Je vindt mensen van alle leeftijden op Twitter. Bovendien zorgen de korte berichten ervoor dat het zeer laagdrempelig is, waardoor mensen zeer vlot berichten (of *tweets* in het jargon) plaatsen. Dit maakt Twitter uitermate geschikt om trends in de bevolking op te volgen. Om

onderzoekers te helpen biedt Twitter hiervoor een API aan. Dit is een manier om de website te bekijken zonder een browser te gebruiken en om de gegevens in een gestructureerde manier terug te krijgen, zodat ze gemakkelijker verwerkt kunnen worden door een computerprogramma.

Om trends te onderzoeken kan je bijvoorbeeld een frequentietabel opstellen van de woorden die gebruikt worden. Als je dit doet met de tweets die je krijgt door te zoeken naar een bepaalde term, dan verwacht je dat die zoekterm een hoge frequentie heeft en dat gerelateerde woorden ook vaak voorkomen.

Een klassieke manier om een frequentietabel voor te stellen op een manier die visueel interessant is voor het menselijke oog, is een woordenwolk. In een woordenwolk krijgen de woorden met een hogere frequentie een groter lettertype, waardoor ze ook gemakkelijker opvallen.

Hierboven zie je een voorbeeld van zo'n woordenwolk. Deze woordenwolk werd opgebouwd aan de hand van de resultaten voor de zoekterm *'word cloud'*. Je ziet hier meteen ook dat zo'n frequentietabel gemakkelijk kan vollopen met zogehete stopwoorden, i.e., woorden die niet direct gerelateerd zijn aan de zoekterm, maar gewoon altijd vaak voorkomen in een bepaalde taal.

Voorbereiding

De zoek-API van Twitter kan je bereiken via de url http://search.twitter.com/search.json?q=*zoekterm*, waarbij je *zoekterm* vervangt door het specifieke woord dat je wil opzoeken. De API geeft dan de resultaten terug in JSON-formaat (*JavaScript Object Notation*). In principe heeft dit formaat niets met Python te maken, maar gelukkig wordt dit op exact dezelfde manier genoteerd als een geneste *dictionary*. Dit wil zeggen dat we de string gewoon met de functie eval kunnen interpreteren. Hierbij duiken er echter drie kleine problemen op: in JavaScript wordt None genoteerd als null, True als true en False als false. Als we echter drie variabelen aanmaken die die namen hebben en de corresponderende waarden bevatten, dan kunnen we zo'n resultaat in één keer verwerken. Hieronder zie je een voorbeeld van hoe dit kan gedaan worden. We gaan er hier vanuit dat de variabele resultaat de tekst bevat die je terugkrijgt van de API.

```
null, true, false = None, True, False
dictionary = eval(resultaat)
```

De dictionary die we hierboven hebben geconstrueerd, bevat een sleutel 'results' waarmee een lijst correspondeert. In die lijst zitten dictionaries die elk de tekst van een tweet bevatten verbonden met de sleutel 'text'.

Opgave

Schrijf een functie twitterwolk die één verplicht argument heeft en twee optionele argumenten. Het verplichte argument is de zoekterm waarop gezocht wordt. Deze zoekterm mag geen spaties bevatten. De functie geeft als resultaat een frequentietabel terug van de woorden in de tweets die teruggegeven worden voor die zoekterm. Een woord is hierbij een zo lang mogelijke opeenvolging van letters (ook met accenten), cijfers, het karakter # en het karakter @. Het eerste optionele argument heeft de naam stopwoorden en bevat een verzameling met de woorden die genegeerd moeten worden. Als dit argument niet opgegeven wordt, dan wordt de verzameling gebruikt die enkel de string 'RT' bevat (i.e. de afkorting die op Twitter wordt gebruikt om een *retweet* aan te duiden). Het tweede optionele argument heeft de naam aantal en bevat standaard

de waarde 20. Dit is het aantal woorden dat opgenomen moet worden in de uiteindelijke frequentietabel. De frequentietabel bevat dan natuurlijk de woorden met de hoogste frequentie. Als er meerdere woorden zijn met hetzelfde aantal, dan wordt voorrang gegeven aan de woorden die alfabetisch eerst gerangschikt worden. Alle vergelijkingen van woorden moeten hoofdlettergevoelig gebeuren.

Voorbeeld

Door de veranderlijke inhoud die je terugkrijgt van de zoek-API van Twitter zal het niet mogelijk zijn om dit voorbeeld als DocTest te gebruiken.

```
>>> twitterwolk('geografie')
{
   'i': 39,
   'na': 48,
   'geografie': 82,
   'mi': 9,
   'nie': 17,
   'sie': 36,
   'in': 9,
   'mam': 12,
   'D': 8,
   'Geografie': 13,
   'a': 13,
   'ale': 9,
   'chemie': 7,
   'fizyke': 8,
   'historie': 12,
   'ja': 13,
   'to': 8,
   'uczyc': 9,
   'z': 14,
   'za': 8
}
```