10.1 Bag of Words and N-Grams

May 9, 2019

1 Chapter 10 Textual Data

You may not be used to thinking about *text*, like an e-mail or a newspaper article, as data. But just as we might want to predict the price of a home or group wines into similar types, we might want to predict the sender of an e-mail or group articles into similar types. To leverage the machine learning techniques we have already learned, we will need a way to convert raw text into tabular form. This chapter introduces some principles for doing this.

2 10.1 Bag of Words and N-Grams

In data science, a text is typically called a **document**, even though a document can be anything from a text message to a full-length novel. A collection of documents is called a **corpus**. In this chapter, we will work with a corpus of text messages, which contains both spam and non-spam ("ham") messages.

```
In [1]: import pandas as pd
        pd.options.display.max_rows = 10

        texts = pd.read_csv(
            "https://raw.githubusercontent.com/dlsun/data-science-book/master/data/SMSSpamColle
            sep="\t",
            names=["label", "text"]
        )
        texts
```

Out[1]:

	label	text
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...
...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will ü b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...

```
5571   ham                          Rofl. Its true to its name

[5572 rows x 2 columns]
```

We might, for example, want to train a classifier to predict whether or not a text message is spam. To use machine learning techniques like k-nearest neighbors, we have to transform each of these "documents" into a more regular representation.

A **bag of words** representation reduces a document to just the multiset of its words, ignoring grammar and word order. (A *multiset* is like a set, except elements are allowed to appear more than once.)

So, for example, the **bag of words** representation of the string "I am Sam. Sam I am." would be `{I, I, am, am, Sam, Sam}`. In Python, it is easiest to represent multisets using dictionaries, where the keys are the (unique) words and the values are the counts. So we would represent the above bag of words as `{"I": 2, "am": 2, "Sam": 2}`.

Let's convert the text messages to a bag of words representation. To do this, we will use the `Counter` object in the `collections` module of the Python standard library. First, let's see how the `Counter` works.

```
In [2]: from collections import Counter
        Counter(["I", "am", "Sam", "Sam", "I", "am"])

Out[2]: Counter({'I': 2, 'am': 2, 'Sam': 2})
```

It takes in a list and returns a dictionary of counts—in other words, the bag of words representation that we want. But to be able to use `Counter`, we have to first convert our text into a list of words. We can do this using the string methods in Pandas, such as `.str.split()`, which splits a string into a list based on some character (which, by default, is whitespace).

```
In [3]: texts["text"].str.split()

Out[3]: 0       [Go, until, jurong, point,, crazy.., Available...
        1                     [Ok, lar..., Joking, wif, u, oni...]
        2       [Free, entry, in, 2, a, wkly, comp, to, win, F...
        3       [U, dun, say, so, early, hor..., U, c, already...
        4       [Nah, I, don't, think, he, goes, to, usf,, he,...
                                      ...
        5567    [This, is, the, 2nd, time, we, have, tried, 2,...
        5568        [Will, ü, b, going, to, esplanade, fr, home?]
        5569    [Pity,, *, was, in, mood, for, that., So...any...
        5570    [The, guy, did, some, bitching, but, I, acted,...
        5571                     [Rofl., Its, true, to, its, name]
        Name: text, Length: 5572, dtype: object
```

There are several problems with this approach:

- **It is case-sensitive.** The word "the" in message 5567 and the word "The" in message 5570 are technically different strings and will be treated as different words by the `Counter`.
- **There is punctuation.** For example, in message 0, one of the words is "point,". This will be treated differently from the word "point".

We can **normalize** the text for case by

- converting all of the characters to lowercase, using the `.str.lower()` method
- stripping punctuation using a regular expression. The regular expression `[^\w\s]` tells Python to look for any pattern that is not (`^`) either an alphanumeric character (`\w`) or whitespace (`\s`). That is, it will detect any occurrence of punctuation. We will then use the `.str.replace()` method to replace all detected occurrences with the empty string, effectively removing all punctuation from the string.

By chaining these commands together, we obtain a list, to which we can apply the `Counter` to obtain the bag of words representation.

```
In [4]: words = (
            texts["text"].
            str.lower().
            str.replace("[^\w\s]", "").
            str.split()
        )

        words
```

```
Out[4]: 0       [go, until, jurong, point, crazy, available, o...
        1                           [ok, lar, joking, wif, u, oni]
        2       [free, entry, in, 2, a, wkly, comp, to, win, f...
        3       [u, dun, say, so, early, hor, u, c, already, t...
        4       [nah, i, dont, think, he, goes, to, usf, he, l...
                                       ...
        5567    [this, is, the, 2nd, time, we, have, tried, 2,...
        5568         [will, ü, b, going, to, esplanade, fr, home]
        5569    [pity, was, in, mood, for, that, soany, other,...
        5570    [the, guy, did, some, bitching, but, i, acted,...
        5571                      [rofl, its, true, to, its, name]
        Name: text, Length: 5572, dtype: object
```

```
In [5]: words.apply(Counter)
```

```
Out[5]: 0       {'go': 1, 'until': 1, 'jurong': 1, 'point': 1,...
        1       {'ok': 1, 'lar': 1, 'joking': 1, 'wif': 1, 'u'...
        2       {'free': 1, 'entry': 2, 'in': 1, '2': 1, 'a': ...
        3       {'u': 2, 'dun': 1, 'say': 2, 'so': 1, 'early':...
        4       {'nah': 1, 'i': 1, 'dont': 1, 'think': 1, 'he'...
                                       ...
        5567    {'this': 1, 'is': 2, 'the': 2, '2nd': 1, 'time...
        5568    {'will': 1, 'ü': 1, 'b': 1, 'going': 1, 'to': ...
        5569    {'pity': 1, 'was': 1, 'in': 1, 'mood': 1, 'for...
        5570    {'the': 1, 'guy': 1, 'did': 1, 'some': 1, 'bit...
        5571    {'rofl': 1, 'its': 2, 'true': 1, 'to': 1, 'nam...
        Name: text, Length: 5572, dtype: object
```

2.1 N-Grams

The problem with the bag of words representation is that the ordering of the words is lost. For example, the following sentences have the exact same bag of words representation, but convey different meanings:

1. The dog bit her owner.
2. Her dog bit the owner.

The first sentence has only two actors (the dog and its owner), but the second sentence has three (a woman, her dog, and the owner of something). To better capture the *semantic* meaning of these two documents, we can use **bigrams** instead of individual words. A **bigram** is simply a pair of consecutive words. The "bag of bigrams" of the two sentences above are quite different:

1. {"The dog", "dog bit", "bit her", "her owner"}
2. {"Her dog", "dog bit", "bit the", "the owner"}

They only share 1 bigram (out of 4) in common, even though they share the same 5 words.

Let's get the bag of bigrams representation for the words above. To generate the bigrams from the list of words, we will use the `zip` function in Python, which takes in two lists and returns a single list of pairs (consisting of one element from each list):

```
In [6]: list(zip([1, 2, 3], [4, 5, 6]))

Out[6]: [(1, 4), (2, 5), (3, 6)]

In [7]: def get_bigrams(words):
            # We need to line up the words as follows:
            #   words[0], words[1]
            #   words[1], words[2]
            #       ... ,  ...
            # words[n-1], words[n]
            return zip(words[:-1], words[1:])

        words.apply(get_bigrams).apply(Counter)

Out[7]: 0       {('go', 'until'): 1, ('until', 'jurong'): 1, (...
        1       {('ok', 'lar'): 1, ('lar', 'joking'): 1, ('jok...
        2       {('free', 'entry'): 1, ('entry', 'in'): 1, ('i...
        3       {('u', 'dun'): 1, ('dun', 'say'): 1, ('say', '...
        4       {('nah', 'i'): 1, ('i', 'dont'): 1, ('dont', '...
                                    ...
        5567    {('this', 'is'): 1, ('is', 'the'): 1, ('the', ...
        5568    {('will', 'ü'): 1, ('ü', 'b'): 1, ('b', 'going...
        5569    {('pity', 'was'): 1, ('was', 'in'): 1, ('in', ...
        5570    {('the', 'guy'): 1, ('guy', 'did'): 1, ('did',...
        5571    {('rofl', 'its'): 1, ('its', 'true'): 1, ('tru...
        Name: text, Length: 5572, dtype: object
```

Instead of taking 2 words at a time, we could take 3, 4, or, in general, n words. A tuple of n consecutive words is called an n-gram, and we can convert any document to a "bag of n-grams" representation.

The larger n is, the better the representation will capture the meaning of a document. But if n is so large that hardly any n-gram occurs more than once, then we will not learn much from this representation.

3 Exercises

Exercise 1. Read in the OKCupid data set (`/data301/data/okcupid/profiles.csv`). Convert the users' responses to `essay0` ("self summary") into a bag of words representation.

(*Hint:* Test your code on the first 100 users before testing it on the entire data set.)

```
In [8]: profiles = pd.read_csv("/data301/data/okcupid/profiles.csv")
        profiles.head()
```

Out[8]:

	age	body_type	diet	drinks	drugs	\
0	22	a little extra	strictly anything	socially	never	
1	35	average	mostly other	often	sometimes	
2	38	thin	anything	socially	NaN	
3	23	thin	vegetarian	socially	NaN	
4	29	athletic	NaN	socially	never	

	education	\
0	working on college/university	
1	working on space camp	
2	graduated from masters program	
3	working on college/university	
4	graduated from college/university	

	essay0	\
0	about me: \n \ni would love to think...	
1	i am a chef: this is what that means. \n1...	
2	i'm not ashamed of much, but writing public te...	
3	i work in a library and go to school. . .	
4	hey how's it going? currently vague on the pro...	

	essay1	\
0	currently working as an international agent fo...	
1	dedicating everyday to being an unbelievable b...	
2	i make nerdy software for musicians, artists, ...	
3	reading things written by old dead people	
4	work work work work + play	

	essay2	\
0	making people laugh. \nranting about a go...	
1	being silly. having ridiculous amonts of fun w...	
2	improvising in different contexts. alternating...	

```
3  playing synthesizers and organizing books acco...
4  creating imagery to look at:<br />\nhttp://bag...

                                                essay3    ...      \
0  the way i look. i am a six foot half asian, ha...    ...
1                                                NaN    ...
2  my large jaw and large glasses are the physica...    ...
3                  socially awkward but i do my best    ...
4            i smile a lot and my inquisitive nature    ...

                          location  \
0  south san francisco, california
1              oakland, california
2        san francisco, california
3             berkeley, california
4        san francisco, california

                                      offspring orientation  \
0  doesn&rsquo;t have kids, but might want them    straight
1  doesn&rsquo;t have kids, but might want them    straight
2                                           NaN    straight
3                       doesn&rsquo;t want kids    straight
4                                           NaN    straight

                         pets                                    religion sex  \
0  likes dogs and likes cats     agnosticism and very serious about it   m
1  likes dogs and likes cats  agnosticism but not too serious about it   m
2                   has cats                                        NaN   m
3                 likes cats                                        NaN   m
4  likes dogs and likes cats                                        NaN   m

                                 sign     smokes  \
0                              gemini  sometimes
1                              cancer         no
2  pisces but it doesn&rsquo;t matter         no
3                              pisces         no
4                            aquarius         no

                                            speaks     status
0                                          english     single
1  english (fluently), spanish (poorly), french (...     single
2                            english, french, c++  available
3                          english, german (poorly)     single
4                                          english     single

[5 rows x 31 columns]
```

In [9]:
```
essay0_words = (
```

```
        profiles["essay0"].
        str.lower().
        str.replace("<br />", "").
        str.replace("[^\w\s]", "").
        str.split()
    )

    essay0_words = essay0_words.fillna("")
```

```
In [10]: essay0_words.apply(Counter)
```

```
Out[10]: 0        {'about': 4, 'me': 5, 'i': 8, 'would': 2, 'lov...
         1        {'i': 13, 'am': 7, 'a': 5, 'chef': 1, 'this': ...
         2        {'im': 3, 'not': 1, 'ashamed': 1, 'of': 10, 'm...
         3        {'i': 1, 'work': 1, 'in': 1, 'a': 1, 'library'...
         4        {'hey': 1, 'hows': 1, 'it': 1, 'going': 1, 'cu...
                                    ...
         59941    {'vibrant': 1, 'expressive': 1, 'caring': 1, '...
         59942    {'im': 4, 'nick': 1, 'i': 4, 'never': 2, 'know...
         59943    {'hello': 1, 'i': 9, 'enjoy': 2, 'traveling': ...
         59944    {'all': 1, 'i': 2, 'have': 1, 'in': 1, 'this':...
         59945    {'is': 2, 'it': 1, 'odd': 1, 'that': 3, 'havin...
         Name: essay0, Length: 59946, dtype: object
```

Exercise 2. The text of *Green Eggs and Ham* by Dr. Seuss can be found in (https://raw.githubusercontent.com/dlsun/data-science-book/master/data/drseuss/greeneggsandham.
Read in this file and convert this "document" into a bag of trigrams (3-grams) representation.
Which trigram appears most often? Some code has been provided to get you started.

```
In [13]: # TYPE YOUR CODE HERE.
         import urllib.request
         import re
         text = urllib.request.urlopen("https://raw.githubusercontent.com/dlsun/data-science-b

         words = []
         for line in text:
             words.extend(
                 re.sub(r'[^-\w\s]', '', line.decode()).
                 lower().
                 split()
             )

         trigrams = list(zip(words[:-2], words[1:-1], words[:-1]))
         Counter(trigrams).most_common()
```

```
Out[13]: [(('not', 'like', 'not'), 34),
          (('like', 'them', 'like'), 34),
          (('i', 'do', 'i'), 33),
          (('do', 'not', 'do'), 32),
```

```
(('in', 'a', 'in'), 30),
(('eat', 'them', 'eat'), 23),
(('with', 'a', 'with'), 19),
(('not', 'in', 'not'), 19),
(('i', 'will', 'i'), 17),
(('them', 'in', 'them'), 14),
(('i', 'would', 'i'), 13),
(('would', 'not', 'would'), 13),
(('would', 'you', 'would'), 11),
(('them', 'with', 'them'), 11),
(('not', 'eat', 'not'), 11),
(('in', 'the', 'in'), 11),
(('green', 'eggs', 'green'), 10),
(('eggs', 'and', 'eggs'), 10),
(('and', 'ham', 'and'), 10),
(('them', 'here', 'them'), 10),
(('will', 'not', 'will'), 10),
(('like', 'green', 'like'), 9),
(('them', 'sam-i-am', 'them'), 9),
(('a', 'train', 'a'), 9),
(('ham', 'i', 'ham'), 8),
(('here', 'or', 'here'), 8),
(('or', 'there', 'or'), 8),
(('there', 'i', 'there'), 8),
(('them', 'anywhere', 'them'), 8),
(('a', 'house', 'a'), 8),
(('a', 'mouse', 'a'), 8),
(('could', 'not', 'could'), 8),
(('anywhere', 'i', 'anywhere'), 7),
(('a', 'box', 'a'), 7),
(('a', 'fox', 'a'), 7),
(('not', 'with', 'not'), 7),
(('a', 'car', 'a'), 7),
(('a', 'tree', 'a'), 7),
(('on', 'a', 'on'), 7),
(('the', 'dark', 'the'), 7),
(('and', 'i', 'and'), 7),
(('will', 'eat', 'will'), 7),
(('mouse', 'i', 'mouse'), 6),
(('could', 'you', 'could'), 6),
(('not', 'could', 'not'), 6),
(('you', 'like', 'you'), 4),
(('house', 'i', 'house'), 4),
(('you', 'could', 'you'), 4),
(('you', 'may', 'you'), 4),
(('let', 'me', 'let'), 4),
(('me', 'be', 'me'), 4),
(('be', 'i', 'be'), 4),
```

```
(('fox', 'i', 'fox'), 4),
(('train', 'not', 'train'), 4),
(('the', 'rain', 'the'), 4),
(('a', 'goat', 'a'), 4),
(('try', 'them', 'try'), 4),
(('i', 'am', 'i'), 3),
(('that', 'sam-i-am', 'that'), 3),
(('sam-i-am', 'would', 'sam-i-am'), 3),
(('you', 'in', 'you'), 3),
(('car', 'you', 'car'), 3),
(('you', 'will', 'you'), 3),
(('tree', 'not', 'tree'), 3),
(('box', 'i', 'box'), 3),
(('train', 'a', 'train'), 3),
(('not', 'on', 'not'), 3),
(('a', 'boat', 'a'), 3),
(('and', 'in', 'and'), 3),
(('am', 'sam', 'am'), 2),
(('sam', 'i', 'sam'), 2),
(('sam-i-am', 'i', 'sam-i-am'), 2),
(('you', 'eat', 'you'), 2),
(('fox', 'not', 'fox'), 2),
(('box', 'not', 'box'), 2),
(('house', 'not', 'house'), 2),
(('they', 'are', 'they'), 2),
(('may', 'like', 'may'), 2),
(('them', 'you', 'them'), 2),
(('will', 'see', 'will'), 2),
(('tree', 'i', 'tree'), 2),
(('you', 'let', 'you'), 2),
(('you', 'would', 'you'), 2),
(('you', 'on', 'you'), 2),
(('i', 'could', 'i'), 2),
(('not', 'would', 'not'), 2),
(('dark', 'would', 'dark'), 2),
(('rain', 'i', 'rain'), 2),
(('dark', 'not', 'dark'), 2),
(('you', 'see', 'you'), 2),
(('you', 'do', 'you'), 2),
(('goat', 'i', 'goat'), 2),
(('boat', 'i', 'boat'), 2),
(('them', 'and', 'them'), 2),
(('and', 'you', 'and'), 2),
(('say', 'i', 'say'), 2),
(('i', 'like', 'i'), 2),
(('would', 'eat', 'would'), 2),
(('so', 'good', 'so'), 2),
(('thank', 'you', 'thank'), 2),
```

```
(('sam', 'sam', 'sam'), 1),
(('am', 'that', 'am'), 1),
(('sam-i-am', 'that', 'sam-i-am'), 1),
(('like', 'that', 'like'), 1),
(('sam-i-am', 'do', 'sam-i-am'), 1),
(('do', 'you', 'do'), 1),
(('ham', 'would', 'ham'), 1),
(('house', 'would', 'house'), 1),
(('box', 'would', 'box'), 1),
(('eat', 'green', 'eat'), 1),
(('car', 'eat', 'car'), 1),
(('them', 'eat', 'them'), 1),
(('here', 'they', 'here'), 1),
(('are', 'i', 'are'), 1),
(('see', 'you', 'see'), 1),
(('tree', 'd', 'tree'), 1),
(('d', 'not', 'd'), 1),
(('do', 'mot', 'do'), 1),
(('mot', 'like', 'mot'), 1),
(('sam-i-am', 'a', 'sam-i-am'), 1),
(('train', 'could', 'train'), 1),
(('car', 'sam', 'car'), 1),
(('sam', 'let', 'sam'), 1),
(('sam-i-am', 'say', 'sam-i-am'), 1),
(('say', 'in', 'say'), 1),
(('dark', 'here', 'dark'), 1),
(('here', 'in', 'here'), 1),
(('dark', 'i', 'dark'), 1),
(('rain', 'not', 'rain'), 1),
(('car', 'not', 'car'), 1),
(('them', 'sam', 'them'), 1),
(('sam', 'you', 'sam'), 1),
(('see', 'not', 'see'), 1),
(('mouse', 'not', 'mouse'), 1),
(('anywhere', 'you', 'anywhere'), 1),
(('sam-i-am', 'could', 'sam-i-am'), 1),
(('you', 'with', 'you'), 1),
(('goat', 'would', 'goat'), 1),
(('not', 'will', 'not'), 1),
(('them', 'on', 'them'), 1),
(('sam-i-am', 'you', 'sam-i-am'), 1),
(('them', 'so', 'them'), 1),
(('so', 'you', 'so'), 1),
(('you', 'say', 'you'), 1),
(('say', 'try', 'say'), 1),
(('them', 'try', 'them'), 1),
(('may', 'try', 'may'), 1),
(('may', 'i', 'may'), 1),
```

```
 (('i', 'say', 'i'), 1),
 (('say', 'sam', 'say'), 1),
 (('sam', 'if', 'sam'), 1),
 (('if', 'you', 'if'), 1),
 (('will', 'let', 'will'), 1),
 (('will', 'try', 'will'), 1),
 (('see', 'say', 'see'), 1),
 (('do', 'i', 'do'), 1),
 (('sam-i-am', 'and', 'sam-i-am'), 1),
 (('boat', 'and', 'boat'), 1),
 (('goat', 'and', 'goat'), 1),
 (('rain', 'and', 'rain'), 1),
 (('dark', 'and', 'dark'), 1),
 (('and', 'on', 'and'), 1),
 (('train', 'and', 'train'), 1),
 (('car', 'and', 'car'), 1),
 (('tree', 'they', 'tree'), 1),
 (('are', 'so', 'are'), 1),
 (('good', 'so', 'good'), 1),
 (('good', 'you', 'good'), 1),
 (('see', 'so', 'see'), 1),
 (('so', 'i', 'so'), 1),
 (('box', 'and', 'box'), 1),
 (('fox', 'and', 'fox'), 1),
 (('house', 'and', 'house'), 1),
 (('mouse', 'and', 'mouse'), 1),
 (('here', 'and', 'here'), 1),
 (('and', 'there', 'and'), 1),
 (('there', 'say', 'there'), 1),
 (('do', 'so', 'do'), 1),
 (('so', 'like', 'so'), 1),
 (('ham', 'thank', 'ham'), 1),
 (('you', 'thank', 'you'), 1)]
```

In [12]:

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-12-47cbfcab36b7> in <module>()
----> 1 word_list


NameError: name 'word_list' is not defined
```