

Word Completion

Thomas Graf

Stony Brook University
`lin120@thomasgraf.net`

LIN 220, Spring 2019
Lecture 2

Word completion: a simple problem(?)

- ▶ When you write something on your phone, it automatically suggests words while you're typing them.
- ▶ Seems easy, but it's **fairly intricate**.

Key Insights/Skills

- ▶ transition probabilities with n -grams
- ▶ "More than 1, but less than 5."
- ▶ efficient data structures: tries/prefix trees

Attempt 1: simple lookup

- ▶ To make suggestions for completions, we only need an English/German/... dictionary.

```
1  # load English dictionary from nltk package
2  from nltk.corpus import words
3  # and see if "test" is in the list
4  "test" in words.words()
5  >>> True
```

- ▶ Given word **w**, the set of possible completions for **w** consists of all listed words that start with **w**:

$$\text{completions}(\mathbf{w}) = \{w' \mid w' \text{ starts with } \mathbf{w}\}$$

Attempt 1 [cont.]

- completions(**w**) is easily computed in Python.

```
1  def completions(word, wordlist):  
2      """Return set of all known completions for word."""  
3      return {comp for comp in wordlist  
4              if comp.startswith(word)}
```

```
1  completions("testing", words.words())  
2  >>> {"testing", "testingly"}
```

Evaluating attempt 1

- Is this a **good solution?**

Pro	Contra

```
1 completions("test", words.words())
2 >>> {"test", "testa", "testable", "testacean", ...}
```

Evaluating attempt 1

- Is this a **good solution?**

Pro	Contra
simple	

```
1 completions("test", words.words())
2 >>> {"test", "testa", "testable", "testacean", ...}
```

Evaluating attempt 1

- Is this a **good solution?**

Pro	Contra
simple	too many completions

```
1 completions("test", words.words())
2 >>> {"test", "testa", "testable", "testacean", ...}
```

Evaluating attempt 1

- Is this a **good solution?**

Pro	Contra
simple	too many completions unlikely completions

```
1 completions("test", words.words())
2 >>> {"test", "testa", "testable", "testacean", ...}
```


Improving attempt 1

- ▶ We can limit the number of suggestions, if we use a **list** instead of a **set**.

```
1  def completions(word, wordlist):
2      """Return list of all known completions for word."""
3      return [comp for comp in wordlist
4              if comp.startswith(word)]
5
6  completions("test", words.words())[:3]
7  >>> ["test", "testa", "testable"]
```

- ▶ But this still includes unlikely completions like *testa*.
- ▶ We need **probabilities**!

Probability = Frequency (?)

- ▶ We only want to show the **most likely completions**.
- ▶ But what is most likely?
- ▶ **Naive proposal:** the most frequent word!

Probability = Frequency (?)

- ▶ We only want to show the **most likely completions**.
- ▶ But what is most likely?
- ▶ **Naive proposal:** the most frequent word!

Um, but how do we figure out what is most common?

How to find common words

- ▶ **Task:** determine which words are common
- ▶ **Solution:**
 - 1 Collect sufficiently large sample of texts
 - 2 For each word (**type**), count how often it occurs in the entire sample (= its number of **tokens**).
 - 3 Calculate the **frequency** of the word in the sample:

$$\text{freq}(\text{word}, \text{sample}) = \frac{\text{number of tokens of word}}{\text{word length of whole sample}}$$

Types vs tokens

We have to distinguish **word types** (*a, the, Mary, red, ...*) from their **word tokens**, which are the instances of a specific word type. For instance, the type “word” has 4 tokens in this box.

How to find common words

- ▶ **Task:** determine which words are common
- ▶ **Solution:**
 - 1 Collect sufficiently large sample of texts
 - 2 For each word (**type**), count how often it occurs in the entire sample (= its number of **tokens**).
 - 3 Calculate the **frequency** of the word in the sample:

$$\text{freq}(\text{word}, \text{sample}) = \frac{\text{number of tokens of word}}{\text{word length of whole sample}}$$

Types vs tokens

We have to distinguish **word types** (*a*, *the*, *Mary*, *red*, ...) from their **word tokens**, which are the instances of a specific word type. For instance, the type “word” has 4 tokens in this box.

How to find common words

- ▶ **Task:** determine which words are common
- ▶ **Solution:**
 - 1 Collect sufficiently large sample of texts
 - 2 For each word (**type**), count how often it occurs in the entire sample (= its number of **tokens**).
 - 3 Calculate the **frequency** of the word in the sample:

$$\text{freq}(\text{word}, \text{sample}) = \frac{\text{number of tokens of word}}{\text{word length of whole sample}}$$

Types vs tokens

We have to distinguish **word types** (*a, the, Mary, red, ...*) from their **word tokens**, which are the instances of a specific **word** type. For instance, the type “**word**” has **4 tokens** in this box.

Example calculation

Sample: 1000 words long

Words: be, bed, bee, bell

Type	be	bed	bee	bell
Tokens	13	2	0	3

$$\text{freq}(\text{be}) = \frac{13}{1000} = 1.3\%$$

$$\text{freq}(\text{bed}) = \frac{2}{1000} = 0.2\%$$

$$\text{freq}(\text{bee}) = \frac{0}{1000} = 0.0\%$$

$$\text{freq}(\text{bell}) = \frac{3}{1000} = 0.3\%$$

Example calculation

Sample: 1000 words long

Words: be, bed, bee, bell

Type	be	bed	bee	bell
Tokens	13	2	0	3

$$\text{freq}(\text{be}) = \frac{13}{1000} = 1.3\%$$

$$\text{freq}(\text{bee}) = \frac{0}{1000} = 0.0\%$$

$$\text{freq}(\text{bed}) = \frac{2}{1000} = 0.2\%$$

$$\text{freq}(\text{bell}) = \frac{3}{1000} = 0.3\%$$

Ordered completions for *be*:

Example calculation

Sample: 1000 words long

Words: be, bed, bee, bell

Type	be	bed	bee	bell
Tokens	13	2	0	3

$$\text{freq}(\text{be}) = \frac{13}{1000} = 1.3\%$$

$$\text{freq}(\text{bee}) = \frac{0}{1000} = 0.0\%$$

$$\text{freq}(\text{bed}) = \frac{2}{1000} = 0.2\%$$

$$\text{freq}(\text{bell}) = \frac{3}{1000} = 0.3\%$$

Ordered completions for *be*: be

Example calculation

Sample: 1000 words long

Words: be, bed, bee, bell

Type	be	bed	bee	bell
Tokens	13	2	0	3

$$\text{freq}(\text{be}) = \frac{13}{1000} = 1.3\%$$

$$\text{freq}(\text{bee}) = \frac{0}{1000} = 0.0\%$$

$$\text{freq}(\text{bed}) = \frac{2}{1000} = 0.2\%$$

$$\text{freq}(\text{bell}) = \frac{3}{1000} = 0.3\%$$

Ordered completions for *be*: be, bell

Example calculation

Sample: 1000 words long

Words: be, bed, bee, bell

Type	be	bed	bee	bell
Tokens	13	2	0	3

$$\text{freq}(\text{be}) = \frac{13}{1000} = 1.3\%$$

$$\text{freq}(\text{bee}) = \frac{0}{1000} = 0.0\%$$

$$\text{freq}(\text{bed}) = \frac{2}{1000} = 0.2\%$$

$$\text{freq}(\text{bell}) = \frac{3}{1000} = 0.3\%$$

Ordered completions for *be*: be, bell, bed

Example calculation

Sample: 1000 words long

Words: be, bed, bee, bell

Type	be	bed	bee	bell
Tokens	13	2	0	3

$$\text{freq}(\text{be}) = \frac{13}{1000} = 1.3\%$$

$$\text{freq}(\text{bee}) = \frac{0}{1000} = 0.0\%$$

$$\text{freq}(\text{bed}) = \frac{2}{1000} = 0.2\%$$

$$\text{freq}(\text{bell}) = \frac{3}{1000} = 0.3\%$$

Ordered completions for *be*: be, bell, bed, bee

Types of corpora

- **Corpus** = large, structured collections of texts
 - mono-/multilingual just one language, or many?
 - annotated not just text, but additional annotations (e.g. tags for part of speech, syntax trees)

Some common corpora

Brown	1 million words, tagged, 500 samples across 15 genres (fiction, news paper, ...)
Gutenberg	1.8 million words, 18 classic texts of fiction
Penn	40k words, tagged and parsed
Reuters	1.3 million words, news documents
Switchboard	36 phone calls, fully transcribed and parsed
Wordlist	960k words (no repetitions) and 20k affixes for 8 languages

Getting probabilities from the corpus

```
1  from nltk.corpus import brown
2  from collections import Counter
3
4  # load Brown corpus as sequence of words
5  brown_text = brown.words()
6  # total number of words = length of text
7  total = len(brown_text)
8  # calculate counts
9  brown_counts = Counter(brown_text)
10 # convert counts to frequencies
11 for word in brown_counts:
12     brown_counts[word] = brown_counts[word]/total
```

- Alright, we have frequencies for each word.
Now what?

Ordering completions by frequency

- For a good user experience, completions should appear in **descending order of probability**.

```
1 def completions(word, counts):
2     """Return set of all known completions for word.
3
4     The completions are sorted by frequency,
5     in descending order.
6     """
7     comps = [comp for comp in counts
8                if comp.startswith(word)]
9     return sorted(comps,
10                   key=lambda x: counts[x],
11                   reverse=True)
```

```
1 completions("test", words.words(), brown_counts)
2 >>> ["test", "testimony", "tested", "testing", ...]
```

Summary for revised attempt 1

- 1 **Needed resources:** corpus
- 2 Compute frequencies for all words in corpus
- 3 Look up possible completions for user input
- 4 Sort completions by their frequency

Great, we're done, right?! Not quite. . .

Probability \neq word frequency

- ▶ The probability of a word isn't fixed, it varies by **context**.

Example

	tested	testing	testimony
I have	hi	low	mid
I have been	hi	hi	low
I have the	low	low	hi

- ▶ The frequency of words is not enough,
we need frequencies of sequences of words \Rightarrow **n-grams**

Defining n-grams

n-gram a contiguous sequence of n words

n	Name	Example
1	unigram	John
2	bigram	John to
3	trigram	John to be
4	4-gram	John to be in
5	5-gram	John to be in the

Example

String

John and Marie are not Bill and Sue

Defining n-grams

n-gram a contiguous sequence of n words

n	Name	Example
1	unigram	John
2	bigram	John to
3	trigram	John to be
4	4-gram	John to be in
5	5-gram	John to be in the

Example

String

John and Marie are not Bill and Sue

Defining n-grams

n-gram a contiguous sequence of n words

n	Name	Example
1	unigram	John
2	bigram	John to
3	trigram	John to be
4	4-gram	John to be in
5	5-gram	John to be in the

Example

String

John and Marie are not Bill and Sue

Defining n-grams

n-gram a contiguous sequence of n words

n	Name	Example
1	unigram	John
2	bigram	John to
3	trigram	John to be
4	4-gram	John to be in
5	5-gram	John to be in the

Example

String

John and Marie are not Bill and Sue

Defining n-grams

n-gram a contiguous sequence of n words

n	Name	Example
1	unigram	John
2	bigram	John to
3	trigram	John to be
4	4-gram	John to be in
5	5-gram	John to be in the

Example

String

John and Marie are not Bill and Sue

Defining n-grams

n-gram a contiguous sequence of n words

n	Name	Example
1	unigram	John
2	bigram	John to
3	trigram	John to be
4	4-gram	John to be in
5	5-gram	John to be in the

Example

String

John and Marie are not Bill and Sue

Defining n-grams

n-gram a contiguous sequence of n words

n	Name	Example
1	unigram	John
2	bigram	John to
3	trigram	John to be
4	4-gram	John to be in
5	5-gram	John to be in the

Example

String

John and Marie are not Bill and Sue

Defining n-grams

n-gram a contiguous sequence of n words

n	Name	Example
1	unigram	John
2	bigram	John to
3	trigram	John to be
4	4-gram	John to be in
5	5-gram	John to be in the

Example

String

John and Marie are not Bill and Sue

Defining n-grams

n-gram a contiguous sequence of n words

n	Name	Example
1	unigram	John
2	bigram	John to
3	trigram	John to be
4	4-gram	John to be in
5	5-gram	John to be in the

Example

String

John and Marie are not Bill and Sue

Defining n-grams

n-gram a contiguous sequence of n words

n	Name	Example
1	unigram	John
2	bigram	John to
3	trigram	John to be
4	4-gram	John to be in
5	5-gram	John to be in the

Example

String

John and Marie are not Bill and Sue

Defining n-grams

n-gram a contiguous sequence of n words

n	Name	Example
1	unigram	John
2	bigram	John to
3	trigram	John to be
4	4-gram	John to be in
5	5-gram	John to be in the

Example

String

John and Marie are not Bill and Sue

Defining n-grams

n-gram a contiguous sequence of n words

n	Name	Example
1	unigram	John
2	bigram	John to
3	trigram	John to be
4	4-gram	John to be in
5	5-gram	John to be in the

Example

String

John and Marie are not Bill and Sue

Defining n-grams

n-gram a contiguous sequence of n words

n	Name	Example
1	unigram	John
2	bigram	John to
3	trigram	John to be
4	4-gram	John to be in
5	5-gram	John to be in the

Example

String

John and Marie are not Bill and Sue

Defining n-grams

n-gram a contiguous sequence of n words

n	Name	Example
1	unigram	John
2	bigram	John to
3	trigram	John to be
4	4-gram	John to be in
5	5-gram	John to be in the

Example

String

John and Marie are not Bill and Sue

Defining n-grams

n-gram a contiguous sequence of n words

n	Name	Example
1	unigram	John
2	bigram	John to
3	trigram	John to be
4	4-gram	John to be in
5	5-gram	John to be in the

Example

String

John and Marie are not Bill and Sue

Defining n-grams

n-gram a contiguous sequence of n words

n	Name	Example
1	unigram	John
2	bigram	John to
3	trigram	John to be
4	4-gram	John to be in
5	5-gram	John to be in the

Example

String

John and Marie are not Bill and Sue

Defining n-grams

n-gram a contiguous sequence of n words

n	Name	Example
1	unigram	John
2	bigram	John to
3	trigram	John to be
4	4-gram	John to be in
5	5-gram	John to be in the

Example

String

John and Marie are not Bill and Sue

Defining n-grams

n-gram a contiguous sequence of n words

n	Name	Example
1	unigram	John
2	bigram	John to
3	trigram	John to be
4	4-gram	John to be in
5	5-gram	John to be in the

Example

String

John and Marie are not Bill and Sue

Defining n-grams

n-gram a contiguous sequence of n words

n	Name	Example
1	unigram	John
2	bigram	John to
3	trigram	John to be
4	4-gram	John to be in
5	5-gram	John to be in the

Example

String

John and Marie are not Bill and Sue

Defining n-grams

n-gram a contiguous sequence of n words

n	Name	Example
1	unigram	John
2	bigram	John to
3	trigram	John to be
4	4-gram	John to be in
5	5-gram	John to be in the

Example

String

John and Marie are not Bill and Sue

Defining n-grams

n-gram a contiguous sequence of n words

n	Name	Example
1	unigram	John
2	bigram	John to
3	trigram	John to be
4	4-gram	John to be in
5	5-gram	John to be in the

Example

String

John and Marie are not Bill and Sue

Defining n-grams

n-gram a contiguous sequence of n words

n	Name	Example
1	unigram	John
2	bigram	John to
3	trigram	John to be
4	4-gram	John to be in
5	5-gram	John to be in the

Example

String

John and Marie are not Bill and Sue

Defining n-grams

n-gram a contiguous sequence of n words

n	Name	Example
1	unigram	John
2	bigram	John to
3	trigram	John to be
4	4-gram	John to be in
5	5-gram	John to be in the

Example

String

John and Marie are not Bill and Sue

Frequencies for n-grams

Frequencies can be computed for n-grams, too.

Example: Calculating Bigram Frequencies

- ▶ **String**

when buffalo buffalo buffalo buffalo buffalo

- ▶ **Bigram token list**

- ▶ **Bigram counts and frequencies**

Frequencies for n-grams

Frequencies can be computed for n-grams, too.

Example: Calculating Bigram Frequencies

- ▶ **String**

when buffalo buffalo buffalo buffalo buffalo

- ▶ **Bigram token list**

when buffalo,

- ▶ **Bigram counts and frequencies**

Frequencies for n-grams

Frequencies can be computed for n-grams, too.

Example: Calculating Bigram Frequencies

- ▶ **String**

when buffalo buffalo buffalo buffalo buffalo buffalo

- ▶ **Bigram token list**

when buffalo, buffalo buffalo,

- ▶ **Bigram counts and frequencies**

Frequencies for n-grams

Frequencies can be computed for n-grams, too.

Example: Calculating Bigram Frequencies

- ▶ **String**

when buffalo buffalo buffalo buffalo buffalo

- ▶ **Bigram token list**

when buffalo, buffalo buffalo, buffalo buffalo,

- ▶ **Bigram counts and frequencies**

Frequencies for n-grams

Frequencies can be computed for n-grams, too.

Example: Calculating Bigram Frequencies

- ▶ **String**

when buffalo buffalo buffalo buffalo buffalo buffalo

- ▶ **Bigram token list**

when buffalo, buffalo buffalo, buffalo buffalo, buffalo buffalo,

- ▶ **Bigram counts and frequencies**

Frequencies for n-grams

Frequencies can be computed for n-grams, too.

Example: Calculating Bigram Frequencies

- ▶ **String**

when buffalo buffalo buffalo buffalo buffalo buffalo

- ▶ **Bigram token list**

when buffalo, buffalo buffalo, buffalo buffalo, buffalo buffalo,
buffalo buffalo,

- ▶ **Bigram counts and frequencies**

Frequencies for n-grams

Frequencies can be computed for n-grams, too.

Example: Calculating Bigram Frequencies

- ▶ **String**

when buffalo buffalo buffalo buffalo buffalo buffalo

- ▶ **Bigram token list**

when buffalo, buffalo buffalo, buffalo buffalo, buffalo buffalo,
buffalo buffalo, buffalo buffalo

- ▶ **Bigram counts and frequencies**

Frequencies for n-grams

Frequencies can be computed for n-grams, too.

Example: Calculating Bigram Frequencies

- ▶ **String**

when buffalo buffalo buffalo buffalo buffalo

- ▶ **Bigram token list**

when buffalo, buffalo buffalo, buffalo buffalo, buffalo buffalo,
buffalo buffalo, buffalo buffalo

- ▶ **Bigram counts and frequencies**

- 1 when buffalo: $1 \Rightarrow \frac{1}{6} = 16.7\%$
- 2 buffalo buffalo: $5 \Rightarrow \frac{5}{6} = 83.3\%$

Adapting the strategy

- 1 **Needed resources:** corpus
- 2 Convert corpus to list of n -gram tokens
- 3 Compute frequencies for all n -grams
- 4 Look up possible completions for user input
- 5 Look at previous $n - 1$ words.
- 6 Sort completions by n -gram probability

Example

► Trigram frequencies

bus is late	30%	train is late	15%
bus is lovely	25%	train is lovely	8%
bus is lazy	10%	train is lazy	2%

► Input

I will text you if the train is I

► Sorted completions

Adapting the strategy

- 1 **Needed resources:** corpus
- 2 Convert corpus to list of n -gram tokens
- 3 Compute frequencies for all n -grams
- 4 Look up possible completions for user input
- 5 Look at previous $n - 1$ words.
- 6 Sort completions by n -gram probability

Example

► Trigram frequencies

bus is late	30%	train is late	15%
bus is lovely	25%	train is lovely	8%
bus is lazy	10%	train is lazy	2%

► Input

I will text you if the train is I

► Sorted completions

late

Adapting the strategy

- 1 **Needed resources:** corpus
- 2 Convert corpus to list of n -gram tokens
- 3 Compute frequencies for all n -grams
- 4 Look up possible completions for user input
- 5 Look at previous $n - 1$ words.
- 6 Sort completions by n -gram probability

Example

► Trigram frequencies

bus is late	30%	train is late	15%
bus is lovely	25%	train is lovely	8%
bus is lazy	10%	train is lazy	2%

► Input

I will text you if the train is I

► Sorted completions

late, lovely

Adapting the strategy

- 1 **Needed resources:** corpus
- 2 Convert corpus to list of n -gram tokens
- 3 Compute frequencies for all n -grams
- 4 Look up possible completions for user input
- 5 Look at previous $n - 1$ words.
- 6 Sort completions by n -gram probability

Example

► Trigram frequencies

bus is late	30%	train is late	15%
bus is lovely	25%	train is lovely	8%
bus is lazy	10%	train is lazy	2%

► Input

I will text you if the train is I

► Sorted completions

late, lovely, lazy

How it is done: The easy generalization step

```
1 def bigrams(text):  
2     """Convert text to list of bigram tokens."""  
3     return [text[n:n+2] for n in range(len(text) - 1)]
```

```
1 brown_bigrams = bigrams(brown.words())  
2 total = len(brown_bigrams)  
3 brown_bicounts = Counter(brown_bigrams)  
4 for bigram in brown_bicounts:  
5     brown_bicounts[bigram] = brown_bicounts[bigram]/total
```

How it is done: The super-easy part

```
1  def bigram_completions(word, previous_word, counts):
2      # set of all compatible bigrams
3      comps = [comp for comp in counts
4                  if comp[0] == previous_word and
5                      comp[-1].startswith(word)]
6      # sort the bigram completions
7      ordered_ngrams = sorted(comps,
8                              key=lambda x: counts[x],
9                              reverse=True)
10     # only keep second word of each bigram
11     return [ngram[-1] for ngram in ordered_ngrams]
```