



## Markovify

Markovify is a simple, extensible Markov chain generator. Right now, its primary use is for building Markov models of large corpora of text and generating random sentences from that. However, in theory, it could be used for other applications.

- Why Markovify?
- Installation
- Basic Usage
- Advanced Usage
- Markovify In The Wild
- Thanks

### Why Markovify?

Some reasons:

- Simplicity. “Batteries included,” but it is easy to override key methods.
- Models can be stored as JSON, allowing you to cache your results and save them for later.
- Text parsing and sentence generation methods are highly extensible, allowing you to set your own rules.
- Relies only on pure-Python libraries, and very few of them.
- Tested on Python 3.7, 3.8, 3.9, and 3.10.

### Installation

```
1 pip install markovify
```

### Basic Usage

---

```
1 import markovify
2
3 # Get raw text as string.
4 with open("/path/to/my/corpus.txt") as f:
5     text = f.read()
6
7 # Build the model.
8 text_model = markovify.Text(text)
9
10 # Print five randomly-generated sentences
11 for i in range(5):
12     print(text_model.make_sentence())
13
14 # Print three randomly-generated sentences of no more than 280
    characters
15 for i in range(3):
16     print(text_model.make_short_sentence(280))
```

Notes:

- The usage examples here assume you are trying to markovify text. If you would like to use the underlying `markovify.Chain` class, which is not text-specific, check out the (annotated) source code.
- Markovify works best with large, well-punctuated texts. If your text does not use `.` s to delineate sentences, put each sentence on a newline, and use the `markovify.NewlineText` class instead of `markovify.Text` class.
- If you have accidentally read the input text as one long sentence, markovify will be unable to generate new sentences from it due to a lack of beginning and ending delimiters. This issue can occur if you have read a newline delimited file using the `markovify.Text` command instead of `markovify.NewlineText`. To check this, the command `[key for key in txt.chain.model.keys() if "___BEGIN___" in key]` command will return all of the possible sentence-starting words and should return more than one result.
- By default, the `make_sentence` method tries a maximum of 10 times per invocation, to make a sentence that does not overlap too much with the original text. If it is successful, the method returns the sentence as a string. If not, it returns `None`. To increase or decrease the number of attempts, use the `tries` keyword argument, e.g., call `.make_sentence(tries=100)`.
- By default, `markovify.Text` tries to generate sentences that do not simply regurgitate chunks of the original text. The default rule is to suppress any generated sentences that exactly overlaps the original text by 15 words or 70% of the sentence's word count. You can change this rule by passing `max_overlap_ratio` and/or `max_overlap_total` to the `make_sentence` method. Alternatively, this check can be disabled entirely by passing

---

`test_output` as `False`.

## Advanced Usage

### Specifying the model's state size

State size is a number of words the probability of a next word depends on.

By default, `markovify.Text` uses a state size of 2. But you can instantiate a model with a different state size. E.g.,:

```
1 text_model = markovify.Text(text, state_size=3)
```

### Combining models

With `markovify.combine(...)`, you can combine two or more Markov chains. The function accepts two arguments:

- **models**: A list of `markovify` objects to combine. Can be instances of `markovify.Chain` or `markovify.Text` (or their subclasses), but all must be of the same type.
- **weights**: Optional. A list — the exact length of **models** — of ints or floats indicating how much relative emphasis to place on each source. Default: `[ 1, 1, ... ]`.

For instance:

```
1 model_a = markovify.Text(text_a)
2 model_b = markovify.Text(text_b)
3
4 model_combo = markovify.combine([ model_a, model_b ], [ 1.5, 1 ])
```

This code snippet would combine `model_a` and `model_b`, but, it would also place 50% more weight on the connections from `model_a`.

### Compiling a model

Once a model has been generated, it may also be compiled for improved text generation speed and reduced size.

```
1 text_model = markovify.Text(text)
2 text_model = text_model.compile()
```

Models may also be compiled in-place:

---

```
1 text_model = markovify.Text(text)
2 text_model.compile(inplace = True)
```

Currently, compiled models may not be combined with other models using `markovify.combine` (. . .). If you wish to combine models, do that first and then compile the result.

### Working with messy texts

Starting with v0.7.2, `markovify.Text` accepts two additional parameters: `well_formed` and `reject_reg`.

- Setting `well_formed = False` skips the step in which input sentences are rejected if they contain one of the ‘bad characters’ (i.e. `()[]'"`)
- Setting `reject_reg` to a regular expression of your choice allows you change the input-sentence rejection pattern. This only applies if `well_formed` is True, and if the expression is non-empty.

### Extending `markovify.Text`

The `markovify.Text` class is highly extensible; most methods can be overridden. For example, the following `POSifiedText` class uses NLTK’s part-of-speech tagger to generate a Markov model that obeys sentence structure better than a naive model. (It works; however, be warned: `pos_tag` is very slow.)

```
1 import markovify
2 import nltk
3 import re
4
5 class POSifiedText(markovify.Text):
6     def word_split(self, sentence):
7         words = re.split(self.word_split_pattern, sentence)
8         words = [ ":%s:" % tag for tag in nltk.pos_tag(words) ]
9         return words
10
11     def word_join(self, words):
12         sentence = " ".join(word.split(":%s:") [0] for word in words)
13         return sentence
```

Or, you can use `spaCy` which is way faster:

```
1 import markovify
2 import re
3 import spacy
```

---

```
4
5 nlp = spacy.load("en_core_web_sm")
6
7 class POSifiedText(markovify.Text):
8     def word_split(self, sentence):
9         return "::".join((word.orth_, word.pos_) for word in nlp(
            sentence))
10
11     def word_join(self, words):
12         sentence = " ".join(word.split("::")[0] for word in words)
13         return sentence
```

The most useful `markovify.Text` models you can override are:

- `sentence_split`
- `sentence_join`
- `word_split`
- `word_join`
- `test_sentence_input`
- `test_sentence_output`

For details on what they do, see the (annotated) source code.

## Exporting

It can take a while to generate a Markov model from a large corpus. Sometimes you'll want to generate once and reuse it later. To export a generated `markovify.Text` model, use `my_text_model.to_json()`. For example:

```
1 corpus = open("sherlock.txt").read()
2
3 text_model = markovify.Text(corpus, state_size=3)
4 model_json = text_model.to_json()
5 # In theory, here you'd save the JSON to disk, and then read it back
   later.
6
7 reconstituted_model = markovify.Text.from_json(model_json)
8 reconstituted_model.make_short_sentence(280)
9
10 >>> 'It cost me something in foolscap, and I had no idea that he was a
      man of evil reputation among women.'
```

You can also export the underlying Markov chain on its own — i.e., excluding the original corpus and the `state_size` metadata — via `my_text_model.chain.to_json()`.

---

## Generating `markovify.Text` models from very large corpora

By default, the `markovify.Text` class loads, and retains, your textual corpus, so that it can compare generated sentences with the original (and only emit novel sentences). However, with very large corpora, loading the entire text at once (and retaining it) can be memory-intensive. To overcome this, you can (a) tell Markovify not to retain the original:

```
1 with open("path/to/my/huge/corpus.txt") as f:
2     text_model = markovify.Text(f, retain_original=False)
3
4 print(text_model.make_sentence())
```

And (b) read in the corpus line-by-line or file-by-file and combine them into one model at each step:

```
1 combined_model = None
2 for (dirpath, _, filenames) in os.walk("path/to/my/huge/corpus"):
3     for filename in filenames:
4         with open(os.path.join(dirpath, filename)) as f:
5             model = markovify.Text(f, retain_original=False)
6             if combined_model:
7                 combined_model = markovify.combine(models=[
8                     combined_model, model])
9             else:
10                 combined_model = model
11 print(combined_model.make_sentence())
```

## Markovify In The Wild

- BuzzFeed’s Tom Friedman Sentence Generator / @mot\_namdeirf.
- /u/user\_simulator, a Reddit bot that generates comments based on a user’s comment history. [code]
- SubredditSimulator, which uses `markovify` to generate random Reddit submissions and comments based on a subreddit’s previous activity. [code]
- college crapplication, a web-app that generates college application essays. [code]
- @MarkovPicard, a Twitter bot based on *Star Trek: The Next Generation* transcripts. [code]
- sekrits.herokuapp.com, a `markovify`-powered quiz that challenges you to tell the difference between “two file titles relating to matters of [Australian] national security” — one real and one fake. [code]
- Hacker News Simulator, which does what it says on the tin. [code]
- Stak Attak, a “poetic stackoverflow answer generator.” [code]

- 
- MashBOT, a [markovify](#)-powered Twitter bot attached to a printer. Presented by Helen J Burgess at Babel Toronto 2015. [code]
  - The Mansfield Reporter, “a simple device which can generate new text from some of history’s greatest authors [...] running on a tiny Raspberry Pi, displaying through a tft screen from Adafruit.”
  - twitter markov, a tool to “create markov chain (“\_ebooks”) accounts on Twitter.”
  - @Bern\_Trump\_Bot, “Bernie Sanders and Donald Trump driven by Markov Chains.” [code]
  - @RealTrumpTalk, “A bot that uses the things that @realDonaldTrump tweets to create it’s own tweets.” [code]
  - Taylor Swift Song Generator, which does what it says. [code]
  - @BOTtalks / ideasworthautomating.com. “TIM generates talks on a broad spectrum of topics, based on the texts of slightly more coherent talks given under the auspices of his more famous big brother, who shall not be named here.” [code]
  - Internal Security Zones, “Generative instructions for prison design & maintenance.” [code]
  - Miraculous Ladybot. Generates Miraculous Ladybug fanfictions and posts them on Tumblr. [code]
  - @HaikuBotto, “I’m a bot that writes haiku from literature. beep boop” [code]
  - Chat Simulator Bot, a bot for Telegram. [code]
  - emoji pasta.club, “a web service that exposes RESTful endpoints for generating emoji pastas, as well as a simple frontend for generating and tweeting emoji pasta sentences.” [code]
  - Towel Generator, “A system for generating sentences similar to those from the hitchhikers series of books.” [code]
  - @mercurialbot, “A twitter bot that generates tweets based on its mood.” [code]
  - becomeacurator.com, which “generates curatorial statements for contemporary art exhibitions, using Markov chains and texts from galleries around the world.” [code]
  - mannynotfound/interview-bot, “A python based terminal prompt app to automate the interview process.”
  - Steam Game Generator, which “uses data from real Steam games, randomized using Markov chains.” [code]
  - @DicedOnionBot, which “generates new headlines by The Onion by regurgitating and combining old headlines.” [code]
  - @thought\_\_leader, “Thinking thoughts so you don’t have to!” [blog post]
  - @\_murakamibot and @jamesjoycebot, bots that tweet Haruki Murakami and James Joyce-like sentences. [code]
  - shartificialintelligence.com, “the world’s first creative ad agency staffed entirely with copywriter robots.” [code]
  - @NightValeFeed, which “generates tweets by combining @NightValeRadio tweets with @BuzzFeed headlines.” [code]
-

- 
- Wynbot9000, which “mimics your friends on Google Hangouts.” [code]
  - @sealDonaldTrump, “a twitter bot that sounds like @realDonaldTrump, with an aquatic twist.” [code]
  - @veeceeboot, which is “like VCs but better!” [code]
  - @mar\_phil\_bot, a Twitter bot trained on Nietzsche, Russell, Kant, Machiavelli, and Plato. [code]
  - funzo-facts, a program that generates never-before-seen trivia based on Jeopardy! questions. [code]
  - Chains Invent Insanity, a Cards Against Humanity answer card generator. [code]
  - @CanDennisDream, a twitter bot that contemplates life by training on existential literature discussions. [code]
  - B-9 Indifference, a program that generates a *Star Trek: The Next Generation* script of arbitrary length using Markov chains trained on the show’s episode and movie scripts. [code]
  - adam, polish poetry generator. [code]
  - Stackexchange Simulator, which uses StackExchange’s bulk data to generate random questions and answers. [code]
  - @BloggingBot, tweets sentences based on a corpus of 17 years of blogging.
  - Commencement Speech Generator, generates “graduation speech”-style quotes from a dataset of the “greatest of all time” commencement speeches)
  - @alg\_testament, tweets sentences based on The Old Testament and two coding textbooks in Russian. [code]
- 
- @IRAMockBot, uses Twitter’s data on tweets from Russian IRA-associated accounts to produce fake IRA tweets, for educational and study purposes.[code]
  - Personal Whatsapp Chat Analyzer, some basic analytics for WhatsApp chat exports (private & groups), word counting & markov chain phrase generator
  - DeepfakeBot, a system for converting your friends into Discord bots. [code]
  - python-markov-novel, writes a random novel using markov chains, broken down into chapters
  - python-ia-markov, trains Markov models on Internet Archive text files
  - @bot\_homer, a Twitter bot trained using Homer Simpson’s dialogues of 600 chapters. [code].
  - git-commit-gen, generates git commit messages by using markovify to build a model of a repo’s git log
  - fakesocial, a fake social network using generated content. [code]
  - Slovel Bot, a Telegram bot that generates non-existent Russian words using corpus made by algorithmically dividing existent words into syllables.
  - Deuterium, a Discord bot that generates messages on its own, after analyzing yours, and learning constantly. There’s also a global model shared with all other servers.
  - Markovify Piano, generates coherent and plausible music generation.
-



- 
- TweepyPy, a Twitter bot that takes data from the top US trends or user tweets, learn and create own tweets and word clouds. [code]
  - cappuccino/ai.py, an IRC bot with a plugin that generates sentences based on the PostgreSQL-stored logs of the IRC channels it's in. [code]
  - django-markov, a reusable Django app that provides a generic backend to generate and store Markov chains for later retrieval and generation of sentences.

Have other examples? Pull requests welcome.

## Thanks

Many thanks to the following GitHub users for contributing code and/or ideas:

- @orf
- @deimos
- @cjmoehrie
- @Jaza
- @fitnr
- @andela-mfalade
- @nratcliff
- @schollz
- @aalireza
- @bfontaine
- @tmsherman
- @wodim
- @eh11fx
- @ammgws
- @OtakuMegane
- @tsunaminoai
- @MatthewScholefield
- @danmayer
- @kade-robertson
- @erikerlandson
- @briennakh
- @berfr
- @Freestackmejai
- @rokala
- @eumiro
- @monosans

- 
- @aogier
  - @terisikk

Initially developed at BuzzFeed.