Sugaroid

Release v0.15.2

Srevin Saju, The Sugaroid Project

Jan 31, 2021

CONTENTS:

CHAPTER

ONE

INTRODUCTION

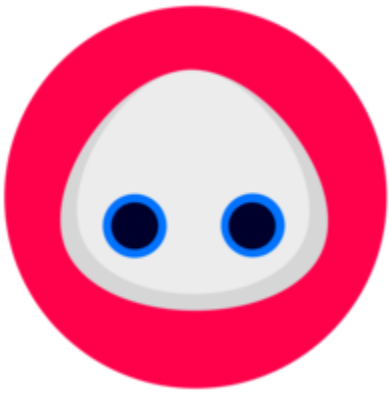

Fig. 1: Sugaroid smiling

1.1 Sugaroid

IMPORTANT : Sugaroid is an open source software. The web server is deployed on Microsoft Azure. Your support for this open source software is highly necessary to make this project continued to be served on the world wide web. Consider being my patron to help Sugaroid host its servers or if you are willing to lend servers for Sugaroid, press the sponsor button and email me. Thanks. However, Sugaroid will always remain free forever :smile:

1.1.1 Introduction

Sugaroid is a new Artificial Intelligence which uses Natural Language Processing (NLP) with Machine Learning and neural networks to manipulate user input to provide a intuitive response. The AI is built on `Python 3.8.2` and was built out of personal interest, to tackle three important issues in the Python framework

- Natural Language Processing / Machine Learning
- Graphical User Interface
- Database Management, Configuration file management and Web Development

Sugaroid Chatbot has a comprehensive and modular interface utilizing Object Oriented Programming to benefit activities of Sugarlabs, a non-profit educational organization. Initially built to serve as a companion bot, the Sugaroid Virtual Assistant helps to comprehend most of the messages, to generate a probable response. The future plans of sugaroid aims to extend Sugaroid as a documentation reader of which beta previews are `still under testing`.

The Sugaroid bot is deployed in production servers particularly for testing.

- The web interface
- The discord bot
- IRC bot hosted on self when necessary

1.2 Configuration

Sugaroid saves some data to your PC. The path where `sugaroid` saves the data is `~/.config/sugaroid` on Linux and Mac OS, but on Windows it is in `C:\Users\<username>\AppData\sugaroid\`

This is the training database used my sugaroid to answer your questions. Particularly related to `sugaroid` brain, the files are `sugaroid.db` and `sugaroid.trainer.json`

- `sugaroid.db` : The Sugaroid bot uses `SQLite` to read data from a persistent database. Remove `sugaroid.db` will reset `sugaroid`'s brain, and a fresh database will be created from scratch
- `sugaroid.trainer.json` : Is a JavaScript Object Notation file which stores trained responses in order to reset or retrain them whenever there is a necessity. This file may or may not be present in end user's systems and depends solely on the type of release `dev` or `stable`
- `sugaroid_internal.db`: A training dataset which learns from user input and accordingly saves them with low confidence. This data is later used to train sugaroid in future according to probability datasets

There might also be additional files in the configuration directory. These are Audio files, In the case that the `audio` keyword is passed as an argument, it creates samples of audio files downloaded from the `Google` server to serve TTS (Text to Speech) to the end user.

1.3 Databases and Training

Sugaroid uses an `sqlite3`-type database for portability. All the responses are explicitly saved and trained on sugaroid. Sugaroid has two types of training: 1. Supervised training 2. Unsupervised training

1.3.1 Supervised training

Supervised training is a list of proper responses, most commonly collected from the Stanford Question Answering Dataset (Natural) (SQuAD 2.0 from Stanford NLP, attribution to Rajpurkar & Jia et al. '18). Other reponses are manually trained from interactions during testing. All the responses are saved to `~/.config/sugaroid/sugaroid.db` which is opened in read-only mode during production mode to prevent people from tampering with the dataset. At local testing, it is possible to teach sugaroid a sequel of responses and this will appended to the SQL database. Using Naive Bayers algorithm.

1.3.2 Unsupervised Training

Unsupervised training are a community collected dataset. The sources of data, are obviously from the community, on its hosted sugaroid.srevinsaju.me instance on Microsoft Azure, frontend on AWS. This data are also appended to the SQL database like *Supervised Training* but they are saved with lesser confidence (`0.1 * confidence_from_statement`), as data from community needs to undergo refining.

1.3.3 `sqlite3`

Sugaroid's backend module is `sqlite3` against the conventional MySQL or MariaDB adapters. `sqlite3` was chosen considering its portability alone. Despite higher IO operations on `sqlite3`, community data collection becomes easier because `sqlite3` databases are more or less, a single file. Another problem it solves is the different ways in which the operating systems consider the file path to be. Using `sqlite3` helps to keep consistency in case. (For Windows, `mysql` is case insensitive, but on GNU/Linux/UNIX its case sensitive). Using `sqlite3` solves that problem.

1.3.4 Privacy policy

Sugaroid collects data from its users which are then used to train. This is done through cookies, on the first response you provide to sugaroid (on the web interface), on adding the bot to your discord channel (on the Discord adapter). However, your data is completely safe, and is not collected for training purposes if its (i) self hosted (ii) run as a desktop / command line app. All data on the desktop version is still appended to your respective configuration folders, which is, for example, on Linux, `~/.config/sugaroid/sugaroid.db` and on Windows its `C:\Users\foobar\AppData\Local\sugaroid\sugaroid.db`.

> Note: `AppData` folder is normally hidden on Windows, manually "Show all hidden folders" to see the AppData folder.

1.3.5 Investigating data from the database

There are certain cases when you would like to analyze the data stored in the database, or you would like to do some debugging. In all such cases, the path to the `sugaroid.db` is very much useful. All you need is an `sqlite3` binary, which is available for all platforms.

> Download `sqlite3` from here

And then, start investigating by

```
$ sqlite3 ~/.config/sugaroid/sugaroid.db
```

This will open a prompt, where you can enter most commands;

Apart from the main database, `sugaroid` also stores data in * `~/.config/sugaroid/sugaroid.db` * `~/.config/sugaroid/sugaroid.trainer.json` * `~/.config/sugaroid/sugaroid_internal.db` * `~/.config/sugaroid/data.json`

Along with SQL, we have also used JSON type files for configuration alone.

1.4 Datasets

Sugaroid's brains lies in its datasets. It might not make sense and can possibly give wrong replies if its not trained with the default dataset. Its more like "*Artificially Foolish*" without a dataset.

1.4.1 Prebuilt datasets

Sugaroid uses a few well known datasets which helps to increase the accuracy of natural language processing. These are provided and fetched by `nltk` and `spacy`, which are popular natural language processing libraries used in Python.

A list of datasets include * `averaged_perceptron_tagger` * `punkt` * `vader_lexicon`

Some of the corpora used by `sugaroid` are * `stopwords` corpus * `wordnet` corpus

> **What is corpus?** Corpus is a text file which contains useful information which can be precisely extracted to get useful information. `stopwords` are words which are commonly used in English speech. Most of the time, `stopwords` do not contain important meanings of the statement to the bot. `stopwords` give meaning to robots. Some examples of stopword are `if`, `on`, `is`, `are`, etc.

Wordnet

Wordnet is a collection of arrays of words which have a unique lemma. Some words may be used as an exaggeration, or sometimes, the same word is used in superlative, comparative tones. At many times, its very useful to ignore such words and depend on the lemma (aka root word). Wordnet is a very interesting library that helps to make things simpler.

Vader Lexicon

Vader Lexicon is a zipped sentiment analyzer which contains many statements with vector scores of a respective words. A resultant vector product is take to find out the approximate sentiment polar score (positive or negative statment). However trained, Vader Lexicon is not very accurate its terms, but however, it remains one of the best datasets used in sugaroid!

Punkt

Punkt is a punctuation library used by Sugar to understand mood of a statement, i.e., interrogative mood, imperative mood, negation, etc.

1.5 Faults

1.5.1 Invalid Responses

Sometimes, the similarity algorithms may give a completely incorrect answer that may lead to false response by the bot to the user. This is because tensors have no resultant displacement and has multiple direction. To compute zero vectors, SpaCy uses an approximation algorithm called Word Mover Distance. This might lead to unknown predictions. Such predictions should be raised as an issue on the Sugaroid repository to create a tackler adapter that would override the answer with a suitable confidence value.

The other complex and efficient algorithms have been neglected. This is to reduce the size of the distribution as well as reduce the time of installation on an end-user's PC. Complex and accurate Natural Language Processing systems like `pytorch` and `tensorflow` exists, but this may result in the net user installation size to be approximately 2 GB +, which is probably not what the end-user requires.

1.6 Execution

Running sugaroid is easy as pie

Just execute

```
$ sugaroid
```

from the Terminal (Linux, Mac OS) and PowerShell (on Windows)

There are few arguments that can be passed to sugaroid

`qt` : Running sugaroid qt will start the sugaroid graphical user interface

`audio` : Running sugaroid audio will include audio support for sugaroid (Data charges may apply)

`train`: Running sugaroid train will start the sugaroid trainer, which you can use to train sugaroid for some responses

`update` : Running sugaroid update will clear the current database and train the new data and store it persistently to the configuration path as sugaroid.db . (See Configuration for more details)

To launch the sugaroid web server on any IP address, do a local clone of the package by

```
git clone https://github.com/srevinsaju/sugaroid-wsgi --depth=1
cd sugaroid-wsgi
python manage.py runserver
```

Follow the on-screen instructions to get it running on your web browser. If the command completed with a status OK, you should be able to see sugaroid running on http://0.0.0.0:8000

1.7 Dependencies

There are certain requirements which are necessary for the proper functioning of Sugaroid chat bot.

- **wikipedia-API** - Handles Wikipedia based questions
- **newsapi-python** - Provides news headlines
- **chatterbot** - Gives basic logic to Sugaroid

- **https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-2.3.1/en_core_web_sm-2.3.1.tar.gz** - Models used for Language Processing
- **pyspellchecker** - Checks spellings to give appropriate results
- **spacy** - A language processor
- **python-dotenv**
- **nltk** - Another Language Processing platform
- **chatterbot** - Used for training Sugaroid
- **colorama** - Prints coloured text
- **freegames** - Collection of free games
- **requests** - Creates HTTP requests
- **lxml** - Handles HTML and XML files
- **beautifulsoup4** - Gets data from other webpages
- **django-googlesearch** - A custom google search engine in Django
- **googletrans** - Translates text
- **akinator.py** - Plays a game of Akinator with the user
- **emoji** - Allows emoji printing
- **pyinflect** - Adds word inflections
- **currencyconverter** - Used to convert currencies

1.8 Requirements

1.8.1 Hardware Requirements

- CPU: AMD/Intel Processor with minimum CPU Frequency, 600 MHz
- Memory: RAM/Swap: 1024 MB or greater
- Internet: For installation, optionally, fetching results from Wikipedia
- Microphone: (Optional), for speech recognition

1.8.2 Software Requirements

- Linux / BSD / Darwin / Windows
- Python 3.8 (recommended, any version greater than 3.6)
- `pip`, preferably on `PATH`

1.9 Acknowledgements

Sugaroid AI has become possible to millions of open source developers. Particularly to mention, I would like to thank @GuntherCox for the `chatterbot` library and @explosion for spaCy , the machine learning library with which it was possible to make natural language processing easy as pie. Also, the millions of word collection on `en_core_web_sm`, `en_core_web_md` was contributed by developers across the globe for translation and linguistic differentiation. Special thanks to contributors, **Sreya Saju** (aka @sreyasaju) and **Joel Anil Chacko** (aka @TheDarkDrake) for helping me document the missed parts, bug triaging and adding more responses, I would also like to thank, Sugar Labs 2019 GCI Team, **Sashreek Magan** (aka @smag), **Andrea Gonzales** (aka @andreagon), **Zakiyah Hasanah** (aka @kiy4h), **Rishikesh Joshi** (aka @Creatune), **Szymon** (aka @sdziuda) and **Marcus Chong** (aka @pidddgy) for continuous testing on servers and reporting bugs. It is only possible to rectify bugs with the help of repeated **testing**. I would also like to thank friends and family who also helped me to work on this project. Along with this, I would like to extend gratitude to **Microsoft** for sponsoring Sugaroid's hosting on Azure.

1.10 Bibliography

1. **Jensen Shannon divergence**, *Wikipedia, the Free Encyclopedia* (en), available on web:

- wiki: https://en.wikipedia.org/wiki/Jensen%E2%80%93Shannon_divergence

2. **Naive Bayes Classifier**, *Wikipedia, the Free Encyclopedia* (en), available on web:

- wiki: https://en.wikipedia.org/wiki/Naive_Bayes_classifier

3. **Chatterbot, Machine learning, converstional bot**, *Gunthercox, et. al.*, available on web: https://chatterbot.readthedocs.io/en/stable/

4. **Google Speech Recognition for Python**, *PyPI: Python Packaging Index, et. al*, available on web:

- repository: https://pypi.org/project/SpeechRecognition

5. **spaCy · Industrial-strength Natural Language Processing**, *explosion.io, et. al*, available on web:

- website: https://spacy.io/,
- source code: GitHub

6. **Stanford Question Answer Dataset**, *Rajpurkar, Pranav, et. al*, available on web:

- research paper: https://arxiv.org/abs/1806.03822

CHAPTER

TWO

SUGAROID

2.1 launcher module

Create a new *Mock* object. *Mock* takes several optional arguments that specify the behaviour of the Mock object:

- *spec*: This can be either a list of strings or an existing object (a class or instance) that acts as the specification for the mock object. If you pass in an object then a list of strings is formed by calling dir on the object (excluding unsupported magic attributes and methods). Accessing any attribute not in this list will raise an *AttributeError*.

  If *spec* is an object (rather than a list of strings) then *mock.__class__* returns the class of the spec object. This allows mocks to pass *isinstance* tests.

- *spec_set*: A stricter variant of *spec*. If used, attempting to *set* or get an attribute on the mock that isn't on the object passed as *spec_set* will raise an *AttributeError*.

- *side_effect*: A function to be called whenever the Mock is called. See the *side_effect* attribute. Useful for raising exceptions or dynamically changing return values. The function is called with the same arguments as the mock, and unless it returns *DEFAULT*, the return value of this function is used as the return value.

  Alternatively *side_effect* can be an exception class or instance. In this case the exception will be raised when the mock is called.

  If *side_effect* is an iterable then each call to the mock will return the next value from the iterable. If any of the members of the iterable are exceptions they will be raised instead of returned.

- *return_value*: The value returned when the mock is called. By default this is a new Mock (created on first access). See the *return_value* attribute.

- *wraps*: Item for the mock object to wrap. If *wraps* is not None then calling the Mock will pass the call through to the wrapped object (returning the real result). Attribute access on the mock will return a Mock object that wraps the corresponding attribute of the wrapped object (so attempting to access an attribute that doesn't exist will raise an *AttributeError*).

  If the mock has an explicit *return_value* set then calls are not passed to the wrapped object and the *return_value* is returned instead.

- *name*: If the mock has a name then it will be used in the repr of the mock. This can be useful for debugging. The name is propagated to child mocks.

Mocks can also be called with arbitrary keyword arguments. These will be used to set attributes on the mock after it is created.

2.2 setup module

Create a new *Mock* object. *Mock* takes several optional arguments that specify the behaviour of the Mock object:

- *spec*: This can be either a list of strings or an existing object (a class or instance) that acts as the specification for the mock object. If you pass in an object then a list of strings is formed by calling dir on the object (excluding unsupported magic attributes and methods). Accessing any attribute not in this list will raise an *AttributeError*.

  If *spec* is an object (rather than a list of strings) then *mock.__class__* returns the class of the spec object. This allows mocks to pass *isinstance* tests.

- *spec_set*: A stricter variant of *spec*. If used, attempting to *set* or get an attribute on the mock that isn't on the object passed as *spec_set* will raise an *AttributeError*.

- *side_effect*: A function to be called whenever the Mock is called. See the *side_effect* attribute. Useful for raising exceptions or dynamically changing return values. The function is called with the same arguments as the mock, and unless it returns *DEFAULT*, the return value of this function is used as the return value.

  Alternatively *side_effect* can be an exception class or instance. In this case the exception will be raised when the mock is called.

  If *side_effect* is an iterable then each call to the mock will return the next value from the iterable. If any of the members of the iterable are exceptions they will be raised instead of returned.

- *return_value*: The value returned when the mock is called. By default this is a new Mock (created on first access). See the *return_value* attribute.

- *wraps*: Item for the mock object to wrap. If *wraps* is not None then calling the Mock will pass the call through to the wrapped object (returning the real result). Attribute access on the mock will return a Mock object that wraps the corresponding attribute of the wrapped object (so attempting to access an attribute that doesn't exist will raise an *AttributeError*).

  If the mock has an explicit *return_value* set then calls are not passed to the wrapped object and the *return_value* is returned instead.

- *name*: If the mock has a name then it will be used in the repr of the mock. This can be useful for debugging. The name is propagated to child mocks.

Mocks can also be called with arbitrary keyword arguments. These will be used to set attributes on the mock after it is created.

2.3 sugaroid package

2.3.1 Subpackages

sugaroid.backend package

Submodules

sugaroid.backend.sql module

Module contents

sugaroid.brain package

Submodules

sugaroid.brain.about module

sugaroid.brain.aki module

sugaroid.brain.areyou module

sugaroid.brain.assertive module

sugaroid.brain.because module

sugaroid.brain.brain module

sugaroid.brain.bye module

sugaroid.brain.canmay module

sugaroid.brain.constants module

sugaroid.brain.convert module

sugaroid.brain.covid module

sugaroid.brain.debug module

sugaroid.brain.dis module

sugaroid.brain.do module

sugaroid.brain.dolike module

sugaroid.brain.either module

sugaroid.brain.emotion module

sugaroid.brain.feel module

sugaroid.brain.fun module

sugaroid.brain.hangman module

sugaroid.brain.iam module

sugaroid.brain.idk module

Submodules

sugaroid.google.google module

Module contents

sugaroid.gui package

Submodules

sugaroid.gui.ux module

Module contents

sugaroid.platform package

Submodules

sugaroid.platform.darwin module

sugaroid.platform.linux module

sugaroid.platform.platform module

sugaroid.platform.windows module

Module contents

sugaroid.reader package

Submodules

sugaroid.reader.markdown module

sugaroid.reader.reader module

sugaroid.reader.rst module

sugaroid.reader.scrawled module

Module contents

sugaroid.trainer package

Submodules

2.3.2 Submodules

2.3.3 sugaroid.sugaroid module

2.3.4 sugaroid.train module

2.3.5 sugaroid.version module

2.3.6 Module contents

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX