Markovman

Generated by Doxygen 1.8.13

Contents

Chapter 1

Main Page

Implementation of markov chains for random text generation.

1.1 Description

Markovman is a program for random text generation based on markov chains. The generator is trained from a corpus. The only supported format for the corpus is as a text file, with dots '.' separating sentences.

1.2 Usage

The following is the interface as I plan to implement it, although it hasn't been written yet. The easiest way to use Markovman is to call it together with a corpus-file.

```
markovman path/to/corpus.txt
```

That will put the program in a loop, reading from stdin. You can pass the following commands:

```
gen N
```

will generate N sentences one after the other based on the corpus.

```
kill X
```

will make the word X disappear from the corpus.

```
exit
```

will exit the program

Another possibility is running the program like the following, which will generate N sentences and close immediately.

```
markovman path/to/corpus.txt -n N
```

See also

https://github.com/IanTayler/markovman.git

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 Markov Struct Reference

Struct that holds all the information relevant to a markov chain.

`#include <statemach.h>`

4.1.1 Detailed Description

Struct that holds all the information relevant to a markov chain.

The struct consists of:

- initlength: the number of words used at the beginning of a sentence.
- initpos: an array with all positions of the wordlist that hold initial words.
- wordlist: a list with all the words.

The documentation for this struct was generated from the following file:

- src/include/statemach.h

4.2 ThisMarkov Struct Reference

Data Fields

- int **initlength**
- int * **initpos**
- Word * **wordlist**

The documentation for this struct was generated from the following file:

- src/include/statemach.h

4.3 ThisWord Struct Reference

Data Fields

- char * **token**
- int * **freqlist**

The documentation for this struct was generated from the following file:

- src/include/statemach.h

4.4 Word Struct Reference

Struct for representing states in a first order Markov chain.

```
#include <statemach.h>
```

4.4.1 Detailed Description

Struct for representing states in a first order Markov chain.

The struct consists of:

- token: a pointer to the string representation of the word.
- freqlist: a pointer to an array of integers. Marks the frequency of each item in a corresponding wordlist.

The documentation for this struct was generated from the following file:

- src/include/statemach.h

Chapter 5

File Documentation

5.1 src/include/lexer.h File Reference

Definitions to handle a lexer.

Functions

- char * **get_next_token** (FILE *filedesc, char *endsymb)

5.1.1 Detailed Description

Definitions to handle a lexer.

Author

Ian G. Tayler

Date

14 May 2017 (creation)

Here we define a lexer for natural language strings. It works by recognising ' ', ',', '.', ':', ';', '!', '?', etc. as special characters that mark the end of a token. Special characters are then added to the token list as another stand-alone token.

Note

This file has the documentation for all exported functions and structs. For the documentation for internal-only aspects of the program, see lexer.c

See also

https://github.com/IanTayler/markovman.git

5.2 src/include/minunit.h File Reference

A very minimal unit test library.

Macros

- #define mu_assert(message, test) do { if (!(test)) return message; } while (0)

  Macro to assert equality in a unit test.
- #define mu_run_test(test)

  Macro to run a test.
- #define mu_assert_freeing(message, test, pointer) do { if (!(test)) { free(pointer); return message; } } while (0)

  Macro to assert equality in a unit test, freeing a pointer.
- #define mu_assert_running(message, test, block) do { if (!(test)) { block; return message; } } while (0)

  Macro to assert equality in a unit test, running a block if the test fails.

Variables

- int tests_run

  Global set to the amount of tests that ran.

5.2.1 Detailed Description

A very minimal unit test library.

Author

Jera Design

Date

Unknown

See also

http://www.jera.com/techinfo/jtns/jtn002.html

5.2.2 Macro Definition Documentation

5.2.2.1 mu_assert

```
#define mu_assert(
              message,
              test ) do { if (!(test)) return message; } while (0)
```

Macro to assert equality in a unit test.

This macro checks whether 'test' is a true value. If it is, then the macro does nothing. Otherwise, it will pass a message as the return value of the function in which the macro will be expanded.

Parameters

message	This message will be the return value of whichever function implements mu_assert. It should be a message to be sent if the assertion fails.
test	This is the value being asserted. It should evaluate to a true value in successful tests.

Note

MinUnit macro.

5.2.2.2 mu_assert_freeing

```
#define mu_assert_freeing(
              message,
              test,
              pointer ) do { if (!(test)) { free(pointer); return message; } } while (0)
```

Macro to assert equality in a unit test, freeing a pointer.

This macro behaves very similarly to mu_assert(), the main difference being that this macro frees a pointer passed as the first argument of the macro before returning.

Parameters

message	This message will be the return value of whichever function implements mu_assert. It should be a message to be sent if the assertion fails.
test	This is the value being asserted. It should evaluate to a true value in successful tests.
pointer	This pointer will be freed before returning if the test fails.

Note

This macro was defined by Ian Tayler in 2017 and doesn't belong to MinUnit

5.2.2.3 mu_assert_running

```
#define mu_assert_running(
              message,
              test,
              block ) do { if (!(test)) { block; return message; } } while (0)
```

Macro to assert equality in a unit test, running a block if the test fails.

Like mu_assert_freeing(), but running a full block of code instead of just freeing a pointer before returning.

Note

This macro was defined by Ian Tayler in 2017 and doesn't belong to MinUnit

5.2.2.4 mu_run_test

```
#define mu_run_test(
               test )
```

Value:

```
do { char *message = test(); tests_run++; \
                             if (message) return message; } while (0)
```

Macro to run a test.

This macro is used to run a 'test' function, which should return 0 if everything is alright. This macro should be included in functions with a ∗char return type.

Parameters

test	A pointer to a function that resturns 0 if everything is alright and a message (∗char) if there's an error.

Note

MinUnit macro.

5.2.3 Variable Documentation

5.2.3.1 tests_run

```
int tests_run
```

Global set to the amount of tests that ran.

This variable gets increased when mu_run_test runs, and it should hold the amount of tests ran at the end of the test program.

See also

mu_run_test

5.3 src/include/statemach.h File Reference

Header for state machines.

Data Structures

- struct ThisWord
- struct ThisMarkov

Typedefs

- typedef struct ThisWord **Word**
- typedef struct ThisMarkov **Markov**

Functions

- Markov * **induce_markov** (FILE *filedesc)

5.3.1 Detailed Description

Header for state machines.

Author

Ian G. Tayler

Date

13 May 2017 (creation)

This is the file where all the important definitions are. We define the struct 'Word' and a few functions for handling it. We also define the 'Markov' struct. That covers most of the program's logic.

Note

This file has the documentation for all exported functions and structs. For the documentation for internal-only aspects of the program, see lexer.h

See also

https://github.com/IanTayler/markovman.git

5.4 src/lib/lexer.c File Reference

Implementation of a lexer.

```
#include <stdio.h>
#include <stdlib.h>
#include "lexer.h"
```

Macros

- #define BASEBUFFSIZE 8 /* Preferably set it to a power of 2. */

Functions

- size_t **append_char** (char ∗∗token, char appc, int pos, size_t size)
- char ∗ **get_next_token** (FILE ∗filedesc, char ∗endsymb)

5.4.1 Detailed Description

Implementation of a lexer.

Author

Ian G. Tayler

Date

14 May 2017 (creation)

Here we implement a lexer for natural language strings. It works by recognising ' ', ',', '.', ':', ';', '!', '?', etc. as special characters that mark the end of a token. Special characters are then added to the token list as another stand-alone token.

Note

This file has the documentation for all *internal* (i.e. unexported) functions and structs. For the documentation of the API, see lexer.h

See also

https://github.com/IanTayler/markovman.git

5.4.2 Macro Definition Documentation

5.4.2.1 BASEBUFFSIZE

```
#define BASEBUFFSIZE 8 /* Preferably set it to a power of 2.  */
```

This is the the minimum size we'll allocate for our tokens. Smaller values mean the program will use less memory, but it will be slower when it has to allocate larger strings.

5.5 src/lib/statemach.c File Reference

Implementation of state machines.

```
#include <stdio.h>
#include <stdlib.h>
#include "lexer.h"
#include "statemach.h"
```

Macros

- #define INITWORD -1

  A constant that's used to mark that there is no 'previous' word.
- #define BASELEXSIZE 512

  A constant that defines how initial allocation of words will be.
- #define BASEINITWORDS 32

  A constant that defines how many int-s we will allocate initially for our list of initial words.

5.5.1 Detailed Description

Implementation of state machines.

Author

Ian G. Tayler

Date

13 May 2017 (creation)

This is the file where all the important definitions are. We define the struct 'Word' and a few functions for handling it. We also define the 'Markov' struct. That covers most of the program's logic.

Note

This file has the documentation for all *internal* (i.e. unexported) functions and structs. For the documentation of the API, see statemach.h

5.5.2 Macro Definition Documentation

5.5.2.1 BASEINITWORDS

```
#define BASEINITWORDS 32
```

A constant that defines how many int-s we will allocate initially for our list of initial words.

A constant that defines how many int-s we will allocate initially for our list of position in the wordlist array that hold pointers to words that have appeared at the beginning of a sentence.

It should normally be set to a power of 2 to optimize malloc and realloc in certain implementations.

5.5.2.2 BASELEXSIZE

```
#define BASELEXSIZE 512
```

A constant that defines how initial allocation of words will be.

It should normally be set to a power of 2 to optimize malloc and realloc in certain implementations.

5.5.2.3 INITWORD

```
#define INITWORD -1
```

A constant that's used to mark that there is no 'previous' word.

When we're inducing a markov chain, we need to save in a variable an int that marks where in the wordlist we can find a pointer to the previous word. When we're on the first word, we save INITWORD in that variable instead, which is defined to be different from all natural numbers.

5.6 src/markovman.c File Reference

The main file, where the interface is implemented.

```
#include <stdio.h>
#include "statemach.h"
```

Macros

- #define VERSION "0.3.0"

  String constant holding the current version of Markovman.

Functions

- int **main** (void)

5.6.1 Detailed Description

The main file, where the interface is implemented.

Author

Ian G. Tayler

Date

13 May 2017 (creation)

See also

https://github.com/IanTayler/markovman.git

Index