SpamBayes Documentation

Release 0.1

Alberto Franzin, Fabio Palese

December 12, 2012

CONTENTS

CHAPTER

ONE

MAIN MODULE

To launch the program, open a terminal, go to hell and type

python spam_bayes.py

Bye.

CHAPTER

TWO

THE BAYES NETWORK DEFINITION

2.1 The Naive Bayes class

class naive_bayes.**Bayes**

Contains the Bayes network and some possible operations: training, validation, k-fold cross-validation, formatted print of the data. For the other operations, instantiate the apposite classes.

__init__()

Constructor.

Initialize all the objects and variables used to define a Bayes network: words stats, overall stats, configuration, trainer, validator. Saves the path of the project.

_k_fold_cross_validation(*spam_list*, *ham_list*)

Internal method, execute the k-fold cross-validation TODO: FINISH

Splits the lists in the desidered number of parts (see config.Config object), then calls the trainer.Trainer.train() function.

Parameters

- **spam_list** (*array of str*) – the list of spam mails to be used;
- **ham_list** (*array of str*) – the list of ham mails to be used;

Returns the accuracy of the training.

bayes_print(*print_words*, *print_gen_stats*)

Prints out the data, padded for alignment.

Slightly adapted from http://ginstrom.com/scribbles/2007/09/04/pretty-printing-a-table-in-python/, many thanks.

Each row must have the same number of columns.

Parameters

- **print_words** (*bool*) – do I have to print the words retrieved?
- **print_gen_stats** (*bool*) – do I have to print the overall stats?

test_bayes()

Performs some test - needed to try some functions.

train()

Train the net. TODO: COMPLETE CROSS-VALIDATION

Read the mails given as training and validation set for spam and ham, then executes the proper training. Two methods are available: the direct training, and the k-fold cross-validation.

First of all, read the training set and validation set mails. If the k-fold cross-validation is chosen (see `config.Config` documentation), then call the apposite method, otherwise calls the `trainer.Trainer` object to extract from the training set the feature stats, then compute the accuracy by calling the `naive_bayes.Bayes.validate()` object, to find out the goodness of the classification.

No parameters are needed, since everything the network needs is already present. The location of the mails is (for now?) hardcoded here.

validate(*ham_val_list*, *spam_val_list*, *words*, *general_stats*, *config*)
Validation function.

Get the validation sets and the results of the training, and compute the accuracy of the classification of the mails in the validation set.

Parameters

- **ham_val_list** (*array of mails*) – the good mails of the validation set;
- **spam_val_list** (*array of mails*) – the spam mails of the validation set;
- **words** (array of `gen_stat.Word` objects) – the list of words read so far, and their stats;
- **general_stats** (associative array {str, `gen_stat.Stat`}) – the overall stats of the features;
- **config** (`config.Config` object) – contains some general parameters and configurations;

Returns accuracy of the validation.

2.2 The configuration options manager

class `config.`**Config**
Contains some general configurations.

The available parameters are (with *[default]* values):

•CROSS_VALIDATION (bool): True if k-fold cross-validation is chosen. False otherwise [True];

•CROSS_VALIDATION_FOLDS (int): the number of folds for cross-validation, if enabled [4];

•OVERALL_FEATS_SPAM_W (float, in [0,1]): the weight of the overall stats when computing the spamicity of a mail. The remaining part is given by the word stats [0.7];

•SHORT_THR (int): length of a word to be identified as *very short* [1];

•SIZE_OF_BAGS (int): number of ham and spam mails for training [50];

•SIZE_OF_VAL_BAGS (int): number of ham and spam mails for validation [10];

•SMOOTH_VALUE (int): smoothing value to be used in classification [1];

•SPAM_THR (float, in [0,1]): probability threshold to mark a mail as spam [0.95];

•VERBOSE (bool): if True, displays more messages [True];

•VERYLONG_THR (int): length of a word to be identified as *very long* [20].

__init__()
Constructor. Initialize all the parameters to their default value.

2.3 The training class

class `trainer.`**`Trainer`**
Trains the network, computing the stats for the main features and for the single words.

`__init__`()
Constructor.

`train`(*mails*, *is_spam*, *words*, *general_stats*, *config*)
The proper trainer method.

For all the mails given, extract the single words and classify them, calculating the overall stats for some interesting features to be evaluated, and for the single words.

Parameters

- **mails** (*array of str*) – the list of mails.
- **is_spam** (*bool*) – are the given mails spam?
- **words** (*array of Word objects*) – the array of stats for the single words detected.
- **general_stats** (array of {str, `gen_stat.Stat`}) – the overall stats of the set.
- **config** (`config.Config` object) – contains some configurations.

`trainer_print`(*general_stats*)
Print out the overall stats given. For test purposes.

Parameters **general_stats** (array of {str, `gen_stat.Stat`}) – the overall stats to be printed.

2.4 The classifier class

class `classifier.`**`Classifier`**
Classify the test set.

Apply Bayesian logic to classify the mails as spam or ham.

static **`classify`**(*mws*, *mgs*, *words*, *general_stats*, *config*)
TODO: INSERT DOCUMENTATION HERE!!!

CHAPTER

THREE

FEATURE STATISTICS MODULES

3.1 General stats for training sets

class gen_stat.**Stat**(*description*, *words_spam*, *words_ham*)

Stats for mail characteristics: how many times this feature appears in a spam mail, and how many times it appears in a ham mail. Class used when training the network.

__init__(*description*, *words_spam*, *words_ham*)

Constructor.

class gen_stat.**Word**(*spam_occurrences*, *ham_occurrences*)

Stats for a single word: how many times this word appears in a spam mail, and how many times it appears in a ham mail. Class used when training the network.

__init__(*spam_occurrences*, *ham_occurrences*)

Constructor.

3.2 General stats for validation and test sets

class test_stat.**Test_stat**(*description*, *count*)

Stats for a single mail belonging to the test set or to the validation set. So, it it not possible, at the stage this object is created, to tell whether the mail is spam or ham. Class used when validating and testing the network.

__init__(*description*, *count*)

Constructor. Initialize the stat.

class test_stat.**Test_word**(*occurrences*)

Stats for a single word: how many times this word appears in the parsed mail. Class used when validating and testing the network.

__init__(*occurrences*)

Constructor.

CHAPTER

FOUR

VARIOUS TOOLS AND UTILITIES

4.1 The lexical analyzer

class `lexer.`**`Lexer`**

Lexical Analyzer. Use Ply's lexer to identify the tokens and to classify them. See http://www.dabeaz.com/ply/ to know how it works.

`__init__``()`

Constructor: creates the *Ply* lexer and defines all the rules to identify and classify the tokens.

All the `t_TOKEN()` methods are defined as inner methods inside here.

`_process_tokens`(*results*, *in_training*, *is_spam*, *words*, *general_stats*, *config*)

Process tokens extracted from the training set.

For every token, extract the value (the word itself) and its type (lowercase word, title, link, etc), then update all the stats for the word and the mail.

Parameters

- **results** (*array of tokens*) – the list of tokens recognized;
- **in_training** – flag to tell if the lexing is performed during training (*True*) or during validation or testing (*False*). If we are performing the training step, then we know if the mail processed is ham or spam, and so we can fill appropriately the *general_stats* array of `gen_stat.Stat`, otherwise the array will be filled with `mail_stat.Mail_stat` objects;
- **is_spam** (*bool*) – flag to identify the mail as spam or ham (useless if *in_training* == *False*);
- **words** (array of `gen_stat.Word` objects) – the list of words read so far, and their stats;
- **general_stats** – the overall stats of the features. Feature type may be of two types: `gen_stat.Stat` (*in_training* == *True*), or `mail_stat.Mail_stat` (*in_training* == *False*);
- **config** (`config.Config` object) – contains some general parameters and configurations.

`lexer_words`(*text*, *in_training*, *is_spam*, *words*, *general_stats*, *config*)

Apply lexical analysis to the text of mails.

May

Parameters

- **text** (*str*) – the text of the mail to be parsed;

- **in_training** – flag to tell if the lexing is performed during training (*True*) or during validation or testing (*False*). If we are performing the training step, then we know if the mail processed is ham or spam, and so we can fill appropriately the *general_stats* array of `gen_stat.Stat`, otherwise the array will be filled with `mail_stat.Mail_stat` objects;
- **is_spam** (*bool*) – flag to identify the mail as spam or ham (useless if *in_training* == *False*);
- **words** (array of `gen_stat.Word` objects) – the list of words read so far, and their stats;
- **general_stats** – the overall stats of the features. Feature type may be of two types: `gen_stat.Stat` (*in_training* == *True*), or `mail_stat.Mail_stat` (*in_training* == *False*);
- **config** (`config.Config` object) – contains some general parameters and configurations.

4.2 Other utilities

class `utils.`**`Utils`**

Collection of various tools used in the project.

static `_read_files`(*path*, *how_many*, *read_mails*, *words*, *gen_stats*, *config*)

Read the desidered number of text files from the given path.

If desidered, extract first the text and then the tokens from the mails. Does nothing on the content of plain text files.

Parameters **path** – the relative path from the current position

to the desidered directory; :type path: str :param how_many: how many files to read. 0 == unlimited; :type how_many: int :param read_mails: tells if the user wants to read mails or plain text; :type read_mails: bool :param words: the list of words read so far, and their stats; :type words: array of `gen_stat.Word` objects :param general_stats: the overall stats of the features; :type general_stats: associative array {str, `gen_stat.Stat`} :param config: contains some general parameters and configurations; :type config: `config.Config` object :return: a list containing all the mails in the given files.

static `chunks`(*l*, *n*)

Yield successive n-sized chunks from l.

From http://stackoverflow.com/questions/312443 (thanks).

Parameters

- **l** (*list of objects*) – the list to be splitted;
- **n** (*int*) – the size of the generated chunks.

static `create_stats`()

Defines a new associative array of (str, Stat), containing all the overall stats to be evaluated by the Bayes network.

Returns the newly created array.

static `read_mails`(*path*, *how_many*, *words*, *general_stats*, *config*)

Read the desidered number of text files from the given path.

Calls method Utils._read_files, passing the same parameters received, with *read_mails* flag set to True.

Parameters

- **path** (*str*) – the relative path from the current position to the desidered directory;

- **how_many** (*int*) – how many files to read. 0 == unlimited;
- **words** (array of `gen_stat.Word` objects) – the list of words read so far, and their stats;
- **general_stats** (associative array {str, `gen_stat.Stat`}) – the overall stats of the features;
- **config** (`config.Config` object) – contains some general parameters and configurations;

Returns a list containing all the mails in the given files.

static `read_text` (*path*, *how_many*, *config*)

Read the desidered number of text files from the given path.

Calls method Utils._read_files, passing the same parameters received, with *read_mails* flag set to False.

Parameters

- **path** (*str*) – the relative path from the current position to the desidered directory;
- **how_many** (*int*) – how many files to read. 0 = unlimited;
- **config** (`config.Config` object;) – contains some general parameters and configurations;

Returns a list containing all the text in the given files.

PYTHON MODULE INDEX

INDEX