SpamBayes Documentation

Release 0.1

Alberto Franzin, Fabio Palese

December 23, 2012

CONTENTS

CHAPTER

ONE

INTRODUCTION

This document provides the documentation for the software written by Alberto Franzin and Fabio Palese as part of the examination for the course of Intelligent Systems (Sistemi Intelligenti), a.y. 2012/13, University of Padova, taught by prof. S. Badaloni and prof. F. Sambo.

Our project consists in writing a bayesian spam classifier, using the Naive Bayes approach. We have built a Bayes network that can be configurated, trained and used to perform a check on a set of mails to detect if these ones are spam or good mails, called, from now onwards, 'ham'.

The theory and the practice lying below the project is available in the report and in the slides associated with it. Here we provide instead a reference for the modules written by us, and the way to use them, just in case.

The whole code of the project is available at google code (see below).

1.1 Download and installation

To download the code, go to *http://code.google.com/etc* and download the source packet.

We suggest to use the svn repository available. To download the project, open a terminal, go to the chosen directory and type *comando svn*.

Python 2.7 is required. We have not performed any test with older versions, as well as newer ones, e.g. Python 3.0, so we cannot guarantee the correct working under these versions.

To successfully launch the program, you need to fullfill the following dependencies:

1. BeautifulSoup (from the bs4 module) (link) to extract the useful informations from the mail structure,
2. Ply (link) to extract and identify the tokens.

The documentation for these two modules is available on the respective sites.

The classifier expects to find the mail sets in the directory */path/of/the/project/spam/*, and, precisely:

- spam mails in */path/of/the/project/spam/spam/*
- ham mails in */path/of/the/project/spam/ham/*

SCRIVI DOVE TROVARE I TEST SET. FORSE È ANCHE IL CASO DI PERMETTERE DI CONFIGURARE DOVE TROVARE LA ROBA...

Warning: No checks are performed to verify the correctness of the settings. If the environment is inconsistent with respect to the specifications given here, the software may die at any moment during the execution.

1.2 Usage

To launch the program, from a terminal type *python /path/of/the/project/spam_bayes.py*.

CHAPTER

TWO

MAIN MODULE

CHAPTER

THREE

THE BAYES NETWORK DEFINITION

In these modules we have defined the actual Bayes network, and all its operations.

3.1 The Naive Bayes class

The `naive_bayes` module provides the `naive_bayes.Bayes` class, which contains the informations and the methods needed to perform training, validation and testing.

Its main variables are the arrays of stats of the words and the features filled during the training. These arrays, respectively of type {str, `gen_stat.Word`} and {str, `gen_stat.Stat`}. This class provides also the methods to train and validate the network.

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.

Methods contained: __init__ bayes_print _k_fold_cross_validation train validate test_bayes

__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?

check()
Compute accuracies for validation and testing.

read_bayes()
Read the overall stats computed in a previous run from files, and then load validation and test set.

Three files wil be created:

1.*ID_words.csv*, containing the stats of the words;

2.*ID_feats.csv*, containing the stats of the features;

3.*ID_params.csv*, containing the configuration used,

where *ID* is the value contained in *params['OUTPUT_ID']*.

Validation and test sets are loaded from the same place they are expected to be when training "normally".

test_bayes()
Performs some test - needed to try some functions.

train()
Train the net.

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.

update_stats(*ws*, *gs*, *is_spam*)
Update the overall stats, using the stats computed for a single mail and its status.

Parameters

- **ws** (array of `test_stat.Test_word`) – the list of words found in the mail;
- **gs** (array of `test_stat.Test_stat`) – the list of features found in the mail;
- **is_spam** (*bool*) – *True* if the mail is spam, *False* otherwise.

validate(*ham_list*, *spam_list*, *words*, *general_stats*)
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;

Returns accuracy of the validation.

write_bayes()
Write the overall stats computed to file.

Three files wil be created: 1. *ID_words.csv*, containing the stats of the words; 2. *ID_feats.csv*, containing the stats of the features; 3. *ID_params.csv*, containing the configuration used, where *ID* is the value contained in *params['OUTPUT_ID']*.

3.2 The configuration options manager

This is the module providing the basic configurations which allow the user to customize the behaviour of the software.

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];
- SIZE_OF_TEST_BAG (int): number of mails in the test set [30]
- 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.

cprint()
Print all the parameters and their assigned value.

get_params()
Return the parameter list.

3.3 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*, *params*)
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.
- **params** (*associative array*) – contains some general parameters and 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.

3.4 The classifier class

class `classifier.`**Classifier**
Classify an item.

This class contains the **:method:‘classifier.Classifier.classify‘** used to assign a class (spam/ham) to a mail, using the statistics computed for the processed mail, and the statistics of the training set.

static classify (*ws*, *gs*, *ovrl_ws*, *ovrl_gs*, *params*)
Classification function which guesses the class of a mail. The Bayesian theory is applied here.

The method iterates through all the words identified in the mail, and for each one computes how much likely it is for the word to belong to a spam mail or to a ham mail. Then it does the same for each general feature of the mail. Finally, the method combines the two results and tells which class the mail is more likely to be.

The statistics computed for each mail will be used to update the general properties tables, based on the the class computed here.

The method relies on the correct tuning of the parameters contained in the `config.Config` class or set by the user.

Parameters

- **ws** (array of `test_stat.Test_word` objects) – the list of words of the mail to be classified, and their stats;
- **gs** (array of `test_stat.Test_stat` objects) – array containing the features encontered in the mail;
- **params** (*associative array*) – contains some general parameters and configurations;

Returns *True* if the mail is classified as spam, *False* if it is considered ham.

CHAPTER

FOUR

FEATURE STATISTICS MODULES

When counting the statistics of the mails, we fall into one of the following two cases:

1. we are training the network, so we process the mails knowing their "spamminess" status. In this case we are working with both spam and ham mails in 'parallel', and we need to keep track of how many times a certain feature appears in spam mails, and how many times the same feature appears in ham mails;
2. we are validating the configuration, or discovering the status of a mail (a set of mails), so we can only count the features found. Of course, since a mail belongs only to one and only one (unknown, so far) class, we need only one value for each feature tracked.

To meet this requirement, we provide two different modules. They are very similar, since they have the same purpose, but are used in different situations.

4.1 General stats for training sets

The *gen_stat* module contains the general stats for the training step. The two classes contained are:

1. `Stat`, containing the number of featured found in both the spam and ham sets, and
2. `Word`, containing the number of times the word has been found in both the spam and ham sets.

Both the classes contain only the constructor, to initialize the variables, which are public and may be modified directly as needed.

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.

4.2 General stats for validation and test sets

The *test_stat* module contains the general stats for the validation and testing step, when the status of the mail is unknown. The two classes contained are:

1. Test_stat, containing the number of featured found in the mail or in the set, and
2. Test_word, containing the number of times the word has been found in the mail or in the set.

Since the purpose of these classes is the same of the ones in the gen_stat module, also in these module both the classes contain only the constructor.

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. It is probably useless, but it keeps some "simmetry" with the one used in training.

__init__(*occurrences*)
Constructor.

CHAPTER

FIVE

VARIOUS TOOLS AND UTILITIES

5.1 The lexical analyzer

This is the module containing the lexical analyzer, based on Ply lexer.

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*, *params*)

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. If the analyzed mail belongs to a training set, then the stats are updated according to the (known) class of the mail. Otherwise, the stats cannot be associated to any class, since this is yet to be detected.

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 `test_stat.Test_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 `test_stat.Test_stat` (*in_training* == *False*);
- **params** (*associative array*) – contains some general parameters and configurations;

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

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 `test_stat.Test_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 `test_stat.Test_stat` (*in_training* == *False*);
- **params** (*associative array*) – contains some general parameters and configurations;

5.2 Other utilities

All the generic purpose methods used in different locations are placed in the `utils.Utils` class. Namely, these methods are used to:

- read text from the files found in a given location. In particular, these methods can read text in mail format;
- split a list in a given numbers of equally long lists (the last list may be shorter than the previous ones);
- build an empty array containing the overall stats for the training set, thus discriminating the features found in the spam mails from the ones found in ham mails;
- build an empty array containing the overall stats from a generic mail, without knowing its class (its status, ham or spam).

All these methods are static, so have to be invoked using the *Utils.method()* syntax.

class `utils.`**`Utils`**
Collection of various tools used in the project.

static `_read_files`(*path*, *how_many*, *read_mails*, *words*, *gen_stats*, *params*)
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** (*str*) – the relative path from the current position to the desidered directory;
- **how_many** (*int*) – how many files to read. 0 == unlimited;
- **read_mails** (*bool*) – tells if the user wants to read mails or plain text;
- **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;
- **params** (*associative array*) – contains some general parameters and configurations;

Returns 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_file` (*file_name*)
Creates an empty file.

static `create_stats` ()
Defines a new associative array of (str, `gen_stat.Stat`), containing all the overall stats to be evaluated by the Bayes network in the training step.

Returns the newly created array.

static `create_test_stats` ()
Defines a new associative array of (str, `test_stat.Stat`), containing all the overall stats to be evaluated by the Bayes network in the validation and testing steps.

Returns the newly created array.

static `merge_lists` (*lists*)
Merge a list of lists into a single one.

From http://stackoverflow.com/questions/406121 (thanks)

Parameters **lists** (*list*) – the list of lists to be flattened;

Returns the new list.

static `read_mails` (*path*, *how_many*, *words*, *general_stats*, *params*)
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;
- **params** (*associative array*) – contains some general parameters and configurations;

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

static `read_text` (*path*, *how_many*, *params*)
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;
- **params** (*associative array*) – contains some general parameters and configurations;

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

PYTHON MODULE INDEX

INDEX

Symbols

B

C

G

L

M

N

R

S

T

U

V

W