Sally Version 0.6.3
— User Manual —

Konrad Rieck

August 22, 2011

Contents

1 NAME

sally – a tool for embedding strings in vector spaces

2 SYNOPSIS

sally [**options**] [**-c** *config*] *input output*

3 DESCRIPTION

sally is a small tool for mapping a set of strings to a set of vectors. This mapping is referred to as embedding and allows for applying techniques of machine learning and data mining for analysis of string data. **sally** can applied to several types of string data, such as text documents, DNA sequences or log files, where it can handle common formats such as directories, archives and text files of string data.

The embedding of strings is carried out incrementally, where **sally** first loads a chunk of strings from *input*, computes the mapping to vectors and then writes the chunk of vectors to *output*. The configuration of this process, such as the input format, the embedding setting and the output format, are specified in the file *config* and additionally using command-line options.

```
.---------.                           .----------.
|   dir   |                           |   text   |
|   arc   |   \   .---------.   /     |  libsvm  |
|  lines  |   --  |  Sally  |  --     |  matlab  |
|   ...   |   /   '---------'   \     |   ...    |
'---------'                           '----------'
  Input          Embedding             Output
```

sally implements a standard technique for mapping strings to a vector space that is often referred to as vector space model or bag-of-words model. The strings are characterized by a set of features, where each feature is associated with one dimension of the vector space. The following types of features are supported by **sally**: bytes, words, n-grams of bytes and n-grams of words.

sally proceeds by counting the occurrences of the specified features in each string and generating a sparse vector of count values. Alternatively, binary or TF-IDF values can be computed and stored in the vectors. **sally** then normalizes the vector, for example using the L1 or L2 norm, and outputs it in a specified format, such as plain text, LibSVM, Cluto or Matlab format.

The processing chain of **sally** makes use of sparse data structures, such that high-dimensional vectors with millions of dimensions can be handled efficiently. All features are indexed by a hash function, where the bit width of this function can be adjusted according to the characteristics of the data. In several settings, a hash width between 24 and 28 bit already suffices to store millions of features with little loss of accuracy.

4 CONFIGURATION

The configuration of **sally** is specified in a configuration file. This file is structured into the three sections **input**, **features** and **output**, which define the parameters of the input format, the feature extraction and the output format, respectively.

All parameters of the configuration can be also be specified on the command-line. That is, if a parameter is specified in the configuration as **xx = ″yy″**, it can be alternatively supplied as a command-line option by **–xx ″yy″**.

Input formats

sally supports different formats for reading data sets of strings, which may range from plain files to directories and other structured resources. Following is a list of supported input formats.

input = {

input_format = ″lines″;
This parameter specifies the input format.

″dir″
The input strings are available as binary files in a directory and the name of the directory is given as *input* to **sally**. The suffixes of the files are used as labels for the extracted vectors.

″arc″
The input strings are available as binary files in a compressed archive, such as a zip or tgz archive. The name of the archive is given as *input* to **sally**. The suffixes of the files are used as labels for the extracted vectors.

″lines″
The input strings are available as lines in a text file. The name of the file is given as *input* to **sally**. No label information is supported by this input format. The lines need to be separated by newline and may not contain the NUL character.

″fasta″
The input strings are available in FASTA format. The name of the file is given as *input* to **sally**. Labels are extracted from the description of each sequence using a regular expression. Comments are allowed if they are preceded by either ′;′ or ′>′.

chunk_size = 256;
To enable an efficient processing of large data sets, **sally** processes strings in chunks. This parameter defines the number of strings in one of these

chunks. Depending on the lengths of the strings, this parameter can be adjusted to balance loading and processing of data.

fasta_regex = " (\\+|-)?[0-9]+";

The FASTA format allows to equip each string with a short description. In several data sets this description contains a numerical label which can be useful in supervised learning tasks. The parameter is a regular expression which can be used to match numerical labels, such as +1 and -1.

lines_regex = "^(\\+|-)?[0-9]+";

If the strings are available as text lines, the parameter can be used to extract a numerical label from the start of the strings. The parameter is a regular expression matching labels, such as +1 and -1.

};

Features

The strings loaded by **sally** are characterized by a set of features, where each feature is associated with one dimension of the vector space. Different types of features can be extracted by changing the following parameters.

features = {

ngram_len = 2;

ngram_delim = "%0a%0d";

The parameter **ngram_len** specifies the numbers of consecutive words that are considered as one feature, while the parameter **ngram_delim** defines characters for delimiting these words. The characters can be either specified as bytes or as hexadecimal numbers prefixed by "%". The following types of features can be extracted using these parameters:

words

The strings are partitioned into substrings (words) using a set of delimiter characters. Such partitioning is typical for natural language processing, where the delimiters are usually defined as white-space and punctuation symbols. An embedding using words is selected by defining a set of delimiter characters (**ngram_delim**) and setting the n-gram length to 1 (**ngram_len**).

byte n-grams

The strings are characterized by all possible byte sequences of a fixed length n (byte n-grams). These features are frequently used if no information about the structure of strings is available, such as in bioinformatics or computer security. An embedding using byte n-grams is selected by defining the n-gram length (**ngram_len**) and setting the delimiters to an empty string (**ngram_delim**).

word n-grams

The strings are characterized by all possible word sequences of a fixed length n (word n-grams). These features require the definition of a set of delimiters and a length n. They are often used in natural language processing as a coarse way for capturing structure of text. An embedding using word n-grams is selected by defining a set of delimiter characters (**ngram_delim**) and choosing an n-gram length (**ngram_len**).

vect_embed = "bin";

This parameter specifies how the features are embedded in the vector space. Supported values are "bin" for associating each dimension with a binary value, "cnt" for associating each dimension with a count value and "tfidf" for using a TF-IDF weighting.

vect_norm = "none";

The feature vectors extracted by **sally** can be normalized to a given vector norm. Supported norms are "l1" for the Taxicab norm (L1) and "l2" for the Euclidean norm (L2). Normalization is often useful, if the lengths of the strings varies significantly and thus vectors differ just due to the number of extracted features.

hash_bits = 22;

sally uses a hash function to map individual features to dimensions in the vector space. This parameter specifies the number of bits used by this hash function and defines the maximum dimensionality of the vector space with 2 ** bits. As the embedding of **sally** is usually sparse and only a small fraction of dimensions is non-zero, 20 to 24 bits are often sufficient for representing the data. If this parameter is chosen too small, the embedding may suffer from collisions of features in the vector space. If it is chosen too large, several application may choke from the vast amount of dimensions.

explicit_hash = 0;

For performance reasons **sally** maps features to dimensions without memorizing the n-grams associating with these features. For certain analysis tasks, however, it may be necessary to retrieve a full mapping including the n-grams of each dimension. If this parameter is enabled **sally** keeps track of all n-grams and depending on the selected output format stores them with the extracted feature vectors.

tfidf_file = "tfidf.fv";

This parameter specifies a file to store the TF-IDF weights. If the embedding **tfidf** is selected, **sally** first checks if the given file is present. If it is not available, TF-IDF weights will be computed from the input data and stored to this file, otherwise the weights will be read from the file. Keeping a separate file for TF-IDF weights allows for computing the weighting for a data set, say the training set, and applying the exact same weighting to further data sets.

};

Output formats

Once the input strings have been embedded in a vector space, **sally** stores the resulting vectors in one of several common formats, which allows for applying typical tools of statistics and machine learning to the data, for example, Matlab, Octave, Shogun, Weka, SVMLight and LibSVM.

output = {

output_format = "libsvm";

Following is a list of output formats supported by **sally**:

"libsvm"
The feature vectors of the embedded strings are stored in the common libsvm format, which is supported by Shogun, SVMLight and LibSVM. The name of the output file is given as *output* to **sally**.

"text"
The feature vectors of the embedded strings are stored as a plain text using a list of dimensions, features and respective values. Almost all comprehensive analysis tools can deal with this type of format. The name of the output file is given as *output* to **sally**.

"matlab"
The feature vectors of the embedded strings are stored in matlab format (v5). The vectors are stored as a 1 x n struct array with the fields: data, src, label and feat. The name of the output file is given as *output* to **sally**. Note that great care is required to efficiently operate with sparse vectors in Matlab. If the sparse representation is lost during computations, excessive run-time and memory requirements are likely.

"cluto"
The feature vectors of the embedded strings are stored as a sparse matrix suitable for the clustering tool Cluto. The first line of the file is a header for Cluto, while the remaining lines correspond to feature vectors. The name of the output file is given as *output* to **sally**. Note that Cluto can not handle arbitrarily large vector spaces and thus the **"hash_bits"** should be set to values below 24.

};

5 OPTIONS

The configuration of **sally** can be refined and altered using several command-line options. In particular, all parameters of the configuration can be specified on the command-line. That is, if a parameter is specified as **xx = "yy"** in the configuration file, it can be changed by using the command-line option **–xx "zz"**. Following is a list of common options:

Basic options

```
  -i, --input_format <format>   Set input format for strings.
  -o, --output_format <format>  Set output format for vectors.
  -c, --config_file <file>      Set configuration file.
  -v, --verbose                 Increase verbosity.
  -q, --quiet                   Be quiet during processing.
  -V, --version                 Print version and copyright.
```

Feature options

```
  -n, --ngram_len <num>         Set length of n-grams.
  -d, --ngram_delim <delim>     Set delimiters of words.
  -E, --vect_embed <embed>      Set embedding mode.
  -N, --vect_norm <norm>        Set normalization mode.
  -b, --hash_bits <num>         Set number of hash bits.
```

6 FILES

/etc/sally.cfg

The system wide configuration file of **sally**. See the configuration section for further details.

7 EXAMPLES

Examples on how to use **sally** in different applications of data analysis and machine learning are available at http://www.mlsec.org/sally/examples.html.

8 COPYRIGHT

Copyright (c) 2010-2011 Konrad Rieck (konrad@mlsec.org)

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version. This program is distributed without any warranty. See the GNU General Public License for more details. =cut