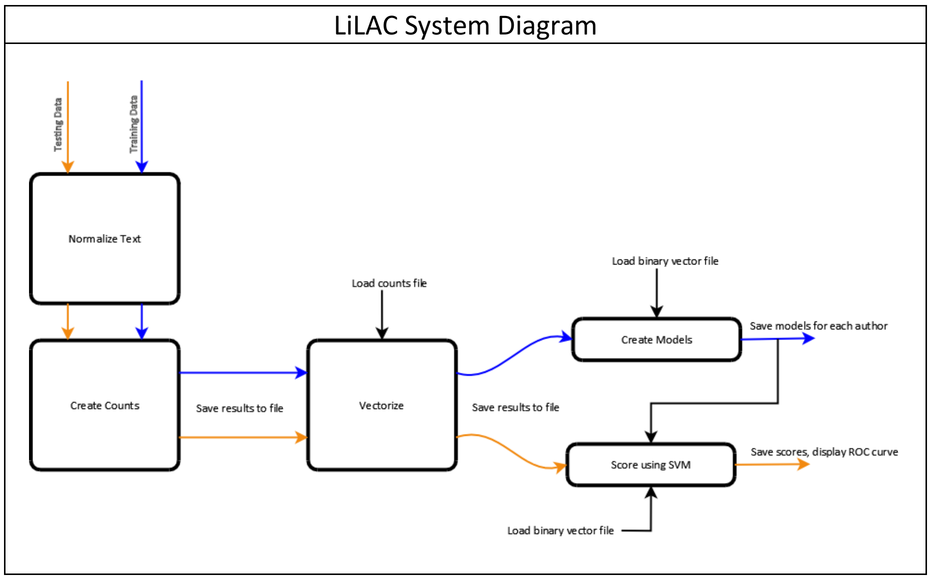
README

Contents

Overview

LiLAC (Lincoln Laboratory Author Classification) is an open source python module built to classify text by author. The program takes in documents as input, normalizes them, creates word count dictionaries, creates sparce, binary vectors from these counts, and then builds a model for each unique author in the dataset. An SVM and one-vs-rest strategy is used to classify documents. The tool has a modular structure so that at each stage of the process a file is produced. This enables the user to start or stop at a chosen point in the process. The block diagram below outlines the classification process.



This tool is a command-line application mainly suited for developers who would like to add author identification to their applications

Download

The code for the LiLAC (Lincoln Laboratory Author Classification is available on the github page at http://www.github.com/mitll/LiLAC.

Once you have downloaded the code, make sure that you have all the dependencies listed in section ***Dependencies***. Then you can get started with an example by following the ***Quick Start Guide***. For more details on the program see ***Running the Program***.

Dependencies

The system is a command-line application for 64-bit Linux written in Python. It requires the following dependencies:

- Python 2.7
- NumPy (Python module)
- SciPy (Python module)
- sklearn (Machine Learning Library for Python)
- NLTK (Natural Language Toolkit)

Target Platform

- 64-bit Linux

Quick Start Guide

Installation

To install LiLAC (2 methods):

1. Clone/download the repository to your computer from http://www.github.com/mitll/LiLAC. In your command line, navigate to the project directory, then install the package locally (for use on our system), with:

```
$ python setup.py install
```

OR,

2. Install the package with symlink, so that changes to the source files will be immediately available to other users of the package on your system:

```
$ python setup.py develop
```

Run an Example

To run the built-in example, from the main ll_author_id directory use:

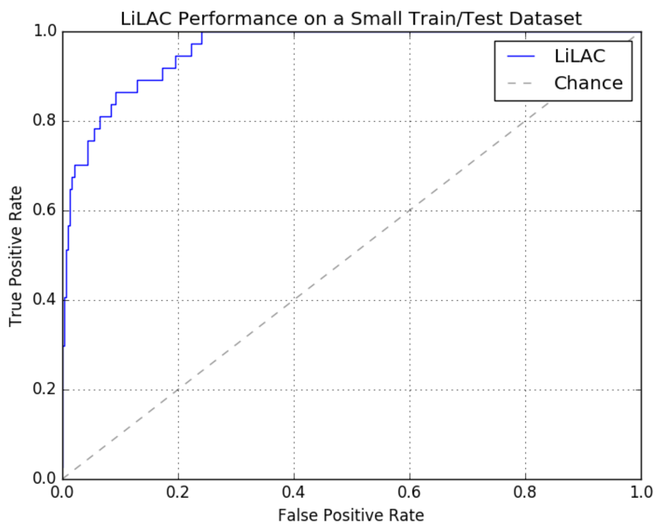
```
train_eg.py
```

This will train a model for each author in the training set. A list of documents in the training set can be found in `/lists/train.txt` The models can be found in `/models/models.dat` And, a list of the testing set can be found in `/lists/test.txt` The files referenced in these lists are found in the `small_corpus` directory.

For testing, from the main ll_author_id directory use:

```
test_eg.py
```

The resulting Receiver Operating Characteristic (ROC) curve should look like this:



Running the Program

Input

The program takes testing and training text data as input. The program reads requires a list of file paths to the training and testing data. The acceptable file formats are:

- .json
- .json.gz
- .txt
- .txt.gz

If using the .json file format, the expected input is of the form:

{"user id 1": ["<path to user id 1 file, file #1>"],

"user id 2": ["<path to user id 2 file, file #1>", "<path to user id 2 file, file #2>"]}

If using the .txt file format, the expected input is of the form:

tgt_author <user id 1>
<path to user id 1 file, file #1>
tgt_author <userid 2>
<path to user id 2 file, file #1>
<path to user id 2 file, file #2>

Output

The program outputs files during different stages of the evaluation process. This way a user can easily come back to any stage of the program and load saved data. File formats can be specified as either .json, .json.gz, .txt, or .txt.gz

A file is produced for each of the following:

- Word Count
- Common N-Gram (CNG) Dissimilarity Score calculation
- Sparse, binary vectors (in the correct format for training models)
- Models
- Final Scores

The final scores are reported in either .txt or .json format in the `/ll_author_id/temp` directory. For .txt files, the scores are reported as follows:

```
<document  #1>   <model #1>    <score>
<document #1>    <model #2>    <score>
<document #1>    <model #3>    <score>
<document #2>    <model #1>    <score>
<document #2>    <model #2>    <score>
<document #2>    <model #3>    <score>
```

And, for .json files, the score are reported as follows:

```
{“document #1”: {“model #1”: score,
                 “model #2”: score,
                 “model #3: score},
“document #2”: {“model #1”: score,
                 “model #2”: score,
                 “model #3: score}}
```

In addition, a Receiver Operating Characteristic (ROC) curve can be produced.

Options

For preprocessing the data, there are several options. Text normalization includes the following configuration options:

Text Normalization Options	Options	Function
utf8_to_ascii	True, False	Converts utf8 to ascii
remove_twitter	True, False	Removes @tags and retweet markers
remove_markup	True, False	Removes html style markers and html special characters
remove_nonsentenial	True, False	Removes non_sentenial punctuation Eg. _ , : , * , / , ; , etc.

Repeats and punctuation are automatically removed. For example, "coooooool!" will be normalized to "cool".

The methods for which counts are made also include configuration options:

Count Options	Options	Function
combine_same_user_counts	True, False	Combines word count dictionaries from all the files from a single author
format	.json, .json.gz, .txt, .txt.gz	Selects the output format for counts
count_separator	Default: '|'	Delimiter for separating counts

Choosing a model for scoring is also selected in the configuration.

Model Options	Options	Function
method	'svmtorch', 'sklearn'	Selects the model used in scoring

---- **Usage** --

To setup default config settings call the `get_default_config()`function. The default settings are as follows:

Text Normalization	Counts	Model
utf8_to_ascii: True remove_twitter: True remove_markup: True remove_nonsentenial: True	combine_same_user_counts: True format: '.json' count_separator: '|'	method: 'svmtorch', 'sklearn'

Modifying the default configurations :

Example:

```
config = get_default_config()
config['text_norm']['remove_twitter'] = False
config['counts']['format'] = '.json.gz'
config['model']['method'] = 'sklearn'
```

Useful Functions

`text_to_counts([`*`counts_filename`*`],[`*`training_filename_list`*`],[`*`config`*`],[`*`text_filter_function`*`], [`*`count_filter_funtion`*`]):`

Reads in the text from all users, normalizes the content, gets word counts, and saves to an output file.

Defaults: text_filter_function = neutral_text_filter, count_filter_function= neutral_count_filter

Location: `/ll_author_id/ll_author_id/get_counts.py`

`find_dict([`*`text_to_counts_output_file`*`], [`*`output_filename`*`], [`*`config`*`]):`

Computes Common N-Gram (CNG) dissimilarity scores from word count vector and saves the results to file.

Location: `/ll_author_id/ll_author_id/ngm_to_vec.py`

`ngm_to_vec([`*`text_to_counts_output_file`*`], [`*`find_dict_output_dictionary`*`], [`*`binary_output_filename`*`], [`*`class_label`*`], [`*`tags_filename`*`], [`*`config`*`], [`*`min_count`*`], [`*`min_words`*`], [`*`doc_tags`*`]):`

Uses the counts from `text_to_counts` and creates binary vectors and saves outputs to a file.

Defaults: min_count = 0, min_words = 0, doc_tags = True

Location: `/ll_author_id/ll_author_id/ngm_to_vec.py`

`train_models([`*`binary_filename`*`], [`*`output_directory`*`], [`*`config`*`], [`*`verbose`*`]):`

Creates model for each author and saves to .dat file in specified output directory.

Defaults: verbose =False

Location: `/ll_author_id/ll_author_id/train_models.py`

`score_all([`*`binary_filename`*`],[`*`model_directory`*`],[`*`training_tags_filename`*`], [`*`testing_tags_filename`*`], [`*`output_filename`*`], [`*`config`*`], [`*`verbose`*`]):`

Uses an SVM to score the documents and saves the output to a file.

Defaults: verbose=False

Location: `/ll_author_id/ll_author_id/score_models.py`

`roc_curve([`*`score_filename`*`], [`*`key_filename`*`]):`

Returns the false positive rate, true positive rate and a threshold

Location: `/ll_author_id/ll_author_id/evaulate.py`

For usage see the example scripts `train_eg.py` and `test_eg.py`.