PySS3: A Python package implementing a novel text classifier with visualization tools for Explainable AI

Sergio G. Burdisso[a,b], Marcelo Errecalde[a], Manuel Montes-y-Gómez[c]

[a] *Universidad Nacional de San Luis (UNSL), Ejército de Los Andes 950, San Luis, San Luis, C.P. 5700, Argentina*
[b] *Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), Argentina*
[c] *Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), Luis Enrique Erro No. 1, Sta. Ma. Tonantzintla, Puebla, C.P. 72840, Mexico*

Abstract

A recently introduced text classifier, called SS3, has obtained state-of-the-art performance on the CLEF's eRisk tasks. SS3 was created to deal with risk detection over text streams and therefore not only supports incremental training and classification but also can visually explain its rationale. However, little attention has been paid to the potential use of SS3 as a general classifier. We believe this could be due to the unavailability of an open-source implementation of SS3. In this work, we introduce PySS3, a package that not only implements SS3 but also comes with visualization tools that allow researchers deploying robust, explainable and trusty machine learning models for text classification.

Keywords: Text classification, XAI, SS3

1. Introduction

A challenging scenario in the machine learning field is the one referred to as "early classification". Early classification deals with the problem of classifying data streams as early as possible without having a significant loss in performance. The reasons behind this requirement of "earliness" could be diverse, but the most important and interesting case is when the classification delay has negative or risky implications. This scenario, known as "early risk detection" (ERD) have gained increasing interest in recent years with potential applications in rumor detection [1, 2, 3], sexual predator detection and aggressive text identification [4], depression detection [5, 6] or terrorism detection [7].

A recently introduced machine learning model for text classification [8], called SS3, has shown to be well suited to deal with ERD problems on social media streams. It obtained state-of-the-art performance on early depression, anorexia and self-harm detection on the CLEF eRisk open tasks [8, 9]. Unlike standard classifiers, this new classification model was specially created to naturally deal with ERD problems since: it supports incremental training and classification over text streams and it has the ability to visually explain its rationale. Moreover, SS3 introduces a classification model that does not require feature engineering and that is robust to the Class Imbalance Problem, which has become one of the most challenging research problems in the field [10].

However, little attention has been paid to the potential use of SS3 as a general classifier for document classification tasks. We think that one of the main reasons could be the fact that there exists no open-source implementation of SS3 available yet. We believe that the availability of open-source implementations is of critical importance to foster the use of new tools, methods and algorithms. On the other hand, Python is a popular programming language in the machine learning community thanks to its simple syntax, as well as a rich ecosystem of efficient open-source implementations of popular algorithms.

In this work, we introduce "PySS3" and share it with the community. PySS3 is an open-source Python package that implements SS3 and comes with two useful tools that allow working with it in a very simple, interactive and visual way. For instance, one of these tools provides post-hoc explanations using visualizations that directly highlight relevant portions of the raw input document which allows researchers to understand the model being deployed. Thus, PySS3 allows researches and practitioners deploying robust, explainable and trusty machine learning models for text classification.

2. Background

In this section, we provide an overview of the SS3 classifier. We will introduce only the general idea and basic terminology needed to better understand the PySS3 package. Readers interested in the formal definition of the model are invited to read Section 3 of [8].

Email addresses: sburdisso@unsl.edu.ar (Sergio G. Burdisso), merreca@unsl.edu.ar (Marcelo Errecalde), mmontesg@inaoep.mx (Manuel Montes-y-Gómez)

2.1. The SS3 classification model

SS3 is a novel supervised machine learning model for text classification. It was originally introduced in [8]. As described in more detail there, SS3 first builds a dictionary of words for each category during the training phase, in which word frequencies are stored. Then, using these word frequencies, and during the classification stage, it calculates a value for each word using a function $gv(w, c)$ to value words in relation to each category.[1] gv takes a word w and a category c and outputs a number in the interval [0,1] representing the degree of confidence with which w is believed to *exclusively* belong to c. For instance, suppose the categories are $technology, business$, and $food$, we could have:

$$gv(apple, tech) = 0.8; \quad gv(the, tech) = 0;$$
$$gv(apple, business) = 0.4; \quad gv(the, business) = 0;$$
$$gv(apple, food) = 0.75; \quad gv(the, food) = 0;$$

Additionally, $\overrightarrow{gv}$ is the vectorial version of gv. $\overrightarrow{gv}$ is only applied to a word and it outputs a vector in which each component is the word's gv value for each category. For instance, following the above example, we have:

$$\overrightarrow{gv}(apple) = (0.8, 0.4, 0.75)$$
$$\overrightarrow{gv}(the) = (0, 0, 0)$$

These vectors are called *confidence vectors*. Thus, in this example $(0.8, 0.4, 0.75)$ is the *confidence vector* of the word "apple", in which the first position corresponds to *technology*, the second to *business*, and so on.

2.1.1. Classification Process

The classification algorithm can be thought of as a 2-phase process. In the first phase, the input is split into multiple blocks (e.g. paragraphs), then each block is in turn repeatedly divided into smaller units (e.g. sentences, words). Thus, the previously "flat" document is transformed into a hierarchy of blocks. In the second phase, the $\overrightarrow{gv}$ function is applied to each word to obtain a set of word *confidence vectors*, which then are reduced to sentence *confidence vectors* by means of a word-level *summary operator*. This reduction process is recursively propagated up to higher-level blocks, using higher-level *summary operators*,[2] until a single *confidence vector*, $\overrightarrow{d}$, is generated for the whole input. Finally, the actual classification is performed by applying some policy based on the *confidence values* stored in $\overrightarrow{d}$ —for instance, selecting the category with the highest *confidence value* in $\overrightarrow{d}$.

It is worth mentioning that, using the *confidence vectors* obtained at different levels (words, sentences, paragraphs, etc.), it is quite straightforward to visually explain the classification process if different blocks of the input are colored in relation to their values. As will be described in section 3, this characteristic is exploited by the PySS3's "Live Test" tool to create interactive visualizations.

2.1.2. The Hyperparameters

The entire classification process depends on the gv function since it is used to create the first set of *confidence vectors* upon which higher-level *confidence vectors* are then created. As described in more detail in [8], the calculation of gv involves three functions, each controlled by a special hyperparameter, as follows:

$$gv(w, c) = lv_\sigma(w, c) \cdot sg_\lambda(w, c) \cdot sn_\rho(w, c)$$

where w is a word, c a category and:

- $lv_\sigma(w, c)$ values w based on its local frequency in c. The σ hyperparameter "smooths" the relationship between the raw frequency and the final value assigned to the word. For instance, $\sigma = 1$ indicates that lv is calculated directly proportional to the raw frequency, whereas smaller σ values decrease the influence of the frequency (i.e the gap between the most and less frequent words becomes smaller).[3]
- $sg_\lambda(w, c)$ captures the global significance of the word by decreasing the gv value in relation to its lv in the other categories. The λ hyperparameter controls how far $lv(w, c)$ must deviate from the other categories $lv(w, c_i)$ for w to be considered important to c.
- $sn_\rho(w, c)$ decreases the gv value in relation to the number of categories that w is significant to (given by sg). The ρ hyperparameter controls how sensitive/severe this sanction is.

3. PySS3

3.1. Software architecture

PySS3 is composed of one main module and three submodules. The main module is called "pyss3" and contains the implementation of the classifier *per se* in a class called "SS3". The $SS3$ class implements not only the "plain-vanilla" version of the classifier (introduced in [8]) but also different variants, such as the one introduced in [11] which allows SS3 to recognize important word n-grams "on the fly". Additionally, the $SS3$ class exposes a very clear API,

[1] gv stands for "global value".

[2] By default in PySS3, the *summary operators* are vector additions. However, PySS3 provides an easy way for the user to define his/her custom *summary operators*. More info on this can be found here: `https://pyss3.rtfd.io/en/latest/user_guide/ss3-classifier.html`

[3] This hyperparameter acts as if it were a "frequency tuner" that can be used to remove the overshadowing effect that extremely frequent words (but not important such as "the", "on", "with", etc.) produce on less frequent but really useful words.

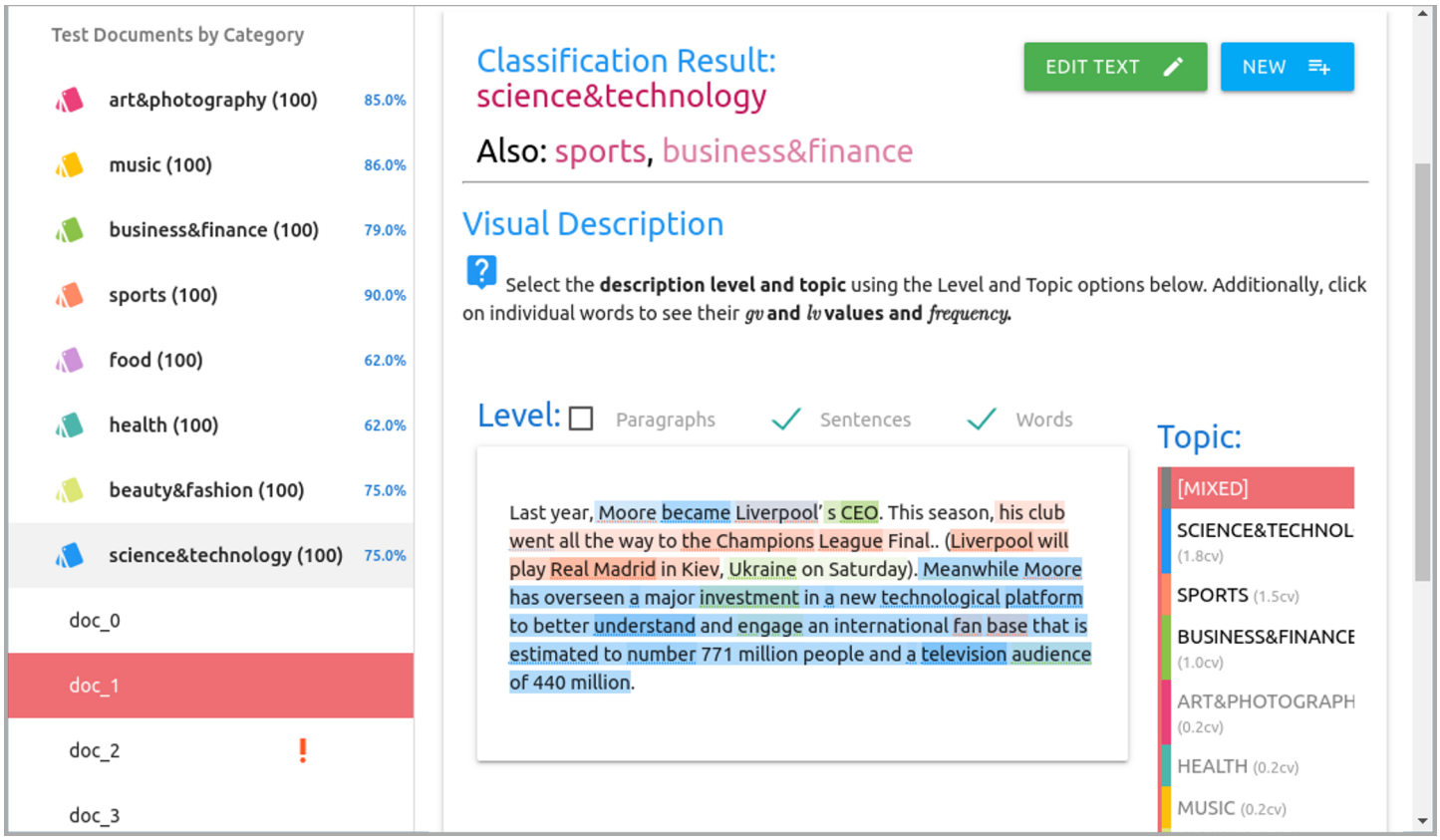

Figure 1: Live Test screenshot. On the left side, the list of test documents grouped by category is shown along with the percentage of success. In this screenshot, the "doc_2" document was misclassified and it is marked with an exclamation mark (!), easing error analysis. The user has selected the "doc_1", the "classification result" is shown above the visual description. In this figure, the user has chosen to display the visual explanation at sentence-and-word level, using mixed topics. For instance, the user can confirm that, apparently, the model has learned to recognize important words and that it has correctly classified the document. At sentence level, the user can see that the first sentence was considered to belong to multiple topics, then the topic is shifted to *sports* since the sentence is colored in orange, however, from the sentence that begins with "Meanwhile" on, the topic is mostly *techlology* and a little bit of *business* given by the words "investment" or "engage" colored in green. Note that the user is also able to edit the document text or even create a new one using the two buttons on the upper-right corner.

similar to that of Scikit-learn models,[4] as the reader will notice in the example shown in subsection 4.1. Finally, this module contains the following three submodules:

- *pyss3.server* — contains the implementations of the server for the "Live Test" tool, described in subsection 3.3. An illustrative example of its use is shown in subsection 4.2.
- *pyss3.cmd_line* — implements the "PySS3 Command Line" tool, described in subsection 3.3. This submodule is not intended to be imported and directly used with Python.
- *pyss3.util* — this submolude consists of a set of utility and helper functions and classes, such as classes for loading data from datasets, preprocessing text, etc.

[4]For instance, it has methods like "fit" for training and "predict" for classifying. Full list available in the API documentation: https://pyss3.rtfd.io/en/latest/api/index.html#pyss3.SS3.

3.2. Implementation platforms

PySS3 was developed using Python and was specially coded to be compatible with both, Python 2.7 and Python 3.x. In addition, it is also compatible with different operating systems, such as Linux, macOS and Microsoft Windows. To make sure this compatibility holds true when new code is added to the source code, we have configured and linked the PySS3 Github repository with the Travis CI service. This service automatically test PySS3 using different operating systems and versions of python when a new push is made to the repository.[5]

3.3. Software functionality

PySS3 is distributed via Python Package Index (PyPI) and therefore can be installed[6] simply by using the *pip*

[5]To monitor the compatibility state of the latest version of PySS3 online, visit `https://travis-ci.org/sergioburdisso/pyss3`

[6]For more details about installation, please refer to our on-line documentation: `https://pyss3.rtfd.io/en/latest/user_guide/installation.html`

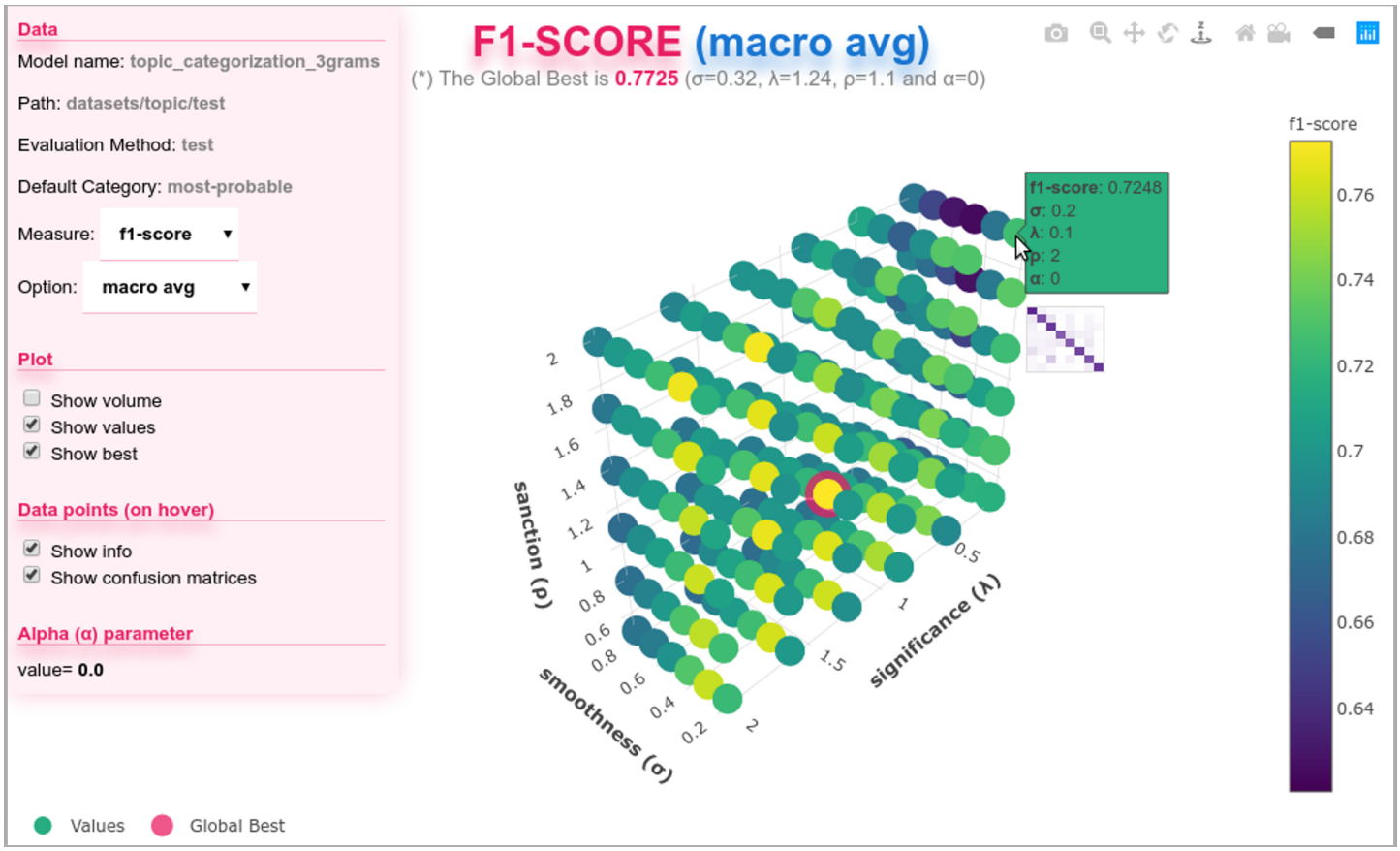

Figure 2: Evaluation plot screenshot. Each data point represents an evaluation/experiment performed using a particular combinations of hyperparameter values. Points are colored in relation to the obtained performance. The performance measure can be interactively changed using the options panel in the upper-left corner. Additionally, points with the global best performance are marked in pink. As shown in this figure, when the user move the cursor over a point, information related to that evaluation is displayed, including a small version of the obtained confusion matrix.

command as follows:

```
$ pip install pyss3
```

The package comes, in addition to the classifier, with two useful tools that allow working with SS3 in a very straightforward, interactive and visual way, namely the "Live Test" tool and the "PySS3 Command Line" tool.

The "Live Test" tool is an interactive visualization tool that allows the user to actively test his/her models. This tool can be lunched with a single line of python code using the *Server* class (see subsection 4.2 for an example). The tool provides a user interface (an screenshot is shown in Figure 1) by which the user can manually and actively test the model being developed using either the documents in the test set or just typing in his/her own. "Live Test" allows researchers analyzing and understanding what the models are actually learning since it gives an interactive and visual description of the SS3 classification process at different levels, word n-grams, sentences, and/or paragraphs. We highly recommend trying out the on-line live demos for Topic Categorization and Sentiment Analysis on Movie Reviews available at `http://tworld.io/ss3`.[7]

The "PySS3 Command Line" is an interactive command-line tool. This tool allows users to deploy SS3 models and interact with them through special commands for every stage of the machine learning pipeline (model selection, training, testing, etc.). Probably one of its most important features is the ability to automatically (and permanently) record the history of every evaluation that the user has performed (tests, k-fold cross-validations, grid searches, etc.). This allows the user to interactively visualize and analyze their classifiers performance in terms of its different hyperparameters values, as shown in subsection 4.3. This tool can be started from the operating-system command prompt using the "pyss3" command that is automatically added when installing the package.

[7]These two live test demos were built following the tutorials available in the documentation (`https://pyss3.rtfd.io/en/latest/user_guide/getting-started.html`).

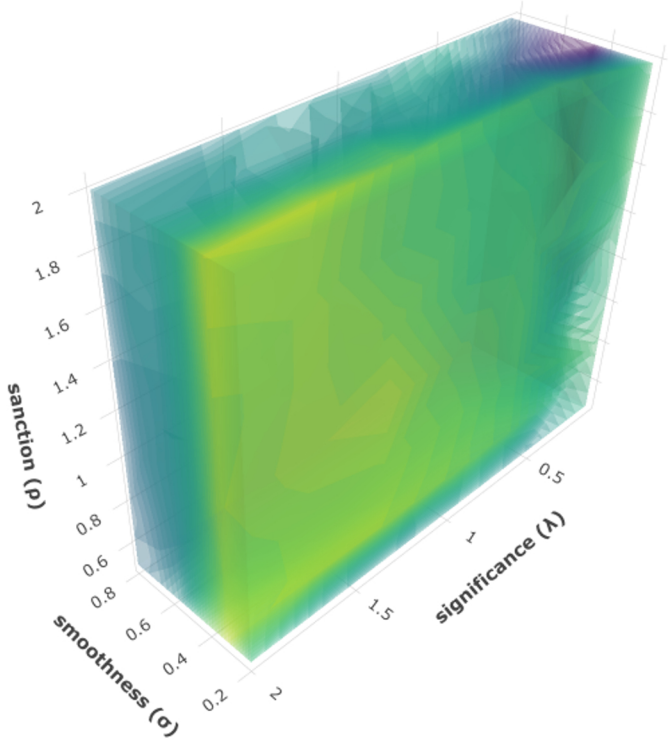

Figure 3: Evaluation plot - "show volume" option enabled.

4. Illustrative examples

In this section, we will introduce three simple illustrative examples. PySS3 provides two main types of workflow. In the classic workflow the user, as usual, imports the needed classes and functions from the package and then writes a python script to train and test the classifiers. In the other workflow, the whole machine learning pipeline is done using only the "PySS3 Command Line" tool which enables a faster way to develop models without coding in Python. Due to space limitation, we will not show examples of the first workflow here, however, for full working examples using both workflows please refer to the tutorials in the documentation (`https://pyss3.rtfd.io/en/latest/user_guide/getting-started.html#tutorials`).[8]

In the following examples, we will assume the user has already loaded the training and test documents and category labels, as usual, in the *x_train*, *y_train*, *x_test*, *y_test* lists, respectively. For instance, this could be done by using the *Dataset* class from the *pyss3.util* submodule, as follows:

```
from pyss3.util import Dataset

x_train,y_train = Dataset.load_from_files("path/to/train")
x_test, y_test = Dataset.load_from_files("path/to/test")
```

4.1. Training and test

This simple example shows how to train and test an SS3 model using default values.

```
from pyss3 import SS3

clf = SS3()
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)

print("Accuracy:", accuracy(y_pred, y_test))
```

The last line prints the obtained accuracy, we are assuming here that this *accuracy* function already exists.[9] Note that since SS3 creates a language model for each category, we do not need to create a document-term matrix, we are simply using the raw *x_train* and *x_test* documents for training and test, respectively.

4.2. Training and (live) test

This example is similar to the previous one, but instead of simply using *predict* and *accuracy* to measure our model's performance, here we are using the "Live Test" tool to visually analyze and test our model.

```
from pyss3 import SS3
from pyss3.server import Server

clf = SS3(name="my_model")
clf.fit(x_train, y_train)

Server.serve(clf, x_test, y_test)  # Live test!
```

Note that in the 4th line, we are naming our model "my_model". This is not required but was only added because we will use it in the following example.[10]

4.3. Hyperparameter optimization

This example shows how we could use the "PySS3 Command Line" tool to find better hyperparameter values for the model trained in the previous example.

[8] Readers interested in trying PySS3 out right away, we have created two Jupyter Notebooks for the tutorials which can be used to interact with PySS3 in an online live environment (`https://mybinder.org/v2/gh/sergioburdisso/pyss3/master?filepath=examples`).

[9] For instance, it was imported from *sklearn.metrics*.

[10] In practice, we would have also needed to add the line "clf.save_model()" to save the model to disk.

First, we need to make sure we are in the directory we are working on:

```
$ cd path/to/our/directory
```

To start the PySS3 Command Line tool, we enter the following command:

```
$ pyss3
```

Now we will load our previous model and perform hyperparameter optimization using the grid search method, as follows:

```
>>> load my_model
>>> grid_search path/to/test s r(.2,.8,6) l r(.1,2,6)
p r(.5,2,6)
```

Note that in PySS3, the hyperparameters are referenced using the s, l, and p letters for the σ, λ, and ρ hyperparameters, respectively. Additionally, in the *grid_search* command, we are using a special built-in function, $r(x0, x1, n)$, that returns a list of n float numbers between $x0$ and $x1$ (including both). Therefore, in this grid search, σ will take 6 different values between .2 and .8, λ between .1 and 2, and ρ between .5 and 2.

Once the grid search is over, we can use the following command to visualize the results:

```
>>> plot evaluations
```

This command will save an interactive 3D plot in a single and portable HTML file in the current directory and then will open it up in the web browser, an screenshot is shown in Figure 2.[11] We can see that the best hyperparameter values are $\sigma = 0.32$, $\lambda = 1.24$, and $\rho = 1.1$. Finally, to use this values with Python we could either replace the 4th line of the previous example with:

```
clf = SS3(s=0.32, l=1.24, p=1.1)
```

Or add the following line anywhere before testing:

```
clf.set_hyperparameters(s=0.32, l=1.24, p=1.1)
```

Note that, in addition to using this plot to obtain the best values, the user can use it to analyze (and better understand) the relationship between hyperparameter values and performance in the problem being addressed. For instance, if the "show volume" option is enabled from the options panel, the plot will turn into the plot shown in Figure 3. Roughly, we can see that the *smoothness* (σ) hyperparameter must always be around 0.35, the *sanction* (ρ) hyperparameter does not seem to have an impact[12], whereas the performance seems to improve as the *significance* (λ) gets bigger.[13]

[11] More info available in the documentation (`https://pyss3.rtfd.io/en/latest/user_guide/visualizations.html#evaluation-plot`).

Nr.	(executable) Software metadata description	
S1	Current software version	v0.3.9
S2	Permanent link to executables of this version	`https://pypi.org/project/pyss3/0.3.9/`
S3	Legal Software License	MIT License
S4	Computing platform/Operating System	Linux, OS X, and Microsoft Windows
S5	Installation requirements & dependencies	Pip, Python 2.7-3.x, Scikit-learn 0.20 or higher, Matplotlib
S6	Link to documentation	`https://pyss3.rtfd.io`
S7	Support email for questions	`sergio.burdisso@gmail.com`

Table A.1: Software metadata

5. Conclusions

We have briefly presented PySS3, an open-source Python package that implements SS3 and comes with useful development and visualization tools. This software should be helpful for researchers and practitioners who need to deploy explainable and trusty machine learning models for text classification.

Appendix A. Required metadata

Appendix A.1. Current executable software version

Table A.1 gives the information about the software release.

Appendix A.2. Current code version

Table A.2 describes the metadata about the source code of PySS3.

References

[1] J. Ma, W. Gao, Z. Wei, Y. Lu, K.-F. Wong, Detect rumors using time series of social context information on microblogging websites, in: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, ACM, 2015, pp. 1751–1754.

[2] J. Ma, W. Gao, P. Mitra, S. Kwon, B. J. Jansen, K.-F. Wong, M. Cha, Detecting rumors from microblogs with recurrent neural networks., in: IJCAI, 2016, pp. 3818–3824.

[12] Perhaps the categories are not so overlapping.

[13] It is worth mentioning that, using PySS3, researchers would be able to share this single file in their papers, which would be nicer for readers and would increase experimentation transparency. For instance, we have uploaded the file of the plot shown in Figure 2 for readers to interact with. It is available here: `https://pyss3.rtfd.io/en/latest/_static/eval_plot.html`

Nr.	Code metadata description	
C1	Current code version	v0.3.9
C2	Permanent link to code/repository used of this code version	`https://github.com/sergioburdisso/pyss3`
C3	Legal Code License	MIT License
C4	Code versioning system used	git
C5	Software code languages, tools, and services used	Python, Javascript, HTML
C6	Compilation requirements, operating environments & dependencies	Python 2.7-3.x, Scikit-learn 0.20 or higher, Matplotlib
C7	Link to developer documentation/manual	`https://pyss3.rtfd.io/en/latest/api`
C8	Support email for questions	`sergio.burdisso@gmail.com`

Table A.2: Code metadata

[3] S. Kwon, M. Cha, K. Jung, Rumor detection over varying time windows, PloS one 12 (1) (2017) e0168344.

[4] H. J. Escalante, E. Villatoro-Tello, S. E. Garza, A. P. López-Monroy, M. Montes-y Gómez, L. Villaseñor-Pineda, Early detection of deception and aggressiveness using profile-based representations, Expert Systems with Applications 89 (2017) 99–111.

[5] D. E. Losada, F. Crestani, J. Parapar, erisk 2017: Clef lab on early risk prediction on the internet: Experimental foundations, in: International Conference of the Cross-Language Evaluation Forum for European Languages, Springer, 2017, pp. 346–360.

[6] D. E. Losada, F. Crestani, A test collection for research on depression and language use, in: International Conference of the Cross-Language Evaluation Forum for European Languages, Springer, 2016, pp. 28–39.

[7] B. S. Iskandar, Terrorism detection based on sentiment analysis using machine learning, Journal of Engineering and Applied Sciences 12 (3) (2017) 691–698.

[8] S. G. Burdisso, M. Errecalde, M. M. y Gómez, A text classification framework for simple and effective early depression detection over social media streams, Expert Systems with Applications 133 (2019) 182 – 197. `doi:10.1016/j.eswa.2019.05.023`. URL `http://www.sciencedirect.com/science/article/pii/S0957417419303525`

[9] S. G. Burdisso, M. Errecalde, M. M. y Gómez, UNSL at erisk 2019: a unified approach for anorexia, self-harm and depression detection in social media, in: Experimental IR Meets Multilinguality, Multimodality, and Interaction. 10th International Conference of the CLEF Association, CLEF 2019, Springer International Publishing, Lugano, Switzerland, 2019.

[10] C. Zhang, J. Bi, S. Xu, E. Ramentol, G. Fan, B. Qiao, H. Fujita, Multi-imbalance: An open-source software for multi-class imbalance learning, Knowledge-Based Systems 174 (2019) 137–143.

[11] S. G. Burdisso, M. Errecalde, M. Montes-y Gómez, t-SS3: a text classifier with dynamic n-grams for early risk detection over text streams, ArXiv e-prints`arXiv:1704.06877`.