The Journal of Open Source Software

PySS3: A new interpretable and simple machine learning model for text classification

Sergio G. Burdisso[*1, 2], **Marcelo Errecalde**[1], **and Manuel Montes-y-Gómez**[3]

1 Universidad Nacional de San Luis (UNSL), Argentina **2** Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET), Argentina **3** Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), México

DOI: 10.21105/joss.03934

Software

- Review
- Repository
- Archive

Editor: Brian McFee

Reviewers:

- @hbaniecki
- @kmichael08

Submitted: 08 November 2021
Published: 19 November 2021

Summary

In this paper, we briefly introduce PySS3[1] and share it with the community. PySS3 is an open-source Python package that implements a novel machine learning model for text classification, called SS3. PySS3 comes with two useful tools that allow working with SS3 in an effortless, 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, allowing researchers to better understand the models being deployed. Therefore, PySS3 could be especially useful for those working with sensitive text classification problems by which people's lives could be affected since it allows researchers and practitioners to deploy interpretable (i.e. self-explainable) and more reliable models for text classification.

Statement of need

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), has gained increasing interest in recent years with potential applications in rumor detection (Kwon et al., 2017; Ma et al., 2015, 2016), sexual predator detection, aggressive text identification (Escalante et al., 2017), depression detection (Losada et al., 2017; Losada & Crestani, 2016), and terrorism detection (Iskandar, 2017), among others.

A recently introduced machine learning model for text classification (Sergio G. Burdisso et al., 2019a, 2020), 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 latest CLEF's eRisk lab challenges (Sergio G. Burdisso et al., 2019b, 2019a, 2020; Loyola et al., 2021). Unlike standard classifiers, this new classification model was specially designed to deal with ERD problems since: (a) it is interpretable and therefore can naturally self-explain its rationale, and (b) it naturally supports incremental training and classification over text streams.

However, little attention has been paid to the potential use of SS3 as a general-purpose classifier for other text classification tasks. One of the main reasons could be the fact that there is

*corresponding author

[1] https://github.com/sergioburdisso/pyss3

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 has come to be the most popular programming language in the machine learning community thanks to its simple syntax and a rich ecosystem of efficient open-source implementations of popular algorithms (Oliphant, 2007). Therefore, we decided to develop an open-source Python package to provide the first official implementation of this new classifier.

Implementation

PySS3 was coded to be compatible with Python 2 and Python 3 as well as with different operating systems, such as Linux, macOS, and Microsoft Windows. To ensure this compatibility holds whenever the source code is updated, we have configured and linked the github repository with the Travis CI service. This service automatically runs the test scripts using different operating systems and versions of Python whenever new code is pushed to the repository.

The package is composed of one main module and three submodules.[2] The main module is called `pyss3` and contains the classifier's implementation *per se* in a class called `SS3`. This class not only implements the "plain-vanilla" version of the classifier (Sergio G. Burdisso et al., 2019a) but also different variations, such as the one introduced later by the same authors (Sergio G. Burdisso et al., 2020), which allows SS3 to recognize important word n-grams on the fly. As the reader will notice in the example shown in the next section, this class exposes a clear and simple API that is similar to that of *Scikit-learn* models . For instance, it provides standard methods like `fit()` and `predict()` for training and classification, respectively.[3] Finally, the three submodules, `pyss3.server`, `pyss3.cmd_line`, and `pyss3.util`, provide a collection of useful tools and utility functions such as, for instance, those related to data loading, evaluation or, as will be shown in the next section, "live" testing the models.

Illustrative examples

In this section, we will only introduce two simple illustrative examples. For full and real working examples, please refer to the tutorials in the documentation. Additionally, for readers interested in trying PySS3 out "right away," we have created Jupyter Notebooks for the tutorials,[4] which can be executed online, for instance, using MyBinder.

Before introducing the 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 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")
```

Training and test example

This simple example shows how to train and test an SS3 model using default values. Since SS3 creates a language model for each category, we do not need to create a document-term

[2] A more detailed description of the package is given in the official documentation (https://pyss3.rtfd.io). Additionally, an extended version of this paper can be found in ArXiv(Sergio G. Burdisso et al., 2019).

[3] https://pyss3.rtfd.io/en/latest/api

[4] https://github.com/sergioburdisso/pyss3/tree/master/examples

matrix, we can simply use the raw `x_train` and `x_test` documents for training and test, respectively, as follows:

```
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))
```

Training and live test example

This example is similar to the previous one. However, instead of simply using `predict` and `accuracy` to measure our model's performance, here we are using the PySS3's "Live Test" tool. The "Live Test" is an interactive visualization tool that allows users to test the models "on the fly" with a single line of extra code, as follows:

```
from pyss3 import SS3
from pyss3.server import Live_Test

clf = SS3()
clf.fit(x_train, y_train)

# The following line will open up the Live Test tool, shown in Figure 1
Live_Test.run(clf, x_test, y_test)
```

As shown in Figure 1, the tool provides a user interface by which users can manually and actively test their model using either the documents in the test set or just typing in their own. More precisely, this tool allows researchers to analyze and understand what their models are learning by providing an interactive and visual explanation of the classification process at three different levels (word n-grams, sentences, and paragraphs).[5]

Alternatively to the Live Test tool, we could also use the `clf.extract_insight()` function. When this function is applied to a document, it returns the list of text fragments involved in the classification decision, ordered by a confidence value. For instance, suppose we need to obtain the text fragment with the highest confidence value that was used to classify the document shown in Figure 1 as "sports," we could use `extract_insight()` as follows:

```
# Assign to "doc" the same document that is shown in Figure 1
doc = "Last year, Moore became Liverpool's CEO. This season, his club ..."

fragments = clf.extract_insight(doc, "sports")

print(fragments[0])
```

Which will print the following (fragment, confidence value) pair:

```
('to the Champions League Final. (Liverpool will play Real Madrid in
Kiev, Ukraine on Saturday). Meanwhile Moore has',  0.97)
```

[5] We recommend trying out the "Topic Categorization" or the "Sentiment Analysis on Movie Reviews" online live demo available at http://tworld.io/ss3

This pair indicates the document was classified as "sports," mainly due to our model finding this fragment as belonging to "sports" with a confidence value of 0.97.[6]

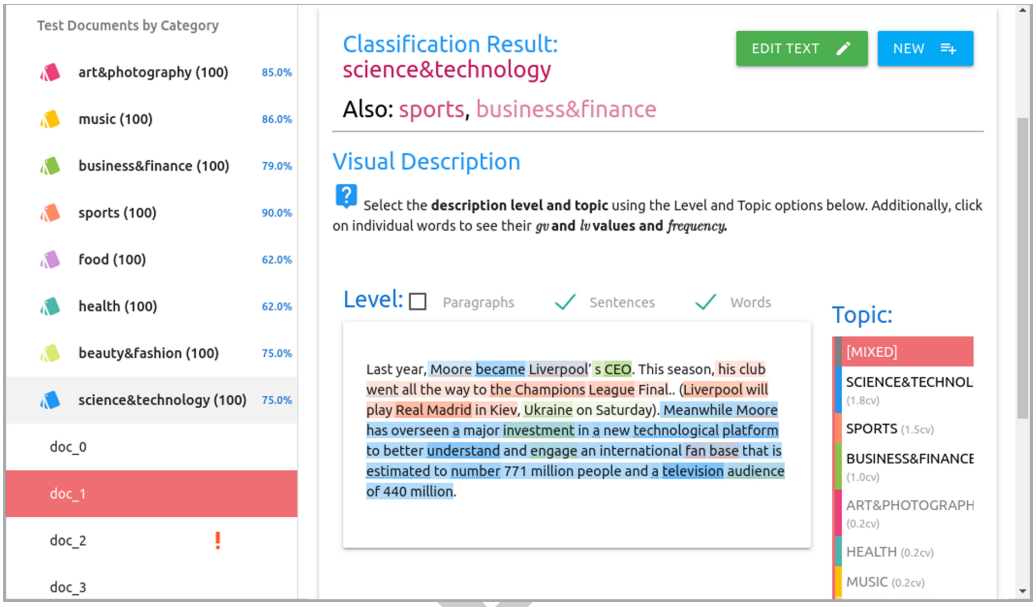

Figure 1: Live Test screenshot. On the left side, the list of test documents grouped by category is shown along with the percentage of success. Note the `doc_2` document is marked with an exclamation mark (!); this mark indicates it was misclassified, which eases error analysis. The user has selected 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. Also, by using the colors, the user could notice that the first sentence belonging to multiple topics, the second sentence shifted the topic to *sports*, and finally, from "Meanwhile" on, the topic is shifted to `technology` (and a little bit of `business` given by the words "investment" or "engage" colored in green). Note that the user can also edit the document or even create new ones using the two buttons on the upper-right corner.

Conclusions

We have briefly introduced `PySS3`, an open-source Python package that implements a novel machine learning model for text classification that comes with useful visualization tools. This software could be especially useful for researchers and practitioners interested in deploying interpretable and more reliable models for text classification. Finally, we hope to continue the advancement and improvement of PySS3 through the direct or indirect help of those in the mathematical and computer science disciplines who want to be part of this open-source project.

References

Burdisso, Sergio G., Errecalde, M., & Montes-y-Gómez, M. (2019a). A text classification framework for simple and effective early depression detection over social media streams. *Expert Systems with Applications*, *133*, 182–197. https://doi.org/10.1016/j.eswa.2019.05.023

[6] Interested readers may refer to the "getting the text fragments involved in the classification decision" tutorial available online

Burdisso, Sergio G., Errecalde, M., & Montes-y-Gómez, M. (2019). PySS3: A python package implementing a novel text classifier with visualization tools for explainable AI. *arXiv Preprint arXiv:1912.09322*.

Burdisso, Sergio G., Errecalde, M., & Montes-y-Gómez, M. (2019b). UNSL at eRisk 2019: A unified approach for anorexia, self-harm and depression detection in social media. *Working Notes of CLEF 2019*.

Burdisso, Sergio G., Errecalde, M., & Montes-y-Gómez, M. (2020). τ-SS3: A text classifier with dynamic n-grams for early risk detection over text streams. *Pattern Recognition Letters*, *138*, 130–137. https://doi.org/10.1016/j.patrec.2020.07.001

Escalante, H. J., Villatoro-Tello, E., Garza, S. E., López-Monroy, A. P., Montes-y-Gómez, M., & Villaseñor-Pineda, L. (2017). Early detection of deception and aggressiveness using profile-based representations. *Expert Systems with Applications*, *89*, 99–111. https://doi.org/10.1016/j.eswa.2017.07.040

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

Kwon, S., Cha, M., & Jung, K. (2017). Rumor detection over varying time windows. *PloS One*, *12*(1), e0168344. https://doi.org/10.1371/journal.pone.0168344

Losada, D. E., & Crestani, F. (2016). A test collection for research on depression and language use. *International Conference of the Cross-Language Evaluation Forum for European Languages*, 28–39. https://doi.org/10.1007/978-3-319-44564-9_3

Losada, D. E., Crestani, F., & Parapar, J. (2017). eRisk 2017: CLEF lab on early risk prediction on the internet: Experimental foundations. *International Conference of the Cross-Language Evaluation Forum for European Languages*, 346–360. https://doi.org/10.1007/978-3-319-65813-1_30

Loyola, J. M., Burdisso, S. G., Thompson, H., Cagnina, L., & Errecalde, M. (2021). UNSL at eRisk 2021: A comparison of three early alert policies for early risk detection. *Working Notes of CLEF 2021*. http://ceur-ws.org/Vol-2936/paper-81.pdf

Ma, J., Gao, W., Mitra, P., Kwon, S., Jansen, B. J., Wong, K.-F., & Cha, M. (2016). Detecting rumors from microblogs with recurrent neural networks. *IJCAI*, 3818–3824.

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

Oliphant, T. E. (2007). Python for scientific computing. *Computing in Science Engineering*, *9*(3), 10–20.