9. FTS log analysis with NLP

Author(s)

Name	Institution	Mail Address	Social Contacts
Federica Legger	INFN Sezione di Torino	federica.legger@to.infn.it	N/A
Micol Olocco	Università di Torino	micol.olocco@edu.unito.it	N/A

How to Obtain Support

Mail	federica.legger@to.infn.it, micol.olocco@edu.unito.it
Social	N/A
Jira	???

General Information

ML/DL Technologies	NLP
Science Fields	High Energy Physics, Computing
Difficulty	Medium
Language	English
Type	fully annotated / runnable / external resource

Software and Tools

Programming Language	Python
ML Toolset	Word2Vec + Rake + sklearn
Additional libraries	panda
Suggested Environments	INFN-Cloud VM, bare Linux Node, Google CoLab

Needed datasets

Data Creator	FTS
Data Type	Error messages
Data Size	8 MB compressed
Data Source	message_example.zip

Short Description of the Use Case

The Worldwide LHC Computing Grid (WLCG) project is a global collaboration of around 170 computing centres in more than 40 countries, linking up national and international grid infrastructures. The mission of the WLCG project is to provide global computing resources to store, distribute and analyse the ~50-70 Petabytes of data expected every year of operations from the Large Hadron Collider (LHC) at CERN. The CERN File Transfer System (FTS) is one of the most critical services for WLCG. FTS is a low level protocol used to transfer data among different sites. FTS sustains a data transfer rate of 20-40 GB/s, and it transfers daily a few millions files.

If a transfer fails, an error message is generated and stored. Failed transfers are of the order of a few hundred thousand per day. Understanding and possibly fixing the cause of failed transfers is part of the duties of the experiment operation teams. Due to the large number of failed transfers, not all can be addressed. We developed a pipeline to discover failure patterns from the analysis of FTS error logs. Error messages are read in, cleaned from meaningless parts (file paths, host names), and the text is analysed using NLP (Natural Language Processing) techniques such as word2vec. FInally the messages can be grouped in clusters based on the similarity of their text using ML algorithms for unsupervised clustering such as DBSCAN. The biggest clusters and their relationship with the host names with largest numbers of failing transfers is presented in a dedicated dashboard for the CMS experiment (access to the dashboard requires login with CERN SSO). The clusters can be used by the operation teams to quickly identify anomalies in user activities, tackle site issues related to the backlog of data transfers, and in the future to implement automatic recovery procedures for the most common error types.

How to execute it

Way #1: Use Jupyter notebooks from an INFN Cloud Virtual machine

Once you have a deployment with JupyterHub on INFN Cloud, you can open a terminal and clone this github repo. Then execute the notebook in

TestsINFNCloud/test_clusterLogs/NLP_example.ipynb

The input file can be found in zipped form in the github repo (message_example.zip), or read in from Minio.

Way #2: Execute from any machine with JupyterHub

Open a terminal and clone this github repo. Then execute the notebook in

TestsINFNCloud/test_clusterLogs/NLP_example.ipynb

Annotated Description

The error message analysis, and in general any text analysis, can be divided in two main phases: **pre-processing**, made of *Data preparation* and *Tokeniza tion*, and **processing**, specific of the analysis. Before going deeper into the description of each phase, we have to define what we mean for **similarity**. Indeed this is a key concept for our purpose, being the metric that will regulate the clustering. In our approach we map the words in numeric vectors so that similarity of *x* and *y* can be expressed, for instance, by the cosine of the angle between the corresponding vectors.

Pre-processing Phase

Let us begin by showing two error messages as example:

```
TRANSFER ERROR: Copy failed with mode 3rd push, with error: Transfer failed: failure: problem sending data:
java.security.cert.CertificateException: The peers certificate with subjects DN CN=pps05.lcg.triumf.ca, OU=triumf.
ca, O=Grid, C=CA was rejected.
The peers certificate status is: FAILED The following validation errors were found:;error at position 0 in chain,
problematic certificate
subject: CN=pps05.lcg.triumf.ca, OU=triumf.ca, O=Grid,C=CA (category: CRL): Signature of a CRL corresponding to
this certificates CA is invalid
```

```
TRANSFER ERROR: Copy failed with mode 3rd pull, with error: copy 0) Could not get the delegation id: Could
not get proxy request: Error 404 fault: SOAP-ENV:Server [no subcode] HTTP/1.1 404 Not Found Detail:
<!DOCTYPE HTML PUBLIC -//IETF//DTD HTML 2.0//EN> <html><head> <title>404 Not Found</title> </head><body>
<h1>Not Found</h1> <p>The requested URL /gridsite-delegation was not found on this server.</p>
</body></html> \n.
```

As we can see, the error message contains "particular information": words between tags, URLs, sites, usernames to mention a few. Therefore the first step, *Data preparation*, aims to reduce the variety of error messages by applying cleaning. The cleaning is performed by using regular expressions, namely REGEX.

After cleaning from a selected set of punctuation, messages are broken into their constituents (i.e words). This is the *Tokenization* phase.

After Data Preparation:

```
TRANSFER ERROR Copy failed with mode pull with error copy Could not get the delegation id Could not get proxy
request Error fault SOAP-ENVServer no subcode HTTP Not Found Detail .
```

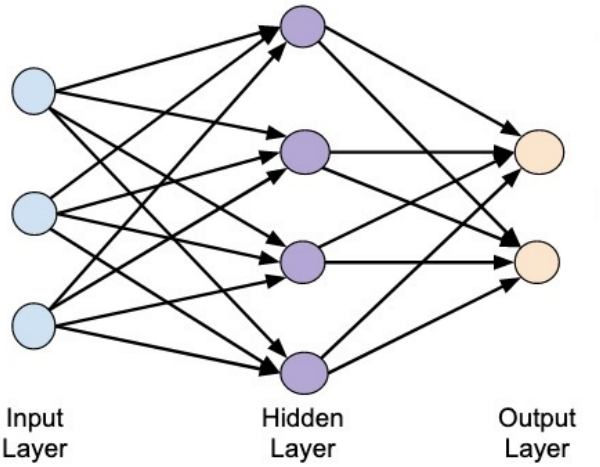
After Tokenization:

```
TRANSFER, ERROR, Copy, failed, with, mode, pull, with, error, copy, Could, not, get, the, delegation, id, Could,
not, get, proxy, request, Error, fault, SOAP-ENVServer, no, subcode, HTTP, Not, Found, Detail, .
```

Processing Phase

Word2Vec is a Machine Learning algorithm used in Natural Language Processing applications to compute vector representations of words (*word embedding*). It was developed in 2013 by a Google team made of Tomas Mikolov, Kai Chen, Greg Corrado and Jeffrey Dean. Why do we need Word Embeddings? Let us consider the words *file, directory, cat*: how can an algorithm know that *file* can be related to *directory* while it would hardly appear with *cat*? Machine Learning algorithms are not good at dealing with strings but they can achieve astonishing results with numbers: this is the reason for word embedding. Mikolov's team found that "similarity of words representations goes beyond simple syntactic regularities". They showed, for example, thatthe vectorial sum of **King - Man + Woman** results in a vector that is closest to the vector representation of the word Queen. The success of this algorithm resides not only in the accuracy it can achieve, but also in the time required; indeed it is able to learn high quality word vectors from billions of words in less than a day.

Word2Vec is a shallow network, that is a network made of an input layer, an output layer and one hidden layer:



It is an example of *Prediction based Embeddings*, because it encodes the probability of a word given its context (or vice-versa). Having a high-level insight about the underlying process, we can say that the algorithm trains a simple neural network (NN) on a certain objective (*fake task*), but we are not going to use that NN for the task we trained it on. Instead, the goal is actually just to learn the weights of the hidden layer: these weights are actually the "word vectors" we are looking for. Having a resume:

- *fake task*: learning the probability for a certain sequence of tokens;
- *goal*: obtaining the parameters that minimize the error in the fake task.

Word2Vec can work with two different architectures:

- *Continuous Bag-of-Words* (CBOW): predicts the center word based on surrounding context words, where the context is *m* (given as hyperparameter) words before and after a center word;
- *Continuous Skip-gram* predicts the context given a word, where, again, the context is *m* words before and after the input word.

Both architectures are "continuous" meaning they use a continuous distributed representations of the context.

References

Heikki Hyyro. “Explaining and Extending the Bit-parallel Approximate StringMatching Algorithm of Myers”. In: (Oct. 2002).

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient Estimation of Word Representations in Vector Space." 2013. arXiv:1301.3781

Attachments

message_example.zip