ANOMALY DETECTION OF SYSTEM LOGS BASED ON NATURAL LANGUAGE PROCESSING AND DEEP LEARNING

DAVID DRAMBIAN

ALESSANDRO AGOSTEO

AIM OF THE PROJECT

The aim of this project is to analyze system logs, which nowadays are complex and rich in semantic information, in search for anomaly detections which then will be classified.

Two feature extraction algorithms are used, Word2vec and TF-IDF, in order to obtain embeddings of the logs for anomaly detection.

Afterwards, those vector representations are used to train machine learning models, like Logistic Regression, Catboost Classifier, Naive Bayes and LSTM. The outcomes are validated using classification metrics: accuracy, precision, recall, F1-score, ROC-AUC.

In conclusion, it is proven that the LSTM has the best performance overall for anomaly detection, proving that it can capture contextual semantic information effectively.

PAPER CONTENTS

In the paper the collected logs are nearly 30 GBs worth of data obtained from the Thunderbird supercomputer.

After preprocessing the logs, removing unnecessary information, every log that contains one or more of the words among «ALERT», «FATAL» and «FAILED» is labeled with a 1, and 0 the rest, the Word2vec and TF-IDF algorithms are then used to extract features, that are subsequently processed by LSTM algorithm.

In conclusion the results are compared to the outcomes obtained by the Naive Bayes and GBDT algorithms, proving that the LSTM to be slightly better.

It can be observed that every algorithm performed better with the features that result from the Word2vec algorithm.

PAPER RESULTS

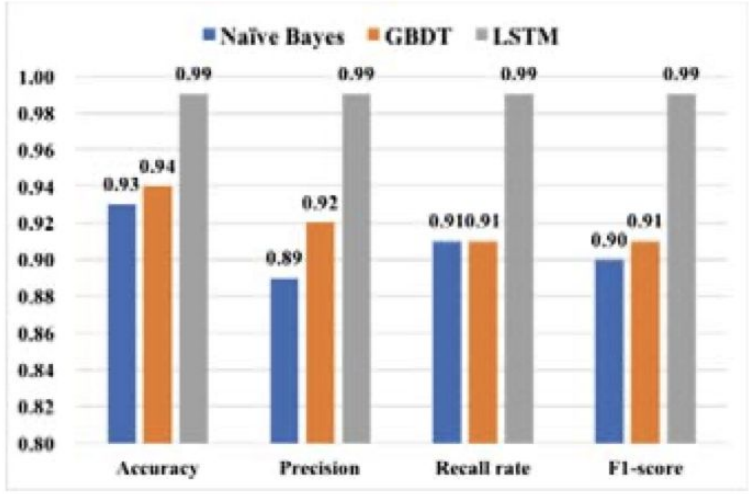


Figure 5. Results based on feature extraction of TF-IDF.

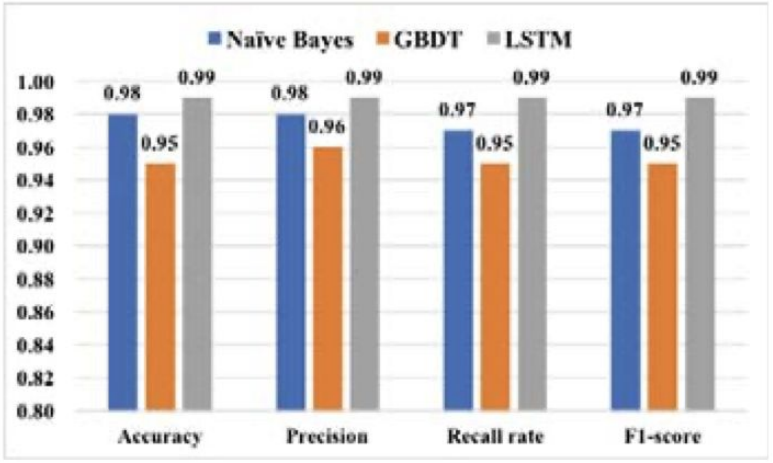


Figure 6. Results based on feature extraction of Word2vec.

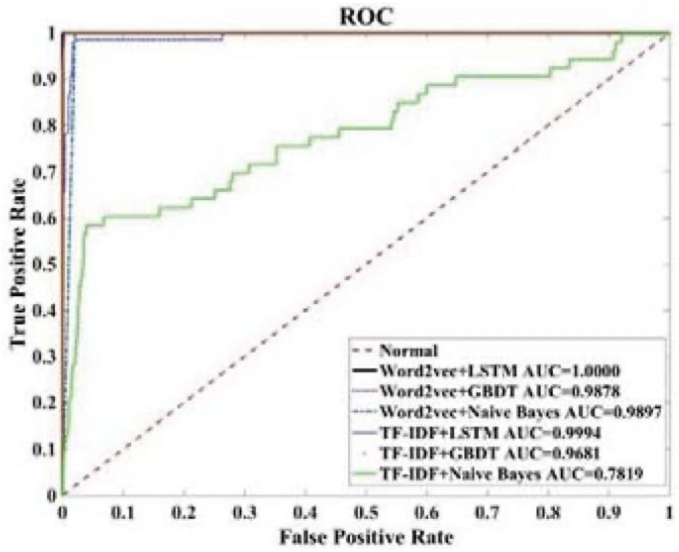


Figure 7. The ROC curve of different methods on Thunderbird dataset.

Feature extraction	Learning algorithms	Accuracy	Precision	Recall rate	F1-score
TF-IDF	Naïve Bayes	0.93	0.89	0.91	0.90
	GBDT	0.94	0.92	0.91	0.91
	LSTM	**0.99**	**0.99**	**0.99**	**0.99**
Word2vec	Naïve Bayes	0.98	0.98	0.97	0.97
	GBDT	0.95	0.96	0.95	0.95
	LSTM	**0.99**	**0.99**	**0.99**	**0.99**

PROJECT CONTENTS

Contrary to the paper, the analysis of the project is done on 1 GB worth of data, for computing performance reasons.

First of all the dataset is established, taking mainly into consideration system logs and their categories, based on the presence of certain keywords, same way as in the paper.

The data is tokenized and lemmatized, then after applying the TF-IDF and Word2vec algorithms, we test the performances of the Logistic Regression, Naive Bayes and Catboost compared to LSTM algorithm.

REQUIRED PACKAGES, FUNCTION USED IN MODEL EVALUATION AND COLLECTION OF LOGS

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import accuracy_score
from imblearn.over_sampling import RandomOverSampler
import math

from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.corpus import stopwords
from pymystem3 import Mystem
from string import punctuation
from sklearn.feature_extraction.text import CountVectorizer
from nltk.stem import WordNetLemmatizer
from scikitplot.metrics import plot_roc # plot_roc_curve


def round_up(n, decimals=2):
    multiplier = 10 ** decimals
    return math.floor(n * multiplier) / multiplier
```

```
with open('tbird2_1G', 'r', newline='\n') as f2:
    file_content = f2.readlines()[:-1]
```

DATASET CREATION

```
df = pd.DataFrame(file_content, columns=['raw'])
df['hyp'] = df.raw.apply(lambda x: x.split()[0])
df['id'] = df.raw.apply(lambda x: x.split()[1])
df['log'] = df.raw.apply(lambda x: ' '.join(x.split()[2:]))

### target definition
df['alert_flg'] = df.log.apply(lambda x: 1 if 'alert' in x.lower() else 0)
df['fatal_flg'] = df.log.apply(lambda x: 1 if 'fatal' in x.lower() else 0)
df['failed_flg'] = df.log.apply(lambda x: 1 if 'failed' in x.lower() else 0)
df['target'] = np.where(df['alert_flg']+df['fatal_flg']+df['failed_flg']>0, 1, 0)

df.drop(columns=['raw'], inplace=True)
```

```
df = df.drop_duplicates(subset=['log'])
```

EXAMPLE OF FILE CONTENTS

```
print(file_content[-1])
print(file_content[-100])
print(file_content[1])
print(file_content[12345])
```

```
- 1132502080 2005.11.20 an230 Nov 20 07:54:40 an230/an230 snmpd[1976]: Got trap from peer on fd 13

- 1132502061 2005.11.20 tbird-sm1 Nov 20 07:54:21 src@tbird-sm1 ib_sm.x[24904]: [ib_sm_discovery.c:470]: Failed to GetNodeInfo
() because of NO RESPONSE

- 1131524071 2005.11.09 tbird-admin1 Nov 10 00:14:31 local@tbird-admin1 postfix/postdrop[10896]: warning: unable to look up pub
lic/pickup: No such file or directory

- 1131569328 2005.11.09 tbird-admin1 Nov 9 12:48:48 local@tbird-admin1 /apps/x86_64/system/ganglia-3.0.1/sbin/gmetad[1682]: dat
a_thread() got not answer from any [Thunderbird_D7] datasource
```

SPLIT OF DATASET INTO TRAIN AND TEST SETS, WITH OVERSAMPLING

```
# IV. B. 'training set and test set with a ratio of 8:2'

X_train, X_test, y_train, y_test = train_test_split(df[['log']], df.target, test_size=0.2)
```

```
### OVERSAMPLING
ros = RandomOverSampler(random_state=42)
X_train, y_train = ros.fit_resample(X_train, y_train)
X_train, X_test = X_train.log, X_test.log

df.target.value_counts(), y_train.value_counts(), y_test.value_counts()
```

```
(0    4280569
 1     960171
 Name: target, dtype: int64,
 0    3425010
 1    3425010
 Name: target, dtype: int64,
 0    855559
 1    192589
 Name: target, dtype: int64)
```

PREPROCESSING

```
# tokenize
X_train_tok = X_train.apply(CountVectorizer().build_tokenizer())
X_test_tok = X_test.apply(CountVectorizer().build_tokenizer())

# create lemmatizer and stopwords list
lemmatizer = WordNetLemmatizer()
stopwords_list = stopwords.words('english')

# preprocess function
def preprocess_text(text):
    tokens = [lemmatizer.lemmatize(w.lower()) for w in text
      if w.lower() not in stopwords_list and w not in punctuation]
    return ' '.join(tokens)

X_train_preproc = X_train_tok.apply(preprocess_text)
print('DONE')
X_test_preproc = X_test_tok.apply(preprocess_text)
print('DONE')
```

TF-IDF VECTORIZATION

```
### no preprocessing option
X_train_preproc = X_train.copy()
X_test_preproc = X_test.copy()

### TF-IDF vectorization
tfidf = TfidfVectorizer(min_df=5, stop_words='english')
X_train_tfidf = tfidf.fit_transform(X_train_preproc)
X_test_tfidf = tfidf.transform(X_test_preproc)
```

```
X_train_tfidf.shape
```

```
(6850020, 68740)
```

MACHINE LEARNING ALGORITHMS

```
models_d = {}
```

```
from sklearn.linear_model import LogisticRegression

models_d['LogisticRegression'] = LogisticRegression(max_iter=100)
models_d['LogisticRegression'].fit(X_train_tfidf, y_train)
```

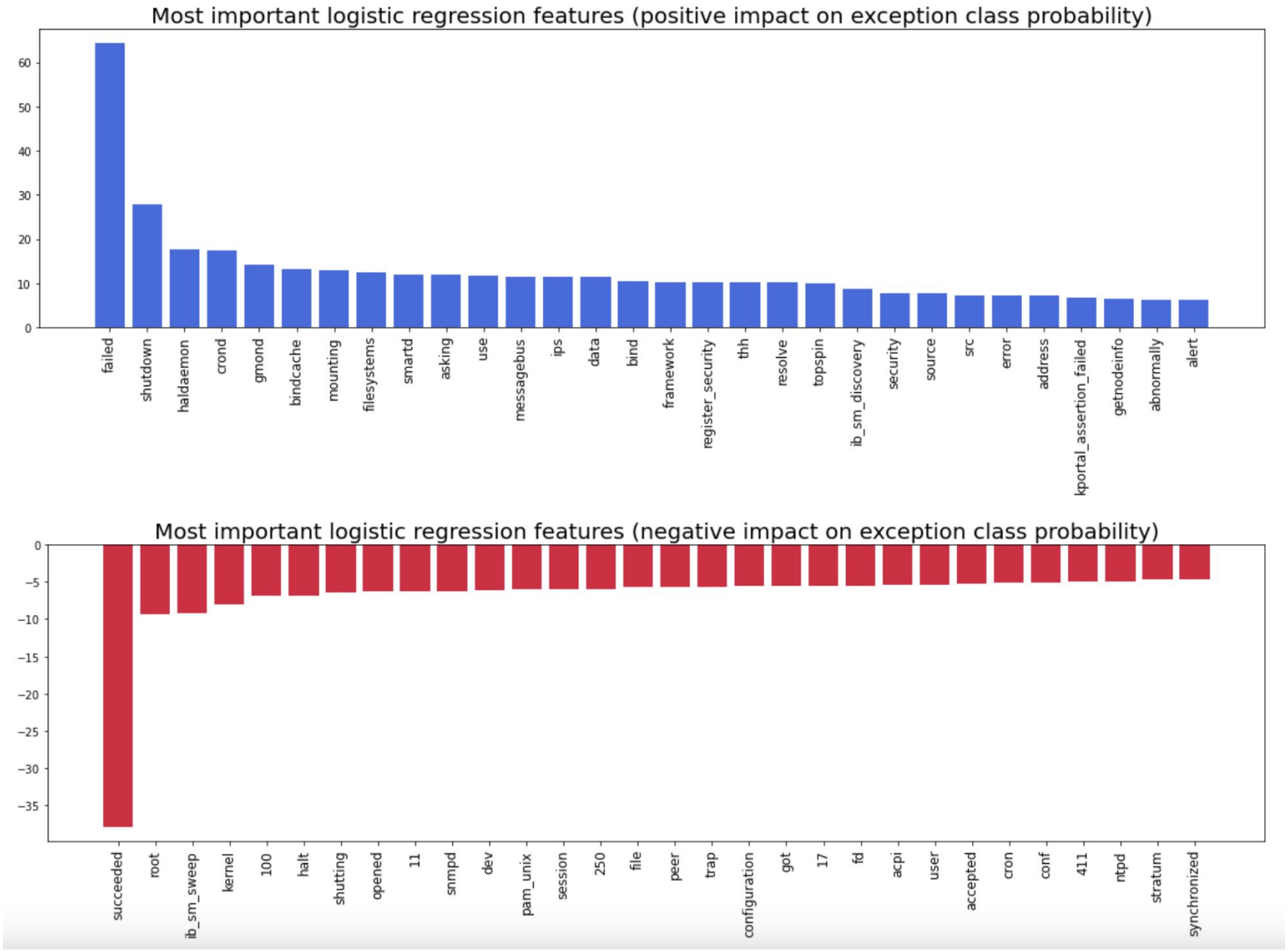
```
from catboost import CatBoostClassifier
from sklearn.linear_model import LogisticRegression

models_d['CatBoostClassifier'] = CatBoostClassifier(iterations=30, depth=5, eval_metric='Accuracy')
models_d['CatBoostClassifier'].fit(X_train_tfidf, y_train)
```

```
from sklearn.naive_bayes import MultinomialNB

models_d['NaïveBayes'] = MultinomialNB(alpha=0.01)
models_d['NaïveBayes'].fit(X_train_tfidf, y_train)
```

Insights


Most important logistic regression features (positive impact on exception class probability)
60
50
40
30
20
10
0
failed
shutdown
haldaemon
crond
gmond
bindcache
mounting
filesystems
smartd
asking
use
messagebus
ips
data
bind
framework
register_security
thh
resolve
topspin
ib_sm_discovery
security
source
src
error
address
kportal_assertion_failed
getnodeinfo
abnormally
alert
Most important logistic regression features (negative impact on exception class probability)
0
−5
−10
−15
−20
−25
−30
−35
succeeded
root
ib_sm_sweep
kernel
100
halt
shutting
opened
11
snmpd
dev
pam_unix
session
250
file
peer
trap
configuration
got
17
fd
acpi
user
accepted
cron
conf
411
ntpd
stratum
synchronized

TOKENIZATION AND PADDING IN PREPARATION OF LSTM ALGORITHM

```
### LSTM
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Embedding, Dropout, GlobalAveragePooling1D, SpatialDropout1D

### preprocessing
max_len = 32
vocab_size = 10000

tokenizer = Tokenizer(num_words=vocab_size, char_level=False, oov_token='<OOV>')
tokenizer.fit_on_texts(X_train)
print('DONE')

training_sequences = tokenizer.texts_to_sequences(X_train)
training_padded = pad_sequences(training_sequences, maxlen=max_len, padding='post', truncating='post')
print('DONE')

testing_sequences = tokenizer.texts_to_sequences(X_test)
testing_padded = pad_sequences(testing_sequences, maxlen=max_len, padding='post', truncating='post')
print('DONE')
```

```
DONE
DONE
DONE
```

LSTM MODEL ARCHITECTURE

```
### model
embedding_dim = 16

# Define Dense Model Architecture
modelLSTM = Sequential()
modelLSTM.add(Embedding(vocab_size, embedding_dim, input_length=max_len))
modelLSTM.add(SpatialDropout1D(0.5)) # as in paper
modelLSTM.add(LSTM(32, return_sequences=False)) # as in paper
modelLSTM.add(Dropout(0.5)) # as in paper
modelLSTM.add(Dense(1, activation='sigmoid'))


modelLSTM.compile(loss='binary_crossentropy', optimizer = 'adam' , metrics = ['accuracy'])
modelLSTM.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding (Embedding)        (None, 32, 16)            160000

 spatial_dropout1d (SpatialD  (None, 32, 16)           0
 ropout1D)

 lstm (LSTM)                  (None, 32)                6272

 dropout (Dropout)            (None, 32)                0

 dense (Dense)                (None, 1)                 33

=================================================================
Total params: 166,305
Trainable params: 166,305
Non-trainable params: 0
_________________________________________________________________
```

MODEL FITTING AND EVALUATION

```
early_stop = EarlyStopping(monitor='val_loss', min_delta=0.01)
history = modelLSTM.fit(training_padded, y_train, epochs=1, batch_size=256,
                    validation_data=(testing_padded, y_test), callbacks =[early_stop]) # , verbose=2)
```

```
26758/26758 [==============================] - 458s 17ms/step - loss: 0.0019 - accuracy: 0.9995 - val_loss: 1.2019e-04 - val_ac
curacy: 1.0000
```

```
# model.evaluate(testing_padded, y_test)
lstm_pred_proba = modelLSTM.predict(testing_padded)
lstm_pred = np.where(lstm_pred_proba.reshape(-1)>0.5, 1, 0)
lstm_train_pred_proba = modelLSTM.predict(training_padded)
lstm_train_pred = np.where(lstm_train_pred_proba.reshape(-1)>0.5, 1, 0)
accuracy_score(y_test, lstm_pred), [x[0] for x in precision_recall_fscore_support(y_test, lstm_pred)]
```

```
(0.9999570671317409,
 [0.9999579235751871, 0.9999894805618315, 0.9999737018195418, 855559])
```

LSTM WITH WORD2VEC: EMBEDDING MATRIX

```
### Word2Vec + LSTM
from gensim.models import Word2Vec

w2v = Word2Vec(sentences=X_train_preproc.apply(lambda x: x.split(' ')),
               vector_size=embedding_dim, window=5, min_count=10, max_vocab_size=vocab_size)

embedding_matrix = np.zeros((vocab_size, embedding_dim))
for word, i in w2v.wv.key_to_index.items():
    embedding_vector = w2v.wv[word]
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

LSTM WITH WORD2VEC: ARCHITECTURE

```
# Define Dense Model Architecture
modelLSTM = Sequential()
modelLSTM.add(Embedding(vocab_size, embedding_dim, weights=[embedding_matrix],
                        input_length=max_len, name="w2v_embedding"))
modelLSTM.add(SpatialDropout1D(0.5))
modelLSTM.add(LSTM(32, return_sequences=False))
modelLSTM.add(Dropout(0.5))
modelLSTM.add(Dense(1, activation='sigmoid'))


modelLSTM.compile(loss='binary_crossentropy', optimizer = 'adam' , metrics = ['accuracy'])
modelLSTM.summary()
```

```
Model: "sequential_12"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 w2v_embedding (Embedding)   (None, 32, 16)            160000

 spatial_dropout1d_7 (Spatia  (None, 32, 16)           0
 lDropout1D)

 lstm_7 (LSTM)               (None, 32)                6272

 dropout_12 (Dropout)        (None, 32)                0

 dense_17 (Dense)            (None, 1)                 33

=================================================================
Total params: 166,305
Trainable params: 166,305
Non-trainable params: 0
_________________________________________________________________
```

LSTM WITH WORD2VEC: FITTING AND EVALUATION

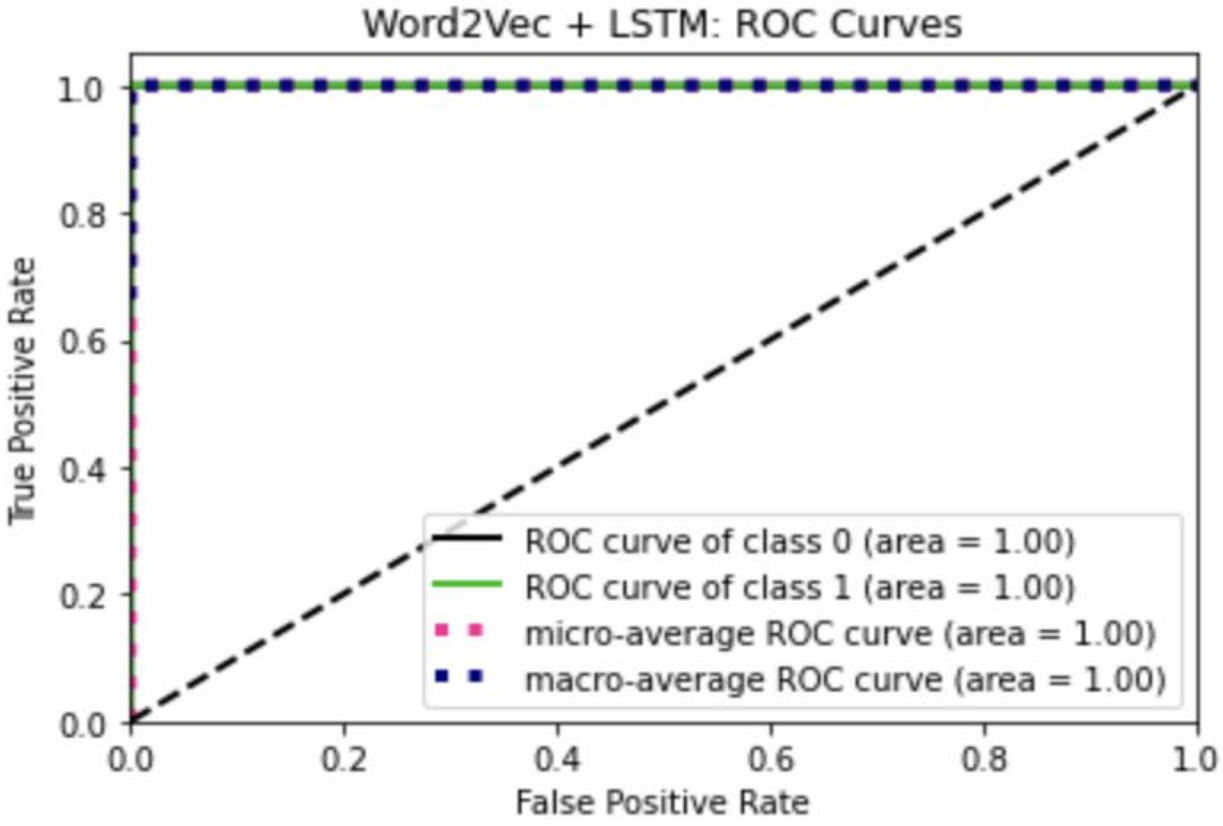
```
early_stop = EarlyStopping(monitor='val_loss', min_delta=0.01)
history = modelLSTM.fit(training_padded, y_train, epochs=1, batch_size=256,
                    validation_data=(testing_padded, y_test), callbacks =[early_stop])
```

```
26758/26758 [==============================] - 470s 17ms/step - loss: 0.0167 - accuracy: 0.9948 - val_loss: 6.6581e-04 - val_ac
curacy: 0.9999
```

```
lstm_pred_proba = modelLSTM.predict(testing_padded)
lstm_pred = np.where(lstm_pred_proba.reshape(-1)>0.5, 1, 0)
lstm_train_pred_proba = modelLSTM.predict(training_padded)
lstm_train_pred = np.where(lstm_train_pred_proba.reshape(-1)>0.5, 1, 0)
accuracy_score(y_test, lstm_pred), [x[0] for x in precision_recall_fscore_support(y_test, lstm_pred)]
```

```
(0.9999208127096555,
 [0.9999357165732025, 0.9999672728590313, 0.999951494467155, 855559])
```

LSTM RESULTS



COMPARING WITH THE PAPER

Differently from the paper, we managed to achieve better results, both from the machine learning models and from the LSTM algorithm, this could be for different reasons like a better choice for the hyperparameters, better preprocessing of the logs or the large difference between the used dataset in the paper and the one we analyzed.

Feature extraction	Learning algorithms	Accuracy	Precision	Recall rate	F1-score
TF-IDF	Naïve Bayes	0.93	0.89	0.91	0.90
	GBDT	0.94	0.92	0.91	0.91
	LSTM	**0.99**	**0.99**	**0.99**	**0.99**
Word2vec	Naïve Bayes	0.98	0.98	0.97	0.97
	GBDT	0.95	0.96	0.95	0.95
	LSTM	**0.99**	**0.99**	**0.99**	**0.99**

model	train accuracy	train precision	train recall	train F1	test accuracy	test precision	test recall	test F1
LogisticRegression	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
CatBoostClassifier	0.99	0.99	1.00	0.99	0.99	0.99	1.00	0.99
NaïveBayes	0.98	0.97	0.99	0.98	0.99	0.99	0.99	0.99
LSTM	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99

CONCLUSIONS

This project proves that the LSTM and other state-of-the art algorithms combined with the TF-IDF or Word2vec feature extraction methods, especially the latter, is greatly efficient at capturing anomaly detection. For simple tasks, even less complex models, like Naive Bayes Classifier and GBDT can perform relatively well.

The obtained results are largely in coherence with the ones obtained on the paper due to the same methods and approaches used for anomaly detection of system logs based on Natural Language Process techniques and Deep Learning models.