# Anomaly Detection
## Data 586 Project Report

**Nelson Tang**
tangaot@mail.ubc.ca

**LingXiang Zou**
czou0101@student.ubc.ca

## 1 Abstract

Mordern computer systems generate vast amounts of data in the form of log files during execution. The information in the log file helps people understand how their system works. In particular, error logs are useful for analyzing the causes of system failures. Machine learning approaches are very helpful in recognizing patterns for identifying behavior. We evaluate the effect of the Logistic Regression, Naive Bayes and Random Forest in Bag of Words (BOW). We also implemented Convolutional Neural Network (CNN), Recurrent Neural Network (RNN) in Word Embedding on identifying anomaly logs.

## 2 Introduction

Log files are important for identifying critical or anomalous events. Almost all software contains log files that store history of actions by the system, such as anomalies, errors, or suspicions. To mitigate risks, organizations need to analyze log files to identify problems before or when they occur, to reducing costs, wasted time, and unnecessary delays. In this project, the logs are collected from a BlueGene/L supercomputer system with labelled alert and non-alert messages. There are 4.7 million log messages in the dataset with 92.7% of them are normal and 7.3% are anomaly. Our goal is to detect anomalous logs that minimize human intervention while improving the accuracy of this automated approach.

This project aims to implement different deep learning approaches. We implemented two methods to study the effect of different models. To compare our approaches, we convert the log messages to Bag of Words (BOW) to build Logistic Regression, Naive Bayes, and Random Forest classification models. On the other hand, we also convert log messages to word vector using the word embedding model (Word2vec, GloVe) to build Convolutional Neural Nets (CNN) and Long Short-Term Memory (LSTM) based RNN classification models. We use accuracy, precision, recall, and F1 scores as our metrics to measuring the performance of each model.

## 3 Background/Related Work

Over the last few years, anomaly detection has been intensively studied, especially in the context of log files data. Researchers have used various machine learning systems to solve the log monitoring problems. Enterprise applications typically generate a large number of logs to record run-time state and important events. Log anomaly detection is very effective for business management and system maintenance. Many companies have developed their Anomaly Detector for sever, that is a machine learning service that uses time series log message data to automatically detect anomalies in application. Existing researches on neural network approaches for log anomaly classification and sentiment classification are reviewed in the following sections.

### 3.1 Content-Based Anomaly Detection for System Logs

Dan Lv, Nurbol Luktarhan, and Yiyong Chen (Dan Lv, 2021) developed Multi-layer Long Short-Term Memory (LSTM) models on a HDFS log dataset. Through vectoring the words in the log content, then removing the invalid words by lexical annotation, and finally obtaining the sequence vector by weighted average method, to capture the semantic information in the log, and the log order relationship. The results show that the F1 of LSTM achieves the best prediction of 0.98. Their finding indicate the advantages of target-dependent LSTM model on the log anomaly classification.

### 3.2 Convolutional Neural Networks for Sentence Classification

Yoon Kim (Kim, 2014) has proposed a series of experiments with convolutional neural networks (CNNs) that have been trained on top of pre-trained word vectors for sentence-level classification tasks. It shows a simple CNN with a limited hyper-parameter tuning and a static vector achieves excellent results on multiple benchmarks. Moreover, a simple modification of the architecture was documented to allow for the both use of task-specific and static vectors. The results showed that, including sentiment analysis and query classification, the CNN models addressed herein enhance the state of the art on 4 out of 7 tasks.

### 3.3 Research on Anomaly Detection and Real-time Reliability Evaluation with The Log of Cloud Platform

A combination of machine learning algorithms was suggested by Bo and Qingyi (Wang et al., 2022). In their experiment, they examined three common machine learning algorithms including Logistic Regression, Support Vector Machine (SVM), and PCA-Q (Principal Component Analysis-Q Statistics) to massive system log analysis based on a systematic analysis of the whole log processing process including log analysis, feature extraction, anomaly detection, prediction evaluation and real-time reliability. The machine learning prediction models are found to perform well in terms of anomaly prediction accuracy, recall and F1 values for large-scale datasets.

### 3.4 Real-time Event Detection Using Recurrent Neural Network in Social Sensors

Van and Tien's (Nguyen et al., 2019) propose a technique for detecting temporal events on textual content data from social media using deep learning and multi-word embedding. First, a convolutional neural network and more than one word embedding architecture are used as text content classifiers to pre-process the input text data. Second, an event detection model using a recurrent neural network is used to learn features of temporally collected data by extracting temporal information. For event detection, a long short-term memory(LSTM) network is used as a predictor to learn a better degree of temporal features with the aim of predicting future values. An error distribution estimation model is constructed to calculate the observed anomaly rating. Event detection was performed for anomaly ratings using a window-based technique.

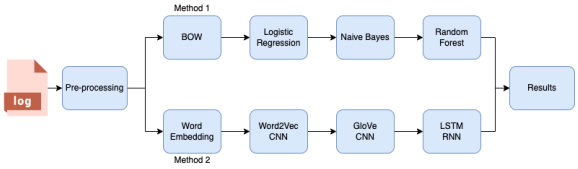


Figure 1: Workflow

### 3.5 Comparison of Word Embedding for Classification Tasks

Hongmin and Xukun (Li et al., 2018) proposed a classifiers for filtering crisis events by using a simple feature based adaptation method where messages are represented with word embedding or sentence encoding as dense numerical vectors of reduced dimensional. They compared message representation of different word embedding and sentence encoding in order to understand what embedding are more suitable for use in classification task. It appears in their results that the GloVe embedding-based messages representations yield better results than the representations that use other embedding. In addition, the embedding of GloVe trained on data yields better results while the embedding of GloVe pre-trained on a wide selection of general data.

## 4 Approach

This project concentrates on various strategies - BOW and Word Embedding - to study the effects of various deep learning approaches. Figure 1 shows the workflow and the tasks performed are summarized below:

1. Data pre-processing and split the data to Train and Test
2. Method 1:
   (a) Convert log files to BOW on train data
   (b) Build Classification Models
      i. Logistic Regression
      ii. Naive Bayes
      iii. Random Forest
   (c) Assess the models' performance on the test dataset

3. Method 2:

   (a) Train a word embedding model on a pre-trained model

   (b) Using the word embedding model to transform text to word vector

   (c) Build Classification Models

      i. CNN - Word2Vec

      ii. CNN - GolVe

      iii. Long Short Term Memory Networks

   (d) Assess the models' performance on the test dataset

In the following, we describe these methods in detail.

### 4.1 Bag of Words

The bag-of-words (BOW) model is a simplifying representation used in natural language processing and information retrieval. It will generate the occurrence of each word from the text data which is used as a feature for training a model. This is especially helpful because the model would see far more occurrences of a specific word and the corresponding target value than the sequence of terms.

#### 4.1.1 Logistic Regression

Logistics regression is common and is a useful regression method for solving the binary classification problem. Also, Logistic Regression is a Machine Learning algorithm which is used for the classification problems(Pant, 2019).Fit the data to a linear regression model, and then operate on it by predicting the logistic function of the target categorical dependent variable. In order to predict which category the data belongs to, a threshold can be set. Based on the threshold, the obtained estimated probabilities are classified.

#### 4.1.2 Naive Bayes

The Naive Bayes classifier is one of the simple and most effective Classification algorithms and is a simple classifier that classifies based on probabilities of events. It is the applied commonly to text classification. Though it is a simple algorithm, it performs well in many text classification problems which helps in building the fast machine learning models that can make quick predictions.

#### 4.1.3 Random Forest

A random forest is a classification algorithm consisting of many decision trees. It uses bagging and feature randomness in the construction of each individual tree in an attempt to create a forest of unrelated trees whose committee predictions are more accurate than those of any individual tree. Random forest classifiers are suitable for processing high-dimensional noisy data in text classification(Islam et al., 2019). A random forest model consists of a set of decision trees, each of which is trained with a random subset of features.

### 4.2 Word Embedding

Word embeddings are a type of word representation that allows words with similar meaning to have a similar representation(Brownlee, 2017). An embedding is a relatively low-dimensional space into which you can translate high dimensional vectors. Embedding make it easier to do machine learning on large inputs like sparse vectors representing words. Ideally, an embedding captures some of the semantics of the input by placing semantically similar inputs close together in the embedding space. We implement a function that transforms the text to word vectors. Basically, we put the word index into the pre-trained embedding layer before the neural network. The embedding vector is a pre-trained word vector space from a large corpus, and the output of the embedding layer is just a list of word coordinates in the vector space. We can also use the distance of these coordinates to detect relevance and context. Words with a closer meaning will be located closer in the word space.

#### 4.2.1 Word2vec CNN

In a large supervised training set, initializing with word vectors obtained from an supervised neural language model is a popular method to improve performance. We use publicly available word2vec vectors trained on 100 billion words in Google News. These vectors have a dimension of 300 and are trained using a continuous bag-of-words architecture, which randomly trains words that do not appear in the pre-trained word set. Once the Word2vec representation of the label is obtained, we can train a neural network that can predict the category of a given input label. They only accept fixed-size digital inputs. But we have some labels that are strings. And we got the word embedding from the Word2vec model. The Keras library has all the necessary conditions to bring all these to-

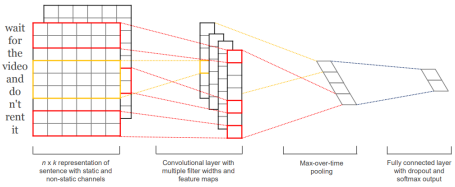


Figure 2: CNN model (Kim, 2014)

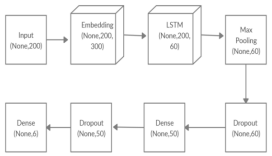


Figure 3: LSTM Architecture

gether. It provides an embedding layer that can be used to train neural networks on text data. You can initialize it with random weights, or you can use word embeddings learned elsewhere. We use Tokenizer to encode tags, and use Embedding layer to tell NN the word embedding in the Word2vec model. In addition, we allow the NN to modify the embedding during training.

#### 4.2.2 GloVe CNN

We use word embeddings from pre-trained Glove that was trained on a dataset of one billion tokens (words) with a vocabulary of 400 thousand words. First, we download the glove embedding. These files contain the mapping of each word to a 300-dimensional vector, also known as embedding. These embeddings are based on the probability of simultaneous occurrence of words. Then, we convert each text file into fixed length sequence of words by padding. We create embedding layer that contains mapping for words into glove vectors. we extract word embeddings to create an embedding matrix, and the embedded matrix contains valuable information for the CNN's embedding layer. Now, we can create a convolutional neural network. We chose CNN because the attributes that make CNN useful for images also apply to text. They preserve the relationship and order of the input, because pixel order is important in image processing, and text order is also important in classification.

#### 4.2.3 RNN-LSTM

Recurrent Neural Network (RNN) is similar to CNN and has internal state (memory) to process the input sequence. We use long short-term memory (LSTM) to preserve the input order (words make up a sentence) and help understand the context for better classification. We use the default Keras embedding layer to generate a vector of length 300. First encode the text data so that each word is represented by a unique integer. This data preparation step can be performed using the Tokenizer API included with Keras, and we add padding to make all vectors have the same length. The embedding layer requires the specification of the vocabulary size, the size of the real-valued vector space EMBEDDING_DIM = 300, and the maximum length of input documents max_length. Now we are ready to define the neural network model. The model will use the embedding layer as the first hidden layer. The embedding layer will be initialized with random weights and will learn the embedding of all words in the training data set during model training.

## 5 Experiments

### 5.1 Data Preprocessing and Augmentation

Prior to performing any machine learning on the dataset, it was required that we parsing the log dataset and cleanse it of impurities and discrepancies. Our preprocessing steps included:

1. Convert all Character to Lowercase:

   In order to facilitate subsequent data processing, we convert all text to lowercase letters at first.

2. Remove MLP in Event Message:

   The MLP in the message such as 'FF:F2:9F:15:7E:DF:00:0D:60:EA:81:20', it is beneficial to remove the MLP for later analysis, because when we delete the numbers and punctuation, we will be left with some words like "FF" that have no meaning.

3. Remove All Numbers and Punctuation:

   We remove all numbers from 0 to 9, we remove all punctuation such as '?','@','!',etc...

4. Remove Single Character:

   After we remove the numbers and punctuation, the IP address is left with a single character like "E", which is unusable for subsequent analysis.

5. Normalize White-space:

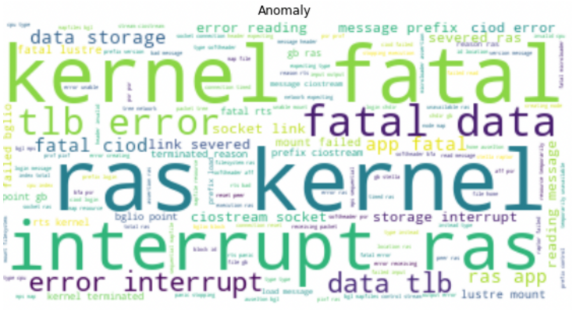


Figure 4: Word cloud: Anomaly

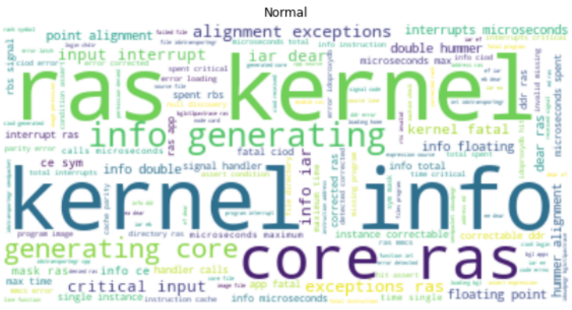


Figure 5: Word cloud: Normal

Since we replaced the numbers with spaces, there will be some extra spaces, so we will replace all the spaces that appear more than twice into one.

6. Remove Stop-words:

   We removed the stopwords such like 'to', 'on', 'of' and 'for' in text of our dataset, The set of stopwords we token from a Python library called 'nltk'. For later task, stopwords are removed from the given text so that more focus can be given to those words which define the meaning of the text.

### 5.2 Exploratory Data Analysis

After the data pre-processed, we found that we have a class imbalance. This is because anomaly class is much more rare than normal class, so this is a usual problem for anomaly detection. We ignored the class imbalance because we will assess multiple metrics for model performance. For modeling and visualization purpose, we use BOW to convert log messages into feature vectors. The valuable information within these log messages allowed us to infer the words 'fatal', 'error', and 'interrupt' are good indicator for anomaly logs. Whereas, 'info', 'core', and 'generating' are good indicator for normal logs, as seen in Figure 4 and 5.

The major concern of the data is that the classes of log messages are imbalanced. The majority of the class are normal which makes sense as anomaly messages are rare in modern systems. This suggests that we need to take this problem into consideration as we evaluate our models.

### 5.3 Evaluation Metrics

Accuracy cannot be used as the only metric because of the classes are imbalance. We have also considered precision, recall and F1 score. Precision of the model denotes how many were correctly identified among the anomaly cases the model predicted. Recall of the model denotes how many were correctly identified among all anomaly cases. F1 score is a harmonic mean of precision and recall. Thus, we want all these metrics to be high.

### 5.4 BOW

BOW is used to convert the log messages to a matrix of word counts. Each row represents a "log message". Each column represents a word in the vocabulary (the set of unique words) in the training data. Each cell represents how often the word occurs in the document. However, some of the words have no distinct meanings such as 'I', 'and, 'a'. To make the training process computational cost faster, we decided to remove those words from the logs and limited the number of word vectors to 200. As a result, the number of features has been drastically decreased.

To make the training process computational cost faster, we decided to remove some of the words from the text. We removed stop words like 'I,' 'and a' has no degrading meaning and is used to make the sentence grammatically correct. As a result, the number of features has been drastically decreased.

| label | ac | active | address | alerts | alignment | allreduce | app | apps | asic | assembly | ... | tlb | torus | total | tree | type | urgent | vnm | volt | vpd | warning |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Anomaly | 0 | 0 | 0 | 0 | 0 | 0 | 63845 | 0 | 0 | 0 | ... | 152735 | 12 | 4751 | 6148 | 18178 | 0 | 0 | 0 | 0 | 0 |
| Normal | 20895 | 69719 | 119946 | 17341 | 588055 | 22528 | 164691 | 77116 | 17513 | 17309 | ... | 296 | 34560 | 309330 | 15083 | 13626 | 29952 | 25090 | 35026 | 44501 | 34585 |

Figure 6: Bag of Words

#### 5.4.1 Logistic Regression

Logistic regression was the first model that we constructed. Each class was considered independent in this approach, therefore, we fed as input the converted massages by BOW and use sklearn's default LogisticRegression package trained a model for two classes on training data. The total training time around 43s. Predictions on validation were com-

parable but in order to solve overfitting, there was still scope for improvement. So, we set the max iteration as 300 and the class of weight as balanced. A test precision of 0.99999 was obtained and recall is 0.99997, F1 score is 0.99998.

#### 5.4.2 Naive Bayes

Our second model is Naive Bayes. To do this, we fed as input the converted massages by BOW into sklearn's naive_bayes package. The approximately training time is 23s, this is much faster than Logistic Regression. We tune the alpha parameter using cross validation to find the best performance for each toxicity label. Finally achieved an precision score 0.99154 on test data and recall is 0.99871, F1 score is 0.99509.

#### 5.4.3 Random Forest

The third model is Random Forest, which is a classification algorithm consisting of many decisions trees. The total training time is around 92s, which is much longer than Logistic Regression and Naive Bayes because it uses bagging and feature randomness when building each individual tree. Random forest is robust to overfitting. The precision score on test set is 0.99999, recall is 0.99998 and F1 score is 0.99998.

### 5.5 Word Embedding

Another advanced text representation we used is word embedding. The input data is prepared by tokenizing the text, convert words to index, padding the sequence to preserve a set size, and replacing the index with pre-trained word vectors using the Keras package in Python. This results in a 2-dimensional sequence representing the comments data that would be used to train a Neural Net. The validation and test data is also converted to a 2-dimensional array that will be used for tuning and testing.

#### 5.5.1 Word2vec CNN

To implement CNN architecture, we use Google Co-laboratory as a platform to run Jupyter Notebook with GPU support. For the CNN model we varied kernel size, pool size, the number of hidden layers, batch size and epoch. As Figure 8, an example of performance vs epoch. The training time is much longer than traditional machine learning, but it is acceptable.

For input with Word2Vec embedding, training time for each epoch is around 330s using Google Co-laboratory with GPU support. We use Adam optimizer with learning rate 0.01 to train this model. Adam optimization is a method of stochastic gradient descent focused on first-order and second-order moments' adaptive estimation (Kingma and Ba, 2017). Finally the precision on test dataset is 0.99994, test recall is 0.99994 and F1 score is 0.99980.

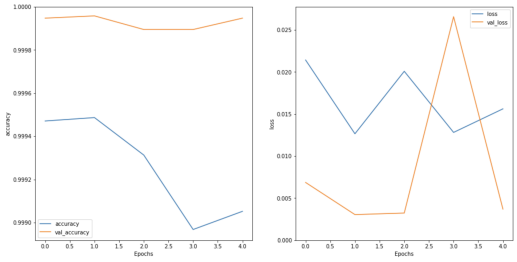


Figure 7: Performance vs Epoch

#### 5.5.2 GloVe CNN

For input with GloVe embedding, training time for each epoch is around 93s using Google Co-laboratory with GPU support. We found the Gradient descent optimizer (GSD) with learning rate 0.001, momentum 0.9, and decay 0.01 has the best performance on both training set and validation set. Finally, the recall is 0.9271, precision is 0.9271, and F1 score is 0.4811 on test dataset.

#### 5.5.3 RNN-LSTM

A recurrent neural network is similar to a CNN in that it has an internal state to process the input sequences. We use LSTM because it preserves the input sequences and assist in understanding the context for better classification. To generate a vector of length 300, we used the default Keras embedding layer.

The data is prepared with Keras embedding. The 2-dimensional input is used to train a LSTM model with one hidden layer at 64 output units per layer. It took around 182s to train the model on Google Co-laboratory with GPU support for each epoch. By incorporating additional layers, which would take more computational power, the model can be further improved. On the test dataset, this model achieved 100% recall, 100% precision, and 100% F1 score, which is the best performance of all models.

### 5.6 Results

Table 1 is the results on test data of the BOW method we tried:

Table 1: Method 1 Results

| Models | Precision | Recall | F1 |
|---|---|---|---|
| Logistic Regression | 0.99999 | 0.99997 | 0.99998 |
| Naive Bayes | 0.99154 | 0.99871 | 0.99509 |
| Random Forest | 0.99999 | 0.99998 | 0.99998 |

Table 2 is the results on test data of the word embedding method we tried:

Table 2: Method 2 Results

| Models | Embedding | Precision | Recall | F1 |
|---|---|---|---|---|
| CNN | Word2Vec | 0.99994 | 0.99994 | 0.99980 |
| CNN | GloVe | 0.92710 | 0.92710 | 0.48110 |
| LSTM | Keras Embedding | 1.0 | 1.0 | 1.0 |

#### 5.6.1 Overall Quantitative Results

We used F1 score to measure the overall quantitative results because it combines the precision and recall of a classifier into a single metric by taking their harmonic mean.

For BOW method, we achieved a best performance of 0.99998 F1 score on both Logistic Regression and Random Forest model. On the other hand, the Naive Bayes model achieved 0.99509 F1 score.

For word embedding method, we achieved a best performance of 1.0 F1 score on our LSTM. And CNN with Word2Vec embedding achieved 0.9998 F1 score, but CNN with GloVe embedding achieved 0.4811.

Overall, LSTM on Keras Embedding performed the best on our classification task. In addition, the average F1 score of BOW method is outperformed the word embedding method.

#### 5.6.2 Overall Quantitative Evaluation

Compare Logistic Regression with Naive Bayes, Naive Bayes models the joint distribution of the feature and target label, and then predicts the posterior probability given as p(feature — target). Logistic regression directly models the posterior probability of p(target — feature) by learning the input to output mapping by minimizing the error (Ottesen, 2017). Naive Bayes assumes that all features are conditionally independent. However, in our case, the features are correlated to each other, therefore, the performance of Naive Bayes is poor compare with other models. And compare with Logistic Regression with Random Forest, their performance is almost the same and both are very close to 1. But in general, logistic regression performs better when the number of noisy variables is less than or equal to the number of explanatory variables, and the true and false positive rate of random forest is higher as the number of explanatory variables in the data set increases(Kirasich and Sadler, 2018).

Both word2vec and gloves enable us to represent a word in vector form. They are the two most popular word embedding algorithms that can reveal the semantic similarity of words and thus capture different aspects of word meaning (MLNerds, 2019). Word2vec embedding is based on training a shallow feed forward neural network, while glove embedding is learned based on matrix factorization technology (MLNerds, 2019). The structure of glove is simpler than word2Vec, so glove is faster than word2vec. Also, word2vec focuses on local information, and glove focuses on local and global information. In our case, we train CNN model using two different word embeddings, and the results of CNN using Word2Vec has better performance.

According to the result from Table 2, LSTM is the best performance in our test because when training the neural network for classification problems, we can learn the word embedding of our dataset. Before presenting the text data to the network, the text data is first encoded so that each word is represented by a unique integer. We perform this data preparation step using the tokenizer API that comes with Keras, and we add padding so that all vectors have the same length. The model uses the embedded layer as the first hidden layer. The embedding layer will be initialized with random weights, and will learn to embed all the words in the training data set during the model training(Nabi, 2018).

In general, from the obtained results Table 1 and Table 2, we can see the performance of all achieve a high degree of precision and recall, but the average F1 score of traditional learning models fed by the BOW representations is higher than deep learning model fed by word embedding. But this not indicates BOW representation enable the traditional machine learning model to better classifying anomaly data because all the results are close to 1, we can't say which model is better at detecting anomalous logs. Moreover, the results in literature BOW vs Word Embedding for real-time event

detection (Nguyen et al., 2019) shows the Word Embedding is outperformed BOW.

### 5.7 Conclusion

In this paper, we have investigated two methods which are BOW and word embedding. The LSTM with word embedding yielded the best performance. It seems to be extracting semantics could help us hinder the content. Natural language processing is a vast area and there is a lot of scope for experimentation. We have done many researches on the conversion of text data using techniques such as 'Word2Vec' and 'GloVe' were learned.

In the feature, we aim to achieve better performance on Neural Networks by trying different optimizers using more complex deep learning models and tuning hyper parameters. We would like to assess the effect of using Van and Tien's (Nguyen et al., 2019) CNNs on bag of word models.

## References


Jason Brownlee. 2017. What are word embeddings for text?

Yiyong Chen Dan Lv, Nurbol Luktarhan. 2021. Conanomaly: Content-based anomaly detection for system logs.

Md Zahidul Islam, Jixue Liu, Jiuyong Li, Lin Liu, and Wei Kang. 2019. A semantics aware random forest for text classification. page 1061–1070.

Yoon Kim. 2014. Convolutional neural networks for sentence classification.

Diederik P. Kingma and Jimmy Ba. 2017. Adam: A method for stochastic optimization.

Trace; Kirasich, Kaitlin; Smith and Bivin Sadler. 2018. Random forest vs logistic regression: Binary classification for heterogeneous datasets.

Hongmin Li, D Caragea, X Li, and Cornelia Caragea. 2018. Comparison of word embeddings and sentence encodings as generalized representations for crisis tweet classification tasks. *en. In: New Zealand*, page 13.

MLNerds. 2019. What is the difference between word2vec and glove ?

Javaid Nabi. 2018. Machine learning — word embedding sentiment classification using keras.

Van Quan Nguyen, Tien Nguyen Anh, and Hyung-Jeong Yang. 2019. Real-time event detection using recurrent neural network in social sensors. *International Journal of Distributed Sensor Networks*, 15(6):1550147719856492.

Christopher Ottesen. 2017. Comparison between naïve bayes and logistic regression.

Ayush Pant. 2019. Introduction to logistic regression.

Bo Wang, Qingyi Hua, Haoming Zhang, Xin Tan, Yahui Nan, Rui Chen, and Xinfeng Shu. 2022. Research on anomaly detection and real-time reliability evaluation with the log of cloud platform. *Alexandria Engineering Journal*, 61(9):7183–7193.