Ham Or Spam?
SMS Classification Using Machine Learning

Matan-Ben Nagar[1] **and Asahel Cohen**[2]

[1]**matannagar@gmail.com**
[2]**asahel.c@gmail.com**

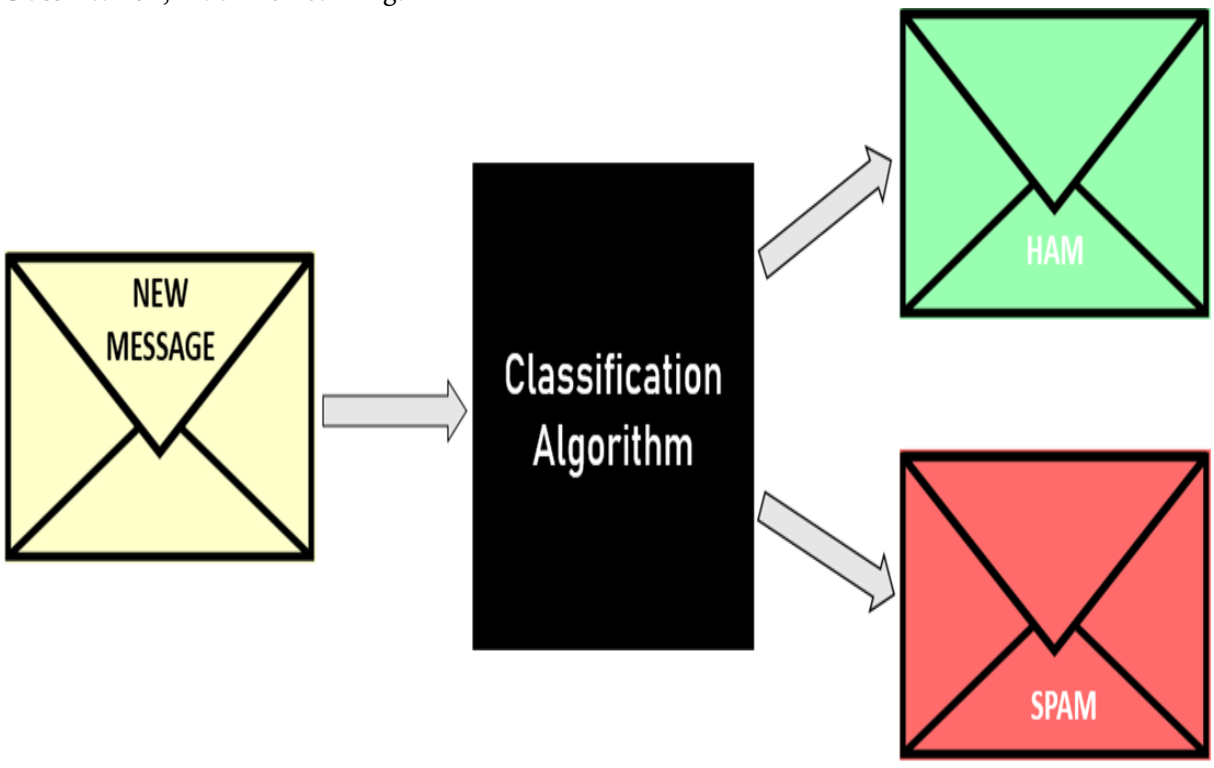
The article was written as part of a course in computer science first degree. The course emphasises on deep learning techniques for analyzing coded languages.The course is run by Dr. Or Anidjar.
Link to code:
`https://github.com/matannagar/SMS_Spam_Detection_Web.git`

Abstract

In our modern era the number of people who have cellphones is high and only rising. But there is a problem that everyone agree on, we receive way to many text messages from unknown senders that are either hackers or scammers. So in order to solve that problem we can use a model based on machine learning, that well classify those SMS's as spam, and we will never even need to them.

Keywords

Deep Learning, Date model, Spam and Ham, Text Classification, Machine Learning.



1. Introduction

SMS is one of the most used forms of communication. Almost everyone uses this service of communication. That is why a lot of government organizations use SMS for communication with their clients, such as banks, medical organizations etc... so due to this high amount of SMS usage, it has become a target for hackers and spammers. The hacker can easily send and transmit a "dangerous" link(control your phone, pass a virus...). Or if its only receiving a spam SMS that annoyingly makes you look at your phone. In order to prevent hackers and scammers, we must create a system that tells the end user whether the SMS is SPAM or HAM(non-spam message). so we created a system based on a deep learning model. for those who don't know what Deep Learning is we will explain it shortly in the rest of the introduction. for those who know you can go ahead and skip to the next section. Deep learning is a type of machine learning and artificial intelligence (AI) that imitates the way humans gain certain types of knowledge. Deep learning is an important element of data science, which includes statistics and predictive modeling. there are three types of machine learning. Supervised learning, Unsupervised Learning, and Reinforcement learning. Each works in a different way. Supervised learning works by receiving data that is split to features, and is labeled. By learning the data the machine try to find a way to "guess" the label of similar data. Unsupervised Learning the machine receive data that is split to features but not labeled, and the machine tries finding a pattern within the data and finally split the data to its own anticipated label. Reinforcement learning we give the machine data and ask it to classify it, after it is classified the developer tells the machine whether it was right or wrong and from that the machine learns for the next time.

2. Methodology

2.1 Workflow:

Data collection: The first phase of the work is finding a collection of data on which we will preform the learning. Data Cleaning: In this stage we organize the data, give it user friendly label's... Train and Test: split the data to train and test(while making sure that the data is split equally). Model: find a model that is suited for the data, and for the experiments goals. Train Data: using the model we'll train with the train data. Test Data: we check out loss on the test data set, if the loss is high we need to train the model more. Prediction: User can text in SMS message and a prediction is given whether it is spam or ham.

2.2 our Workflow:

Data collection:
After searching for the right data set we found our data set on kaggle:
`https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset`

Data Cleaning:
In this stage we organize the data, give it user friendly label's...

Train and Test:
split the data to train and test(while making sure that the data is split equally).

Model:
find a model that is suited for the data, and for the experiments goals.

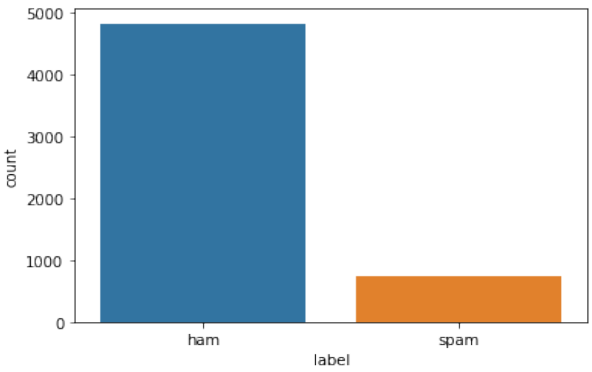
Train Data:
using the model we'll train with the train data.

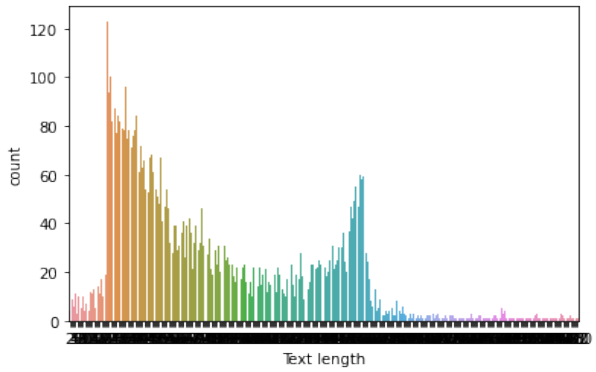
Test Data: we check out loss on the test data set, if the loss is high we need to train the model more.

Prediction:
User can text in SMS message and a prediction is given whether it is spam or ham.

3. Dataset Description

As shown before the dataset used can be found on kaggle:
`https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset`. There are over 5,000 unique values. Each value is a unique text message, which is labelled as Spam or Ham.



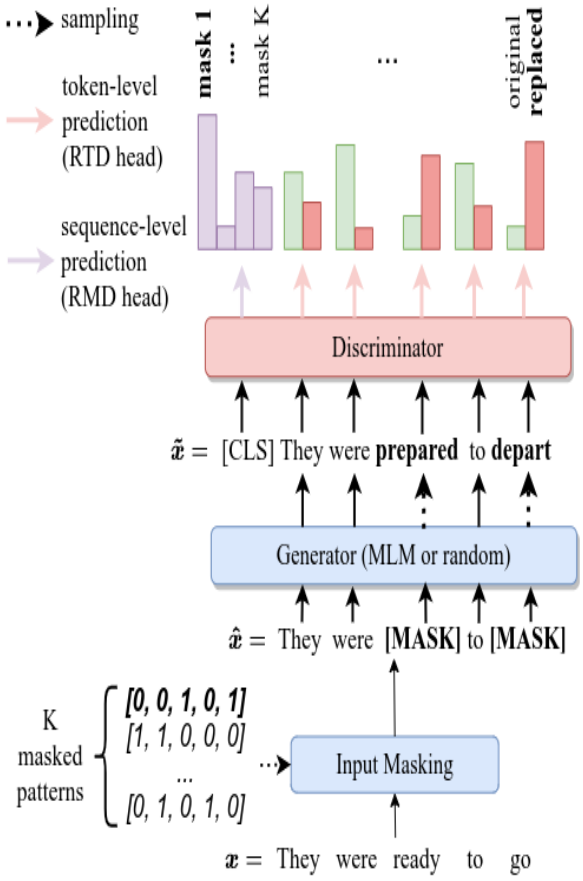


3.1 Preprocessing

In order to send the data to the model there is need to preprocess the dataset. The data starts off very organized so there is not much work ot be done in this stage. Start by removing NaN columns, replace value of ham and spam to 0 and 1 so that the model can see them as values.

4. Deep Learning model for text classification

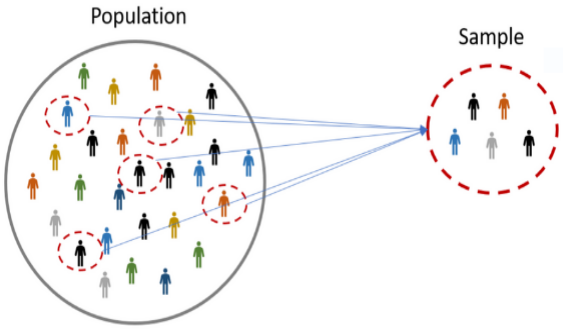
The model that was chosen to use in order of Detecting Anomalies in Text via Self-Supervision of Transformers for the SMS classification is a big-ML model called Date. The model was developed by Andrei Manolache, Florin Brad, and Elena Burceanu. you can find the model in: `https://github.com/bit-ml/date`. Date works by learning how a normal text looks like in an unsupervised or semi-supervised fashion by designing a self-supervised task for the Transformers. The network is built over two components: Generator and Discriminator.



4.1 Date

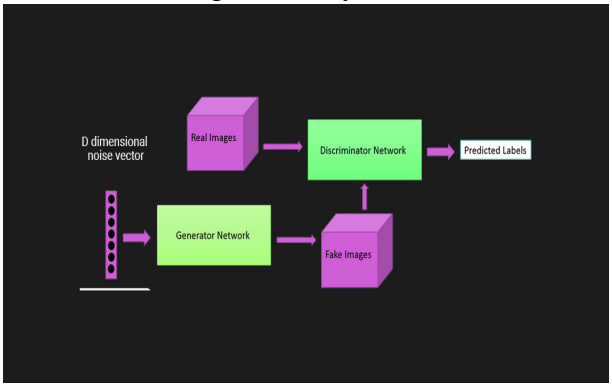
In order to understand how Date works, an understanding of Dates main idea must be made. Dates main idea is to learn how a normal text would look liek in an unsupervised or semi-supervised way, by designating a self-supervised task for transformers. we will go through step by step.

1. Sampling.
2. Input masking.
3. Generator(MLM or random).
4. Discriminator.
5. token- level prediction(RTD head).
6. sequence-level prediction(RMD head).



4.1.1 Sampling
Sample a masking pattern. From a collection of pre-generated masks.

4.1.2 input masking
An input mask is a string of characters that indicates the format of valid input values. You can use input masks in table fields, query fields, and controls on forms and reports. The input mask is stored as an object property. You use an input mask when it's important that the format of the input values is consistent. The masking is done by the Generator.



4.1.3 Generator(MLM or random)
The generator part of a GAN learns to create fake data by incorporating feedback from the discriminator. It learns to make the discriminator classify its output as real. So in our experiment the generator that samples tokens from a word distribution and corrupts the input.

4.1.4 Discriminator
The discriminator in a GAN is simply a classifier. It tries to distinguish real data from the data created by the generator. It could use any network architecture appropriate to the type of data it's classifying. Here we use a discriminator with two heads: first detects which token is corrupt and which is original, and the second detects which masking pattern was used over the input text(Token detector and Mask detector).

4.1.5 token- level prediction(RTD head)
by using the RTD head the model will guess which tokens were corrupt.

4.1.6 sequence-level prediction(RMD head)
Sequence prediction is a problem that involves using historical sequence information to predict the next value or values in the sequence. The sequence may be symbols like letters in a sentence or real values like those in a time series of prices. This is part of the discriminator job, by using the RMD head the model will guess what masking patterns were applied.

4.2 Date Train: Maths
Formally, the goal is optimize the combined loss for the RTD and RMD heads:

$$\underbrace{\mathbb{E}\left[\sum_{i=1}^{T} -\log P_D(m_i|\tilde{x}(m);\theta_D)\right]}_{\mathcal{L}_{RTD}} + \underbrace{\mathbb{E}\left[-\log P_M(m|\tilde{x}(m);\theta_D)\right]}_{\mathcal{L}_{RMD}}$$

4.3 concepts
- unsupervised-
  Unsupervised learning, also known as unsupervised machine learning, uses machine learning algorithms to analyze and cluster unlabeled datasets. These algorithms discover hidden patterns or data groupings without the need for human intervention.

- semi-supervised-
  Semi-supervised machine learning is a combination of supervised and unsupervised machine learning methods. With more common supervised machine learning methods, you train a machine learning algorithm on a "labeled" dataset in which each record includes the outcome information.

- self-supervised-
  Self-supervised learning obtains supervisory signals from the data itself, often leveraging the underlying structure in the data. The general technique of self-supervised learning is to predict any unobserved or hidden part (or property) of the input from any observed or unhidden part of the input.

- Transformers-
  A transformer is a deep learning model that adopts the mechanism of self-attention, deferentially weighting the significance of each part of the input data. It is used primarily in the

fields of natural language processing (NLP) and computer vision (CV).

5. Experimental Setup and Results

This model was implemented in python 3.9 with Cuda 11.2, using libraries such as pandas, logging, transformers and io.

5.1 Setup

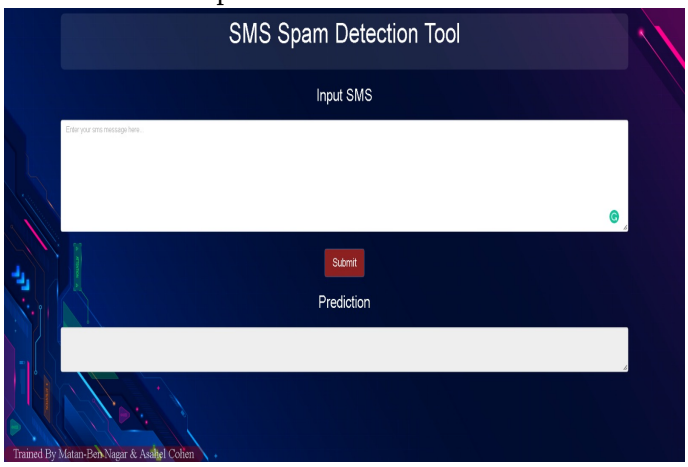
During the building of the project, we were put through a lot of challenges.

5.1.1 Work Environment

In order to train the model,we started of by using google collab. collab is known for its compatibility and easy start when attempting ML projects. The model was easily trained on the collab and it only took a few minutes to finish and to generate the outputs folder (where the trained model is stored). However, we wanted to debug our code and the training process. That is when we attempted moving the code from collab to our local PyCharm IDE. Once we moved to Pycharm, difficulties with compatibility started occurring. My partner and I both attempted to install the necessary libraries and IDE on our VM environment but failed in doing so. One VM collapsed and could not reload, the entire environment had to be deleted and reinstalled. On the other PC, the program did work but could not utilize the GPU for faster training. Other bugs began to occur when we attempted to modify some of the code. For example, the model could not finish the training. It was stuck on tokenizing the data. It was then that we changed some of the settings of the VM and then it stopped working on the other computer as well. Finally, we moved the whole project onto the windows environment. The model did manage to utilize the Cuda (with Cuda.is available()) but even then the model was stuck on tokenizing the data. After researching online, we understood that there was a problem with the Torch version we had. Downgrading the torch allowed us to finish the training, however, we lost the ability to utilize the GPU. We then decided to continue with the project without using the GPU.

5.2 Website and Docker

Building the UI required some HTMLand Flask experience. Matan had already built a similar website, so it was easy for us to duplicate the UI project. Of course, we still encountered problems. At first, the web app refused to load the model from the folder that was located inside the projects folder. We had to access the trained tokenizer instead and through it we were able to perform our predictions. Launching the app on Docker was a day's challenge. The environment wouldn't load several times, a process that took approximately 20 minutes each. After a while, we realized we were creating the docker image without activating the virtual environment which made the entire process that much longer. At first, we tried to download an Ubuntu image, but when it tried to install the requirements.txt file it failed because some of our libraries were meant for windows and not Ubuntu. This happened because we created the project on Windows. We then changed the image again to install a Python environment. This time we managed to get most of the libraries from the requirements.txt file. Unfortunately, there were 2 packages that no matter what we tried, refused to download and caused the entire process to fail. After a while we decided to remove them, finally having a successful image running. Although the model was slower on the Docker, it managed to predict correctly from the user's input.



5.3 Experimental and Results

5.3.1 Test Running

In order to save time and CPU resources while training, instead of running the model every time, the results are saved to a folder called out puts. That way the next time the model is trained it can start by loading the outputs of the previous session and better the results.

5.3.2 Epochs

In order to get the best results we tested the same model with different number of epochs. We tried is with:

- 1 epoch.

- 2 epochs.
- 3 epochs.

and the results were as following:
. . . 1 epoch . . .
loss: 0.027541279721689144.
. . . 2 epochs . . .
loss: 0.04065434819172619.
. . . 3 epochs . . .
loss: 0.053156784460640795.
As seen after a few tries we concluded that the lower amount of epochs gave the best result. So one epoch was chosen.

6. Setup

6.1. Dependencies

- Please refer to .The pyTorch website to install PyTorch.
- Make the appropriate adjustments to enable Cuda on your PC for faster experience.

6.2. Installing

- run in terminal git clone for SMS Spam Detection Web git.
- Create a virtual environment using Python -m venv.
- run pip install -r /path/to/requirements.txt .
- download The Trained Model and extract to the root repository https://drive.google.com/file/d/1mI19Mb4IQC4ScndHirmGIKyn6EXI9Swu/view

6.3. Executing Program

CMD:

- Inside the folder load CMD.
- run: python application.py.

PyCharm:

- load the projects folder and press Run.
- The website will automatically load.
- Insert your messages and press Predict.

7. Conclusion

The primary goal of this paper is to find a way to reduce the number of Spam SMS's received by users. as seen from our results there is still space for improvement but we have shown that results are good non the less. We hope that models like this could be used in every message based platform, so that the end users wont need to worry of being spammed.

Acknowledgements

We would like to thank our lecturer Dr. Or Anidjar for guiding us throughout the process. And we would like to thank Andrei Manolache, Florin Brad, and Elana Burceanu for writing the Date model which was used.

References

[1] Andrei Manolache, Florin Brad, and Elena Burceanu. Date: Detecting anomalies in text via self-supervision of transformers. In *NAACL*, 2021.

[1]