

# Final Report

## Japanese Profanity filter with Sentiment Analysis

Acharya Ashok

[@ashok314](#)

Aug 5<sup>th</sup>,2020

# CONTENTS

LIST OF FIGURES.....	3
ABSTRACT .....	4
INTRODUCTION .....	4
PROCESS.....	5
<b>Part-of-Speech Tagging (User dictionary) .....</b>	<b>5</b>
Steps .....	6
Create user file.....	6
Create a user dictionary .....	6
Apply changes to the definition file .....	6
<b>Sentiment Analysis.....</b>	<b>7</b>
Data Collection.....	7
Analysis Techniques .....	8
Data Preprocessing.....	8
Labeling .....	9
Data splitting:.....	9
Data Visualization .....	10
Training.....	11
Sample Code.....	13
Prediction.....	14
Prediction Result .....	14
Sentiment Analysis Result .....	15
PROFANITY FILTER .....	16
<b>Background.....</b>	<b>16</b>
<b>Implementation.....</b>	<b>16</b>
<b>Result .....</b>	<b>17</b>
CONCLUSION .....	18
FUTURE WORK.....	18
REFERENCES.....	19

## List of Figures

☒ 1. SNAPSHOT OF SYSTEM AND USER DICTIONARY OUTPUT .....	5
☒ 2. POSITIVE WORDS CLOUD .....	10
☒ 3. NEGATIVE WORDS CLOUD.....	10
☒ 4. PROFANE WORDS CLOUD .....	11
☒ 5. RESULT OF TRAINING CHANGING HYPERPARAMETERS.....	12
☒ 6. TRAINING AND VALIDATION ANALYSIS .....	12
☒ 7. PREDICTION ACCURACY ON THE TEST SET .....	15
☒ 8. PROFANITY FILTER WITH SAMPLE USER INPUT (NON-PROFANE)	16
☒ 9. PROFANITY FILTER WITH SAMPLE USER INPUT (PROFANE) .....	17

## Abstract

The Internet has become an integral part of our daily life. Text-based communication is one of the widely used categories among voice, video, and others. Emails, SMS texts, social media interaction are used by a company or an individual for communication. The text is normally faster and convenient. The downfall is, it depends on the interpretation of the reader especially when it is being conveyed with emotion. Mainly in a social media interaction, readers and writers of the text are either anonymous or mostly unknown. The use of informal or bad-words is poisoning the internet. In the case of Japanese text, profanes are used very less compared to other languages like English or Russian. Identifying such profane words is difficult since state-of-art Japanese like IPADIC fails to correctly parse these words. Correctly parsing and possibly filtering or censoring such words is important.

## Introduction

Identifying and correctly tagging the Part-of-Speech for profane words can be done using a readily Morphological Analyzer. Mostly used tagger Mecab<sup>[1]</sup> or similar tools like Kuromoji allows us to access their system dictionary and make changes. Another way around is to create your user dictionary and configure the tagger to use it. In the following project, Mecab is used with utilizing its user dictionary feature. It has the standard format in which we can specify our words and manually tag the part-of-speech like “名詞, 形容詞, 一般” etc.

When we can correctly tag the POS for these words it can be further utilized to analyze the sentiment of the word or a sentence. If We can correctly identify the words in a sentence it opens up to the various possibilities like filtering or censoring, sentiment analysis. It has applications like profanity filter in online gaming where two or group of unknown people play and chat with each other, censorship in social media comments and interactions, and polarity (profane or non-profane) analysis.

## Process

## Part-of-Speech Tagging (User dictionary)

Mecab allows us to define our user dictionary following its format. This step has to be done manually for the words which are not correctly tagged by the IPDIC because the words are entirely new for the dictionary. There are other improved dictionaries like “mecab-ipadic-neologd<sup>[2]</sup>” neologisms (new word) based on the many language resources on the web. The following snapshot shows the differences in using the user dictionary.

```
ashok@acharya:~/Desktop/nlp$ gedit /usr/local/etc/mecabrc
ashok@acharya:~/Desktop/nlp$ /usr/lib/mecab/mecab-dict-index -d /usr/local/lib/m
ecab/dic/ipadic/ -u user.dic -f utf-8 -t utf-8 "/home/ashok/Desktop/nlp/test.csv
"
reading /home/ashok/Desktop/nlp/test.csv ... 366
emitting double-array: 100% |#####|

done!
ashok@acharya:~/Desktop/nlp$ mecab
しんじまえ
しん 名詞,一般,*,*,*,しん,シン,シン
じまえ 名詞,一般,*,*,*,自前,ジマエ,ジマエ
EOS
^[[A^C
ashok@acharya:~/Desktop/nlp$ mecab -u user.dic
しんじまえ
しんじまえ 名詞,一般,*,*,*,しんじまえ,しんじまえ,しんじまえ
EOS
^C
```

### ☒ 1. Snapshot of system and user dictionary output

The above snapshot (Fig 1) shows the process of creating and using the system dictionary. The figure shows the Mecab's command-line interface. "mecab" command uses the system dictionary to tag the words or sentences provided as an argument. "mecab -u user.dic" command is used to point the mecab to the user dictionary (dot dic format) created as the following process.

The following are the steps to create define and use the user dictionary.

## Steps

### Create user file

Create csv with the following format

表層形,左文脈 ID,右文脈 ID,コスト,品詞,品詞細分類 1,品詞細

分類 2,品詞細分類 3,活用型,活用形,原形,読み,発音

---English ---

Surface type, left context ID, right context ID, cost, part of speech, sub-OS 1, sub-POS 2, sub-POS 3, conjugation type, conjugation form, original form, reding, pronunciation

For eg;

しんじまえ,-1,-1,-1,名詞,一般,\*,\*,\*,しんじまえ,シンジマエ,シンジマエ

### Create a user dictionary

Use command mecab-dict-index with

-d option to point to the dictionary

-u user dictionary output file

-f -t encoding UTF

And a CSV file containing user words as an argument.

```
$/usr/lib/mecab/mecab-dict-index -d /usr/local/lib/mecab/dic/ipadic-neolodg/ -u ja_profane_words.dic -f UTF8 -t UTF8 "/home/ashok/Desktop/nlp/mecab_tagger/ja_profane_words.csv"
```

\*(detail about data in data collection section)

### Apply changes to the definition file

add the following line, which points to the user dictionary created above

```
>> userdic= your user directory/user.dic
```

```
$vi /usr/local/etc/mecabrc
```

So, the user dictionary can be created successfully and used from the command line or the program call (Python).

# Sentiment Analysis

## Data Collection

\*Data for user dictionary is collected from the Wikipedia.

/wiki/Japanese\_profanity contribute only about 100 words in total of 460 profane words. Rest of the data are collected from 放送禁止用語一覧

monoroch -モノロク [3].

Data for the sentiment analysis are collected from various sources. The main focus of the source of data was the web, specifically social media interactions.

I gathered readily available datasets from the data.world<sup>[4]</sup> and Kaggle<sup>[5]</sup>.

These datasets are available under category bad-words with some variant tags (insults/offense/comments/explicit content). Since it contributed to the small amount, I used some related English datasets and translated it to Japanese.

Furthermore, I used webscrapper<sup>[7]</sup>, a web scraping tool to scrape some subreddit content and comments. Subreddit like r/steamr/. The sitemap is created to extract the required data from the site. A site map is a JSON file containing information to extract from the selectors and its values from the website. Since comments and posts can be large, navigation to the next post and content should be included in the site map.

For each post, we can either manually create a site map or automate it via a program. I manually selected the specific post for scrapping. Post which has most upvotes were chosen for time-saving.

Using browser extension is found to be straight. We can have as many sitemaps for every and unique post. After importing and saving the sitemap, it works on clicking scrap from the menu. Takes time according to the number of comments for a post. Once scrapping is finished, we can choose to download the file as CSV or other useful text formats.

Downloaded data is saved under some directory for proceeding to the analysis. Data collected from the above method needs to be cleaned. As it contains the scrap id, navigation URL, post URL which will not be useful in further processes. Data values mentioned in the above sitemap are saved in a suitable format(csv/tsv).

## Data Description

As we choose the data for ourselves, it is obvious that data is familiar. Collected data is a text data containing a string of Japanese characters. Other English characters, numbers, emojis, kaomojies are removed in further process.

## Analysis Techniques

For analysis of the collected data machine learning techniques are used. Machine learning tools are available for various data formats. Here we use the available tools for text processing. Tool used in this project is FastText<sup>[6]</sup>, an open-source tool by Facebook for machine learning. Because of the high-performance metrics of the FastText in machine learning, it is chosen for this project. As in the general machine learning process, the following processes are carried out.

## Data Preprocessing

A very important step of any machine learning project is data collecting and processing. Collected data contain noise and need to be cleaned before feeding to the machine learning algorithm.

Data may contain unwanted text and other special characters or emojis. This process involves the cleaning of data. There's a lot of ways we can clean our data. Initially, the Following processes were carried out while cleaning data.

- i. Normalizing

When it comes to the language other than English, normalizing is an obvious step. Normalizing involves Unicode normalization Python library *unicodedata* is used to normalize the data.

- ii. Removing Special character:

Since data is directly from the users, it may contain different special characters in it. It's not meaningful to use such characters in our data. Special characters are removed by matching the regular expression for them.



- iii. Removing Emojis and kaomojies:  
Emojis are eliminated from the data by matching the Unicode pattern for emoji and removing the matched pattern.
- iv. Removing Spaces, spaces-between, newline, etc:  
These are removed using Python library *re* by matching and replacing with nothing.
- v. Tokenizing  
Data is then tokenized using the Python library *MeCab*. While tagging, our previously made user dictionary is utilized to correctly tag and parse.

## Labeling

Since we are using FastText, it is required to label the data prefixed with `__label__`. Each line of the text data file is prefixed with the label. This can be carried out either manually editing the text file or using the program to manipulate the file. The text files are large so the manipulating programmatically found to be efficient.

For labeling the data, all the csv files in a directory were combined in one data frame using the Python programming language and *Dask* module.

Data are stores in three different folders, positive, negative, and profane. After this process, the data is well labeled.

```
__label__positive お待ちおりプロフ見  
__label__negative あなたの投稿が気に入らない  
__label__profane バカじゃないの
```

## Data splitting:

Since such well-labeled data is feed to a machine learning algorithm, It is divided into training and test sets. The command-line tool (head, tail, wc) is used to split the data into the desired size. Data before preprocessing is of 18,196 (lines). After cleaning the data, it is 8234 lines. Split into a training set of size 6082 and rest 2152 as a test/validation set.

## Data Visualization

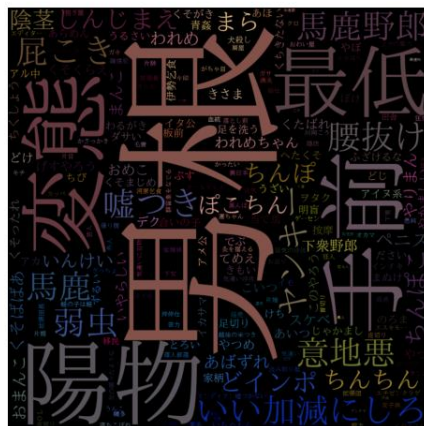
Data are visualized using Python Library *wordcloud*.



図 2. Positive words cloud



図 3. Negative words cloud



☒ 4. Profane words cloud

## Training

Training with FastText is a simple and easy process. Few steps with the processed data result in the machine learning model. Using FastText module for python, a predefined function takes the data as input.

*Train\_supervised()* function is used to create a model. Different metrics can be adjusted to make the model more efficient.

Some of the useful hyperparameters are:

- lr: learning rate
- wordNgrams: An n-gram is a contiguous sequence of max n words from loss functions
- epoch: Number times we go through the entire dataset

These hyperparameters can be tweaked and adjusted for better learning accuracy. Here is the result of altering the hyperparameters.

With the following hyperparameters, the accuracy of 99% is obtained while training.

```
hyper_params = {"lr": 0.225,  
                "epoch": 5000,  
                "wordNgrams": 2,  
                "dim": 20}
```

```
{'lr': 0.255, 'epoch': 50, 'wordNgrams': 2, 'dim': 10},accuracy:0.7588978185993112,validation:0.451784886416319
{'lr': 0.255, 'epoch': 5, 'wordNgrams': 2, 'dim': 10},accuracy:0.7531572904707233,validation:0.45433472415391746
{'lr': 0.5, 'epoch': 5, 'wordNgrams': 2, 'dim': 10},accuracy:0.7531572904707233,validation:0.45433472415391746
{'lr': 0.5, 'epoch': 5, 'wordNgrams': 2, 'dim': 50},accuracy:0.7531572904707233,validation:0.45433472415391746
{'lr': 0.5, 'epoch': 50, 'wordNgrams': 2, 'dim': 10},accuracy:0.7664425127111695,validation:0.45456652758460825
{'lr': 0.5, 'epoch': 50, 'wordNgrams': 2, 'dim': 50},accuracy:0.7664425127111695,validation:0.45479833101529904
{'lr': 0.255, 'epoch': 50, 'wordNgrams': 2, 'dim': 50},accuracy:0.762506150565852,validation:0.4524802967083913
{'lr': 0.255, 'epoch': 500, 'wordNgrams': 2, 'dim': 50},accuracy:0.8755125471543382,validation:0.461984237366713
{'lr': 0.255, 'epoch': 500, 'wordNgrams': 2, 'dim': 20},accuracy:0.8659996719698212,validation:0.45572554473806215
{'lr': 0.255, 'epoch': 1000, 'wordNgrams': 2, 'dim': 20},accuracy:0.9232409381663113,validation:0.45572554473806215
{'lr': 0.5, 'epoch': 1000, 'wordNgrams': 2, 'dim': 20},accuracy:0.9936034115138592,validation:0.461984237366713
{'lr': 0.255, 'epoch': 5000, 'wordNgrams': 2, 'dim': 20},accuracy:0.9937674266032475,validation:0.45989800649049606
{'lr': 0.002, 'epoch': 5000, 'wordNgrams': 2, 'dim': 20},accuracy:0.9937674266032475,validation:0.456884561891516
autotune Progress: 21.1% Trials: 21 Best score: 0.888271 ETA: 0h 3m56s
```

図 5. Result of training changing hyperparameters

The model takes the time learning the words, can be tested with the test set we created earlier by splitting the original set.

Validation also provides important measurements like precision and recall. And the model can be saved(.vec/.bin) for the prediction.

The maximum training time was about 3m56 seconds.

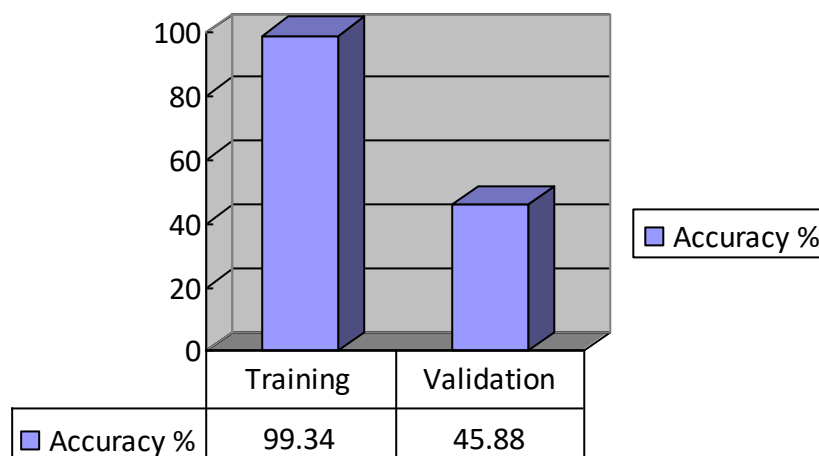


図 6. Training and Validation analysis

## Sample Code

```
import fasttext
import os
hyper_params = {"lr": 0.225,
                 "epoch": 5000,
                 "wordNgrams": 2,
                 "dim": 20}
model = fasttext.train_supervised(input="train", **hyper_params)
#results
result=model.test("train")
validation=model.test("test",k=2)
# DISPLAY ACCURACY OF TRAINED MODEL
text_line = str(hyper_params) + ",accuracy:" + str(result[1]) + ",validation:" +
str(validation[1]) + '\n'
print(text_line)

f_data= open("results", 'a+')
f_data.write(text_line)
f_data.close()

model.save_model("model.bin")
print(model.predict("your sentence"))
```

## Sample Output

```
{'lr': 0.255, 'epoch': 5000, 'wordNgrams': 2, 'dim': 20},
accuracy:0.9937674266032475,validation:0.45989800649049606

Prediction:
(('__label__profane'), array([0.55362666]))
```

## Prediction

The saved model is now used to predict completely new sentences. *Predict()* function takes user input to predict the sentiment of the sentence based on the pre-trained model.

### Sample code run for prediction

```
model = fasttext.load_model("model.bin")  
  
print(model.predict("投稿共感できすぎフォローしまっ"))  
  
-----  
prediction:  
(('__label__positive'), array([0.99984348]))
```

## Prediction Result

Prediction on the test set is carried out with accuracy of the prediction. We can change the value of K and t in our *test()* command to obtain desired precision and recall metrics

Where,

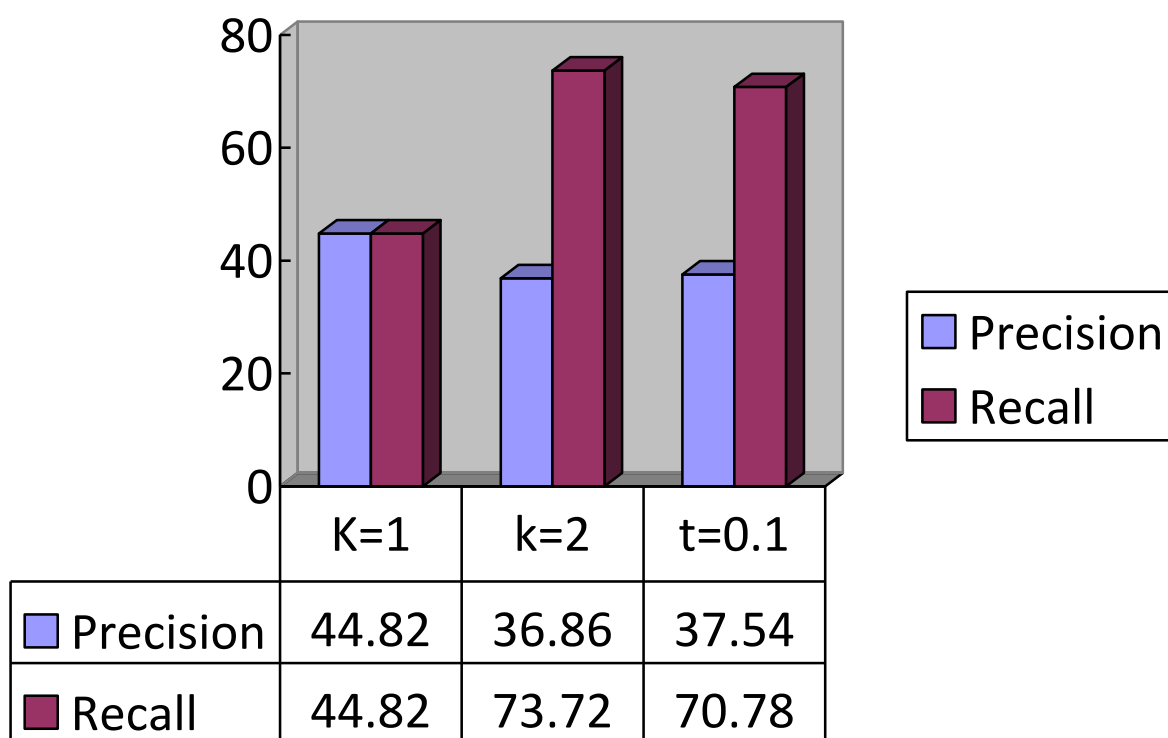
K, number of labels to predict

threshold(t), the threshold for the predicted probability

Test cases

1. k=1, t=0
2. k=2, t=0
3. k=2, t=0.1
4. k=3, does not make sense since we have 3 labels.

The following bar plot graph shows the different precision and recall obtained with the above values.



7. Prediction Accuracy on the test set

## Sentiment Analysis Result

As we can infer from the above graph, the accuracy of the model is very low (45% and 74% (on  $k=2$ )). Since the dataset size was so small (18K), the model is not so precise. The main focus was to identify/predict profane words in the input string. Further process (filtering words) depending on this result.

## Improvements

The very first process, the data collection can be made more specific. Since this data was gathered from a different sources, unrelated data is more resulting in low accuracy.

## Profanity Filter

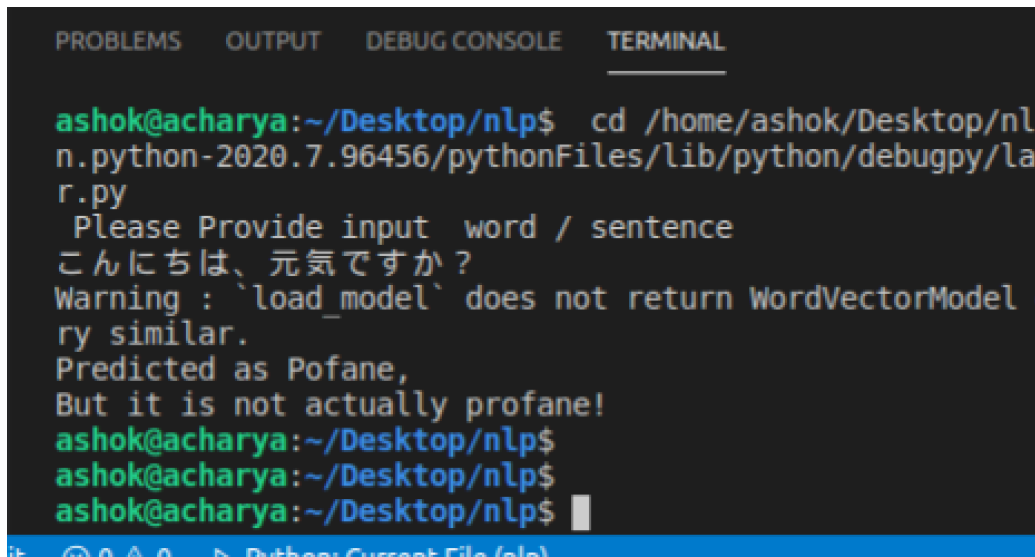
### Background

There is a much robust profanity filter for English or some other language readily and freely available (python's profanity-check, profanity-filter(en,ru)). When it comes to a Japanese text filter, hardly any or heavily priced. There has been significant work done in the other field of NLP in the context of Japanese text. But somehow lacking to advance in these matters. Small efforts to work around the Japanese profanity filter may open up the future possibilities for contributing to the Japanese language.

### Implementation

The implementation idea is to combine the result from the above two processes (POS tagging and sentiment analysis) and filter out the profane words in user input.

### Sample run



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

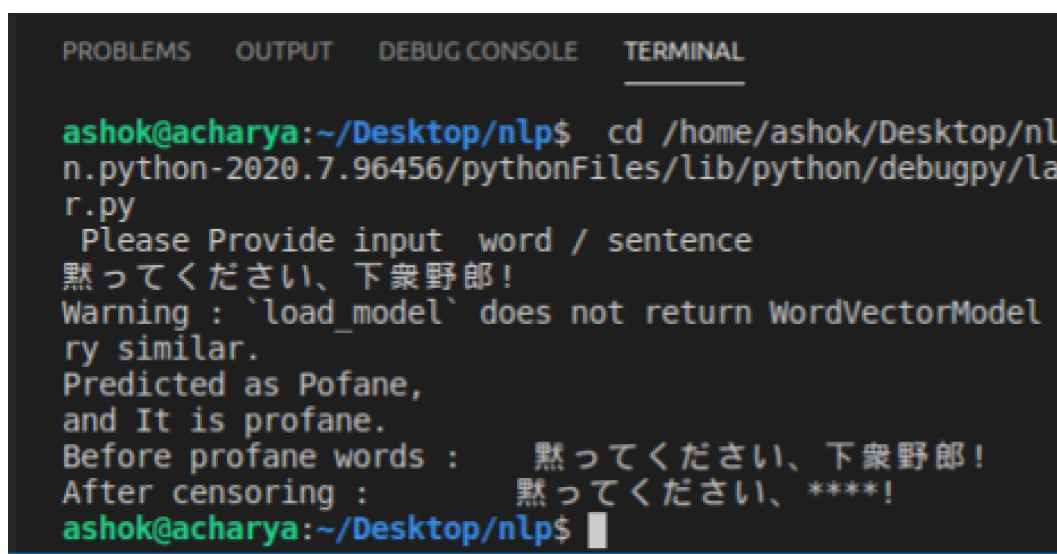
ashok@acharya:~/Desktop/nlp$ cd /home/ashok/Desktop/nlp
n.python-2020.7.96456/pythonFiles/lib/python/debugpy/la
r.py
Please Provide input word / sentence
こんにちは、元気ですか？
Warning : `load_model` does not return WordVectorModel
ry similar.
Predicted as Pofane,
But it is not actually profane!
ashok@acharya:~/Desktop/nlp$
ashok@acharya:~/Desktop/nlp$
ashok@acharya:~/Desktop/nlp$
```

図 8. Profanity filter with sample user input (non-profane)



The Above snapshot is the sample output of the profanity filter. Input is taken from a user as command-line argument or program standard input.

*Filter.py* program first runs the prediction based on the previously generated model. In this specific example, we can it predicted the input “ こんにちは、お元気ですか？” as profane. This is because of the very low accuracy of the trained model. But, we have the dictionary of profane which helped to further match the words with the dictionary. So, this resulted as non-profane output for that input.



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

ashok@acharya:~/Desktop/nlp$ cd /home/ashok/Desktop/nlp
n.python-2020.7.96456/pythonFiles/lib/python/debugpy/launcher.py
Please Provide input word / sentence
黙ってください、下衆野郎！
Warning : `load_model` does not return WordVectorModel
ry similar.
Predicted as Pofane,
and It is profane.
Before profane words :   黙ってください、下衆野郎！
After censoring :       黙ってください、****!
ashok@acharya:~/Desktop/nlp$
```

図 9. Profanity filter with sample user input (profane)

In this example run, profane user input was passed. It is predicted as profane by the model. And also matched in the user dictionary to confirm the result. It resulted in profane word and output by censoring the profane word in the user sentence.

## Result

As a result, a simple working profanity filter based on Japanese text is prepared. With improved machine learning process for sentiment analysis and populating the user dictionary with correct part-of-speech for japanese profane words, Japanese profanity filter can be developed.

## Conclusion

Many profanity filters are based on ML, string matching and complex regular expression exercises. With machine learning and the application of NLP, we can use sentiment analysis for predicting the polarity/ profanity and morphological analyzer for part-of-speech tagging.

Information Extraction like Sentiment analysis is one of the great application of Natural Language Processing, have high usability in today's world where text-based communication has become a part. Search (web and product) is one of the best examples of NLP. Information on the web can be utilized to develop various projects to keep our natural language clean.

## Future Work

There is a lot to be done with this particular project. It is a clear process of improving the existing implementation. Coping with the state-of-art techniques used by the other language enthusiasts may help to uplift the current state. Some of the key points to focus on to improve and further implement are as follows:

- i) proper data collection ideas and implementation,
- ii) precisely tagging the POS with the knowledge of Japanese linguistic and grammar rules,
- iii) Japanese text normalization and clean data preprocessing
- iv) Machine learning with more efficient algorithms
- v) Implementation of robust profanity filter

## References

- [1] MeCab: Yet Another Part-of-Speech and Morphological Analyser
- [2] mecab-ipadic-NEologd: Neologism dictionary for MeCab
- [3] Datasets, <http://monoroch.net/kinshi/>
- [4] data.world, #bad words
- [5] kaggle, <https://www.kaggle.com/datasets>
- [6] FastText, Library for efficient text classification and representation learning
- [7] Web Scraping, webscraper

Link to the repository: <https://github.com/Ashok314/nlp>

-----End-----