Toxic Comment Classification using Naïve Bayes

HW4

Harsha Vardhan, Khurdula.

CS59000 Natural Language Processing

Professor: Jon, Rusert.

Task

Perform and explore ***Naïve Bayes classification***, for a dataset which has comments made by user, to predict if the comment is toxic in nature. The goal is to programmatically implement two methods, *a method to train and return the classifier* and another *method for testing and generating output within a csv file with exact format as a input file*, but with an additional column called '*is_toxic*' that has a Boolean integer flag value i.e. 0 or 1 within it.

Approach

First things first, the given data contains a lot of irrelevant characters, which are not very useful to analyze a comment, and would rather act as a medium which leads to improper fitting of the classifier. So I decided to perform some Pre-Processing with Normalization!

Normalization

Programmatically, normalization in this task is done for the entire dataset by the class called 'Normalization' within the script submitted. The following steps were carried to filter and normalize data:

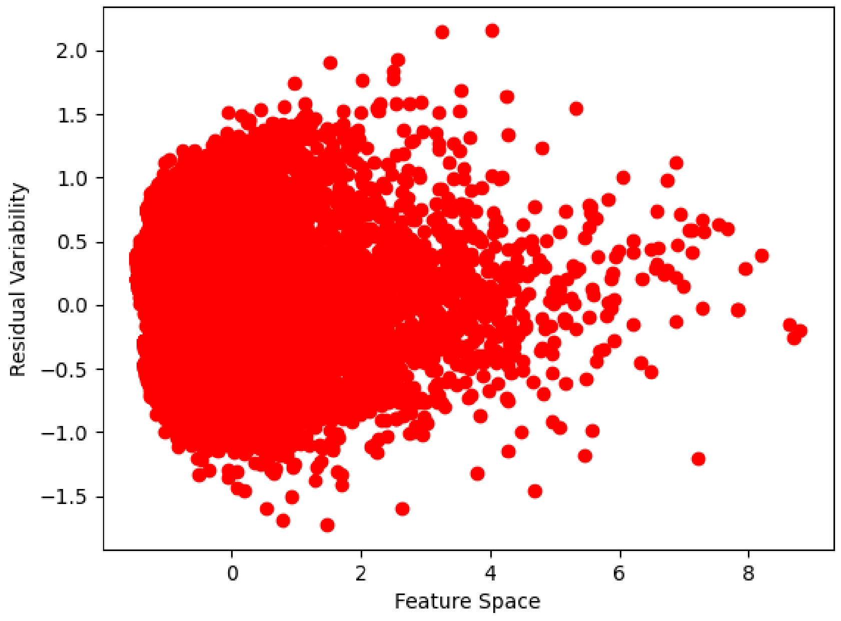
- Irrelevant columns which are not very helpful have been dropped.
- The features which were dropped are: "*id*", "*obscene*", and "severe_toxic", "threat", "insult", "identity_hate".
- Now one might argue that the above features are very useful in order to classify a comment. But after looking at the data a bit longer, I observed that there is a global Boolean feature '*toxic*' that is 1 or true, for any of the dropped columns where their value is 1. Meaning, *toxic* label is always 1, if any of the above dropped columns had 1 instead of 0. And many comments were simply toxic, and not severely toxic, as well as the fact to be remembered is every *severe_toxic* comment is *toxic*, but not all toxic comments are *severe_toxic*.
- All the text has been converted into lower case for case consistency this plays a very important role, while creating probabilities using frequencies by a *CountVectorizer*.
- Only text, has been captured in-order to eliminate special characters.
- And lastly, comments which have no text remaining after performing the above steps have been dropped.

- The ‘***Normalization***’ class also creates the corpus, which is used for training, and testing the classifier.

Understanding the Data using CountVectorizer

After normalization, and before fitting the models was for me to see how far apart the comments are in terms of their toxicity. In-order to achieve this I try to plot a scatter plot for the corpus, instead I get smacked in the face with an error stating, “Failed to allocation 209 gb for plot.” I don’t have such computational resources, the best I can do is run Minecraft with shaders at 80 FPS.

So instead of plotting for the entire *Vectorized sparse matrix*, I reduced the matrix generated by CountVectorizer into 2 component vectors (Two dimensions) i.e. X-axis: “Feature Space” and Y-axis: “Residual Variability”. By plotting this scatter for a sample with just the toxic comments we get the following plot:

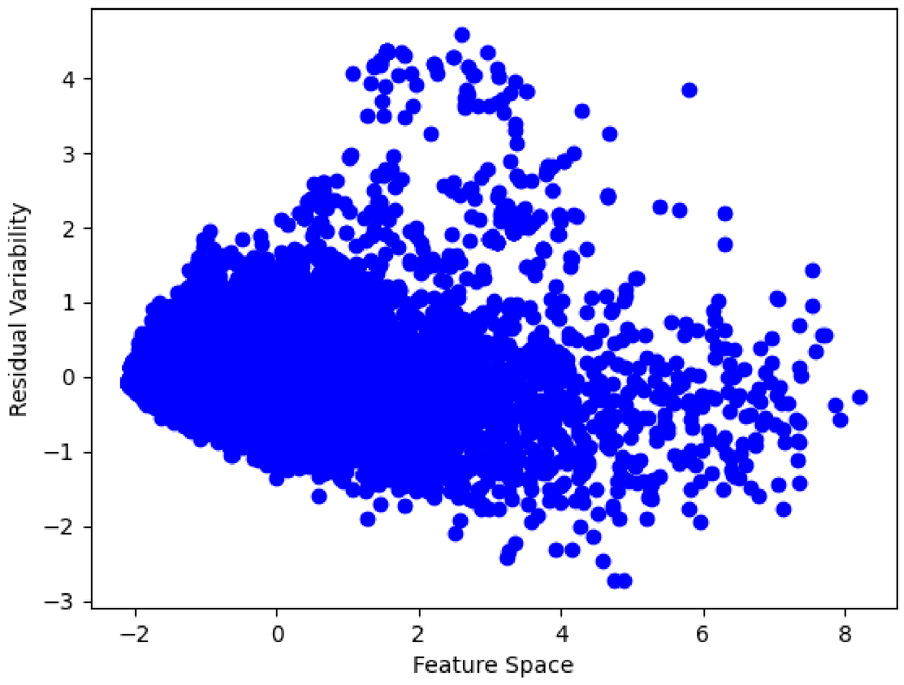


Scatter plot by taking a sample of toxic comments n = 10000.

Note:

The feature space determines the comment's feature count, i.e. total features created from CountVectorizer, and Residual Variability shows how different these features are for each toxic comment.

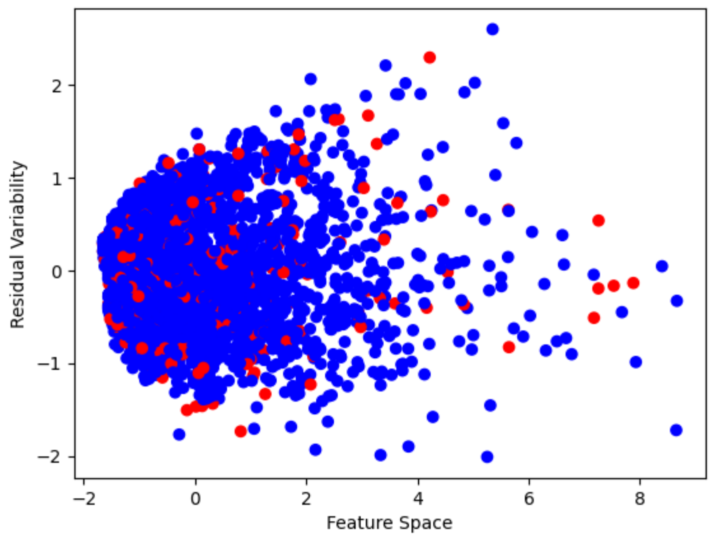
Now that we have a plot for toxic comments, only let's plot a graph for 10000 nontoxic comments:



Scatter plot by taking a sample of non-toxic comments n = 10000

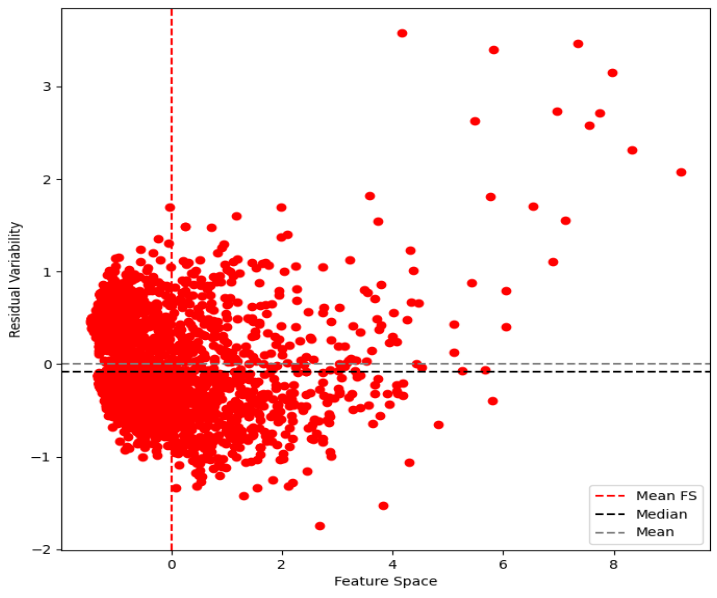
Immediate observations can be made, as we can the *toxic comments* clusters do not have high feature space meaning vectors are a lot more similar, because their vocab is limited mostly by slurs, in-order for the comment to be toxic most of the slur tokens are reused and since these are quite often present in toxic comments, they tend to have lesser or similar vector or feature values of X i.e. feature space! And the nontoxic comments have a much spread Feature Space.

Let's try and plot these both together. We shall project, 2000 toxic comments, over 1000 nontoxic comments. And we get:



Scatter plot by taking a sample of n= 3000, out of which 67% are Toxic.

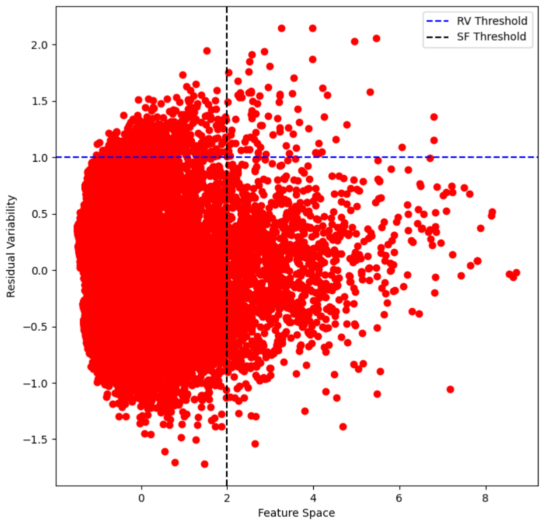
Now instead of using non toxic comments, let understand a how are toxic comments, being plotted or classified over Means of Feature Space and Residual Variability, and Median for it as well.

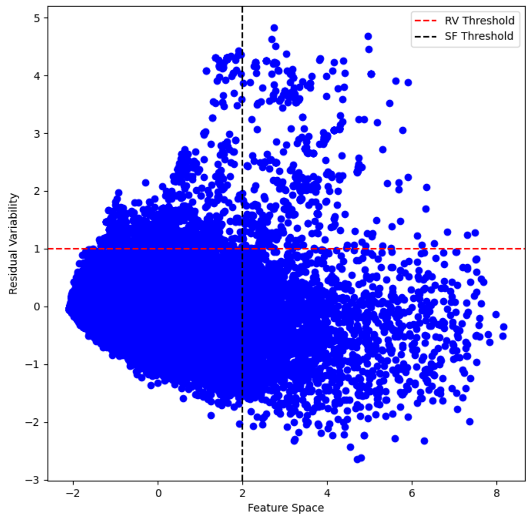


Graph with Mean and Median of Residual Variability and Mean of Feature Space.

Hypothesis: *Most Toxic comments, tend to have Feature Space < 2, and Lesser Residual Variability < 1.*

In-order to check for the above hypothesis lets draw thresholds at the values mentioned above. And see if most of the plots are captured within bottom left section created by intersection of these thresholds. These thresholds are represented by two perpendicularly intersecting lines, called "***RV Threshold****" and "****FS Threshold***".





Toxic Comments n = 20000 ***Non Toxic Comments n = 20000***

We can now confirm our initial hypothesis, most of the Toxic comments have ***Feature Space* < 2 & *Residual Variability* < 1**. *This is more clearly visible if we plot this again with FS Threshold = 0.75.*

Model Training and Persistence

So to begin with, instead of using Gaussian Naïve Bayes, I decided to use Bernoulli Naïve Bayes, as we all already understand the difference between these two i.e. *BernoulliNB* is better suited if the features only contain binary values like 1s or 0s, on the other hand *GaussianNB* is better suited for features with continuous data within them, and additional technical difference is the fact that *GaussianNB* expects a Dense input, whereas *BernoulliNB* expects a Sparse input., I had to just initialize my *Vectorizer ()* with a parameter "*binary = True*".

Initially, the idea was to use *TF-IDF* vectors in order to create the word embeddings, but since this assignment restricts us to use *CountVectorizer*, I created the word embeddings for this model using CountVectorizer, which generates a boolean feature embeddings, it looks for each feature's presence rather than using frequencies. This is done in order to provide these word embeddings as input to the *BernoulliNB* that we have as our main model. As a Reference Model I decided to train *GaussianNB* as well which uses the same Vectorizer to create the word embeddings, by using frequencies rather than boolean values.

A total of 155k+ Comments have been used to create these embeddings, and our Model upon fitting and testing yields an Accuracy of 0.940, whereas the *GuassianNB model* yields an Accuracy score of 0.791. Which is quite a significant margin of difference, keeping in mind the fact, I provided continuous data embeddings as input to *GaussianNB* model rather than embeddings used for *BernoulliNB* model.

Well there's a more of debugging that's to come, now that I had achieved a decent accuracy score. I ran my tests manually on the *BernoulliNB* model. I realized, I had trouble with Precision! (The proportion of true positives, to all positives predicted + false positives.). Here are some of the model metrics with default thresholds before some real analysis:

Note: The following scores are for **underfitted** and **overfitted** iterations of both the models. In many of the training iterations, both the models had common false positives all the time! Which can be observed with moderate to good ***precision***, ***recall*** but poor ***accuracy*** scores!

For instance, the phrase "***what the hell***?"

Is classified as *toxic*, by both models under Iteration I, II, & III, which clearly indicates there are a lot of false positives, hence the precision and recall scores seem inconsistent with the accuracies of these models.

Training Iteration I:

Model	Accuracy	Precision	Recall	F1-Score
Bernoulli	0.7085	0.6528	0.8927	0.7543
Gaussian	0.6743	0.6438	0.7147	0.8031

Table: Model metrics for trainset with toxic comments = 5000 & Nontoxic comments = 5000.

Training Iteration II:

Model	Accuracy	Precision	Recall	F1-Score
Bernoulli	0.7312	0.7138	0.9072	0.7989
Gaussian	0.7042	0.7054	0.8595	0.7749

Table: Model metrics for trainset with toxic comments = 7500 & Nontoxic comments = 5000.

Training Iteration III:

Model	Accuracy	Precision	Recall	F1-Score
Bernoulli	0.7133	0.6570	0.8900	0.7560
Gaussian	0.6722	0.6257	0.8291	0.7132

Table: Model metrics for trainset with toxic comments = 7500 & Nontoxic comments = 7500.

Training Iteration IV:

Model	Accuracy	Precision	Recall	F1-Score
Bernoulli	0.6984	0.6463	0.8787	0.7448
Gaussian	0.6630	0.6243	0.8164	0.7076

Table: Model metrics for trainset with toxic comments = 10000 & Nontoxic comments = 10000

Observation: There is a huge shift in metrics, when we manipulate the total number of Nontoxic comments within the training set. Keeping this in regard, let's perform another iteration of fitting with all possible toxic data, and only 75% of the entire nontoxic comments.

Training Iteration V:

Model	Accuracy	Precision	Recall	F1-Score
Bernoulli	0.8943	0.5326	0.7082	0.6080
Gaussian	0.7295	0.5065	0.6982	0.5723

Table: Model metrics for trainset with toxic comments = ALL & Nontoxic comments = 115416

Observation:

Iteration V does a decent job, in terms of model accuracy, but however, the other metrics can be considered as terrible for such a small data set with corpora being less than 200k, however the *GuassianNB* model fails to fit a lot more times, than *BernoulliNB* due to lack of computational resources. When we try and fit entire data, for both models *BernoulliNB* does not face any issues, on the other hand since we are to calculate the frequencies unlike Boolean feature embeddings used in *BernoulliNB*, the *GaussianNB* does not train! And throws an exception asking for more Main Memory, to be more precise here is the exception thrown when we try and train *GaussianNB*: Unable to allocate 158. G6iB for an array with shape (111669, 189775)

All of the iterations above, where just carried out to understand which model does a better job in classification of comments as toxic! And in my opinion I have my champion! The ***BernoulliNB***.

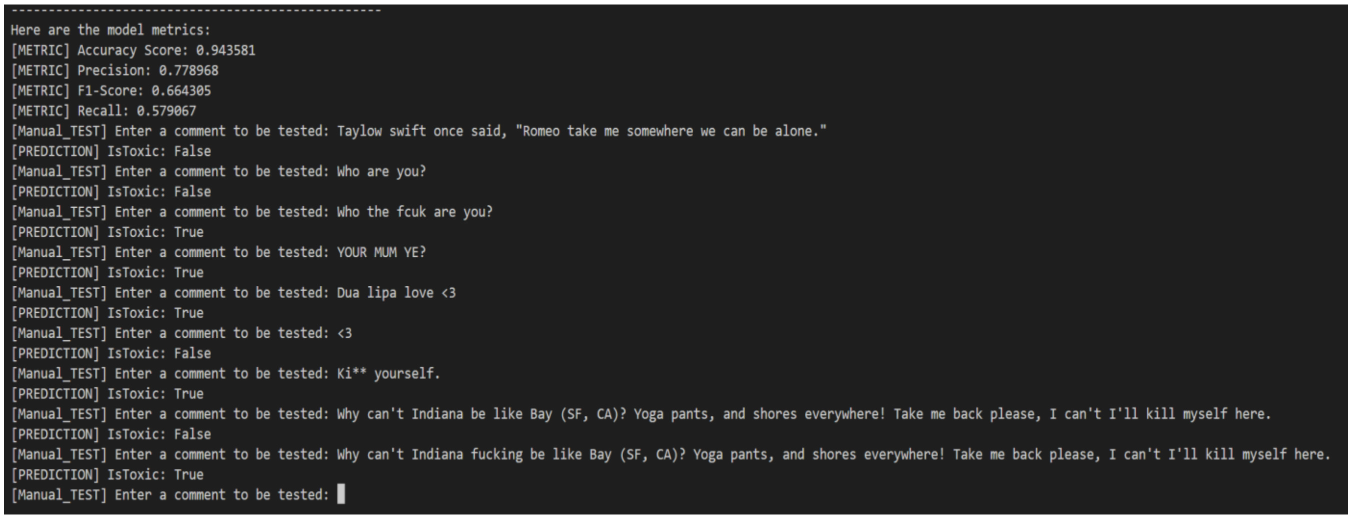
The advantage of using this model is the fact that we create embeddings based on their presence within a comment, rather than having to count frequencies and generate dense matrices which is very computationally expensive in terms of Memory. This model is rather simpler and yields better results for current task at hand. So moving on I have disregarded the usage of *GaussianNB*. So the fine-tuned version of *BernoulliNB* trained on 60% entire data, with default threshold for classification yields the following scores:

Fine-tuned *BernoulliNB*:

Model	Accuracy	Precision	Recall	F1-Score
Bernoulli	0.9435	0.7789	0.5790	0.6643

Table: Model metrics for training done on entire trainset.

This version of the model performs way better than any iterations so far. It is able to classify certain phrases really well, some of which I had expected it to fail at:



```
--------------------------------------------------
Here are the model metrics:
[METRIC] Accuracy Score: 0.943581
[METRIC] Precision: 0.778968
[METRIC] F1-Score: 0.664305
[METRIC] Recall: 0.579067
[Manual_TEST] Enter a comment to be tested: Taylow swift once said, "Romeo take me somewhere we can be alone."
[PREDICTION] IsToxic: False
[Manual_TEST] Enter a comment to be tested: Who are you?
[PREDICTION] IsToxic: False
[Manual_TEST] Enter a comment to be tested: Who the fcuk are you?
[PREDICTION] IsToxic: True
[Manual_TEST] Enter a comment to be tested: YOUR MUM YE?
[PREDICTION] IsToxic: True
[Manual_TEST] Enter a comment to be tested: Dua lipa love <3
[PREDICTION] IsToxic: True
[Manual_TEST] Enter a comment to be tested: <3
[PREDICTION] IsToxic: False
[Manual_TEST] Enter a comment to be tested: Ki** yourself.
[PREDICTION] IsToxic: True
[Manual_TEST] Enter a comment to be tested: Why can't Indiana be like Bay (SF, CA)? Yoga pants, and shores everywhere! Take me back please, I can't I'll kill myself here.
[PREDICTION] IsToxic: False
[Manual_TEST] Enter a comment to be tested: Why can't Indiana fucking be like Bay (SF, CA)? Yoga pants, and shores everywhere! Take me back please, I can't I'll kill myself here.
[PREDICTION] IsToxic: True
[Manual_TEST] Enter a comment to be tested: █
```

Image: Sample predictions for Fine-tuned Model.

This version does a better job classifying positives, i.e. it classifies toxic comments very well and also has higher false positives, Hence the higher Precision, Accuracy but lower Recall score.

Note:

The file that contains expected outputs is _output.csv, which has the required data.

Inorder to run this script please run the following commands:

“>>>from ToxicCommentClassifier import BernoulliDistribution”

“>>>obj = BernoulliDistribution()”

“>>>trained_model = obj.train_NB_model(“./datasets/train.csv”)”

“>>>obj.test_NB_model(“./datasets/test.csv”, trained_model)”

This will produce the necessary outputs.

References:

Khurdula, H. V. (2023, February 21). *Naive-Bayes-ToxicComment-Clasification*. GitHub. Retrieved March 4, 2023, from https://github.com/Khurdhula-Harshavardhan/Naive-Bayes-ToxicComment-Clasification.git

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). *Scikit-learn: Machine learning in Python. Journal of Machine Learning Research*, 12, 2825-2830.