**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**

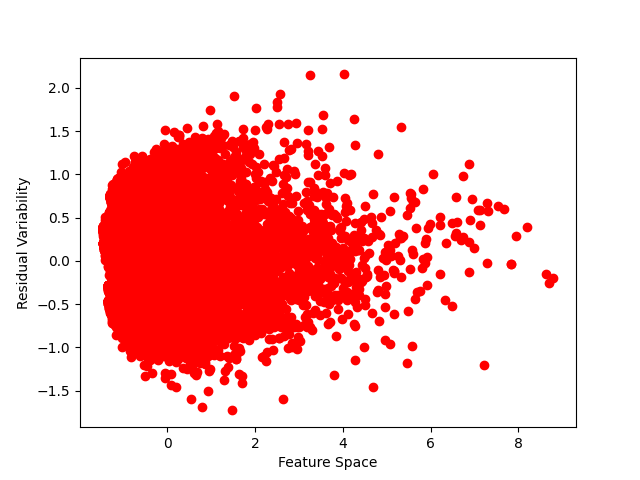
Programmatically, normalization in this task is done for the entire dataset by the class called ‘Normalization’ within the script submitted. The following steps were carried to filter and normalize data:

* 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**

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:

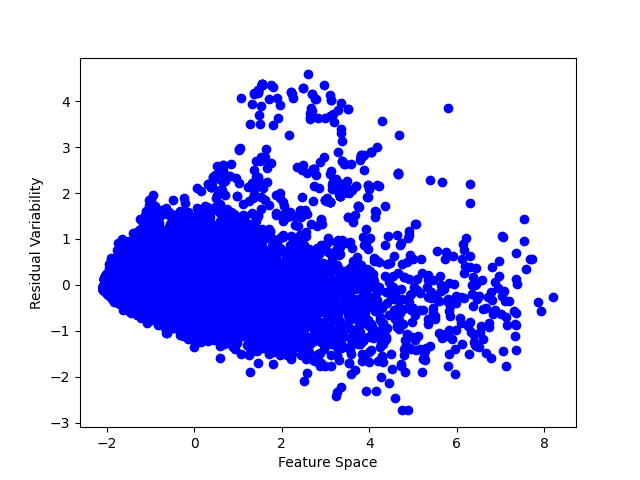


***Scatter plot by taking a sample of toxic comments n = 10000.***

**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.

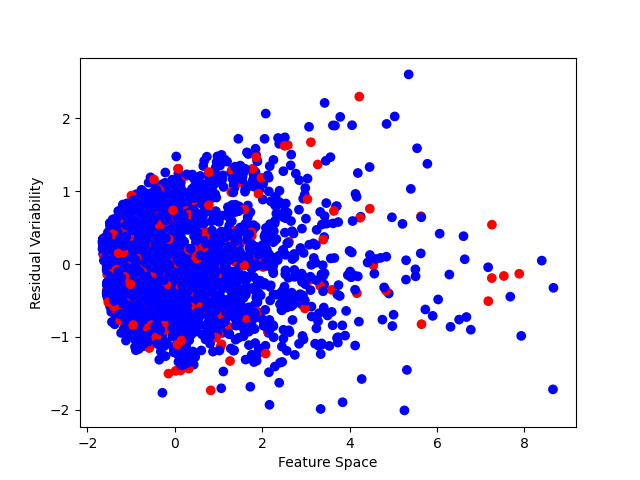
Now that we have a plot for toxic comments, only let’s plot a graph for 10000 nontoxic comments, we get:



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

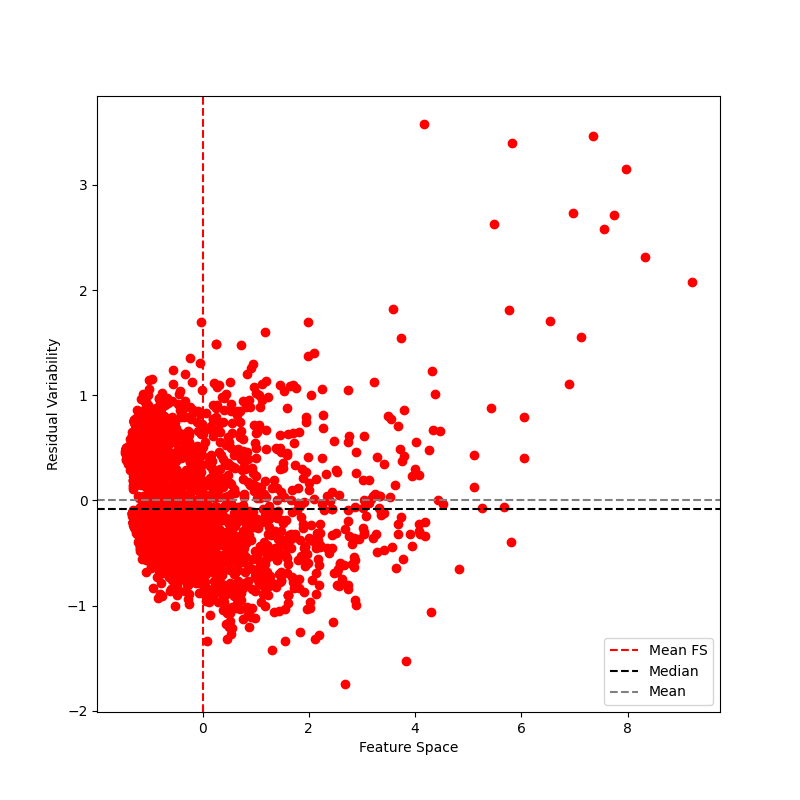
Immediate observations can be made, as we can the *toxic comments* clusters do not have high feature space, because their vocab is limited mostly by slurs, meaning in-order for the comment to be toxic most of the slur tokens are reused and since these are quite often present in toxic comments, they tend to have lesser 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.***

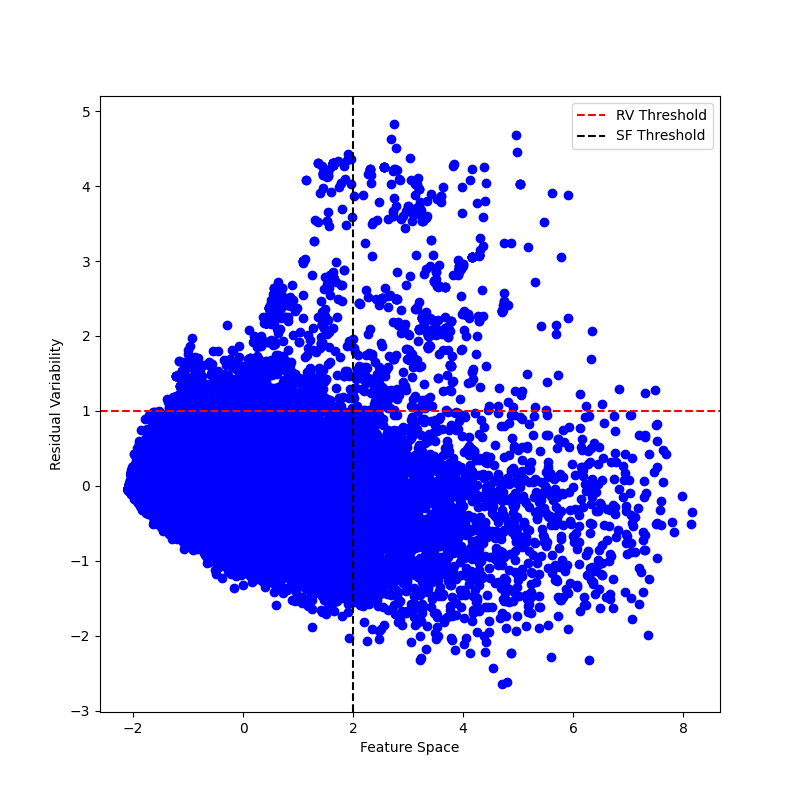
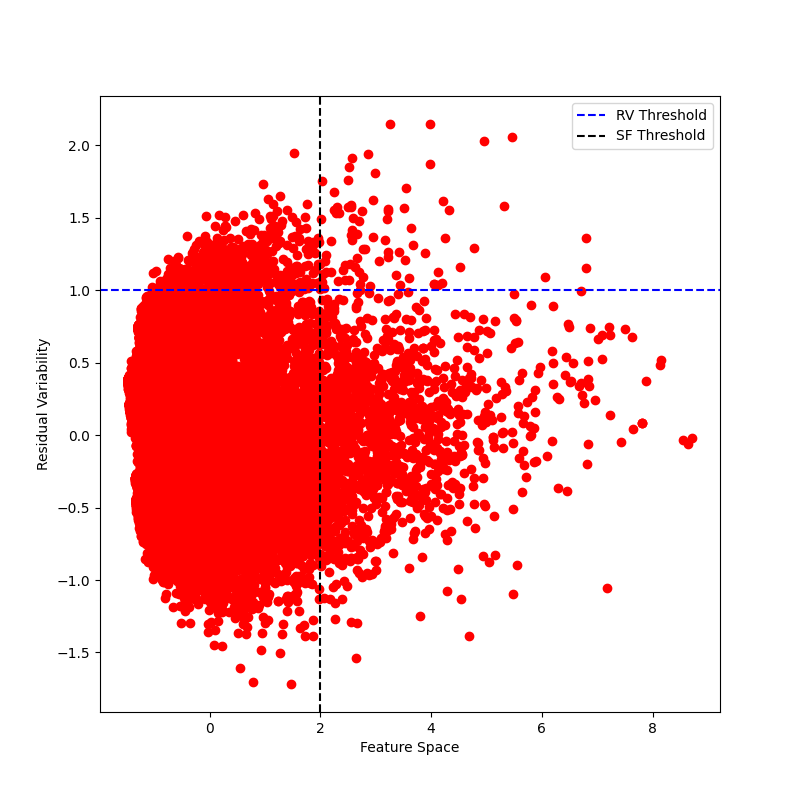
Now instead of using non toxic comments, let understand a how are toxic comments, being plotted or classified over Means of Feature Space and Residual Variability, and Median for it as well.



***Graph with Mean and Median of Residual Variability and Mean of Feature Scace.***

**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 large margin of difference, keep in mind I provided continuous data embeddings as input to *GaussianNB* model rather than embeddings used for *BernoulliNB* model.