Automatic Personal Email Organizer

Saransh Goel	2019A7PS0988P
Aurindom Bhattacharyya	2018A2PS0082P
Dev Anand Gupta	2019A4PS0252P

Submitted to

Dr. Navneet Goyal

Department of Computer Science & Information Systems

For the partial fulfillment of the course

BITS F464 - Machine Learning

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

OBJECTIVE

To classify the emails as spam and ham using N-gram model and bayesian approach and checking the model by applying different types of classification functions like discriminant function , support vector machine, logistic regression.

BUILDING A N-GRAM LANGUAGE MODEL

N-gram Language model

N-gram is a sequence of N tokens or words. For a sentence "*Birla Institute of Technology and Science was incepted as an Institute with Dr. G.D. Birla as Founder Chairman.*"

- **Unigram** => "*Birla", "Institute", "of", "Technology", "and", "Science", "was", "incepted", "as", "an", "Institute", "with", "Dr.", "G.D",. "Birla", "as", "Founder", "Chairman".*
- **Bigram** => *"Birla Institute", "Institute of", "of Technology" or "technology and".*
- **Trigram** => *"Birla Institute of", "Institute of Technology", "of Technology and" or "Technology and Science".*

How the N-gram model works?

N-gram model works on the principle of probability of finding of a word based on the previous words in the sentence. In above example the probability of occurance of *"incepted"* depends on the probability of occurrence of *"Birla Institute of Technology and Science was"*.

Chain rule of probability

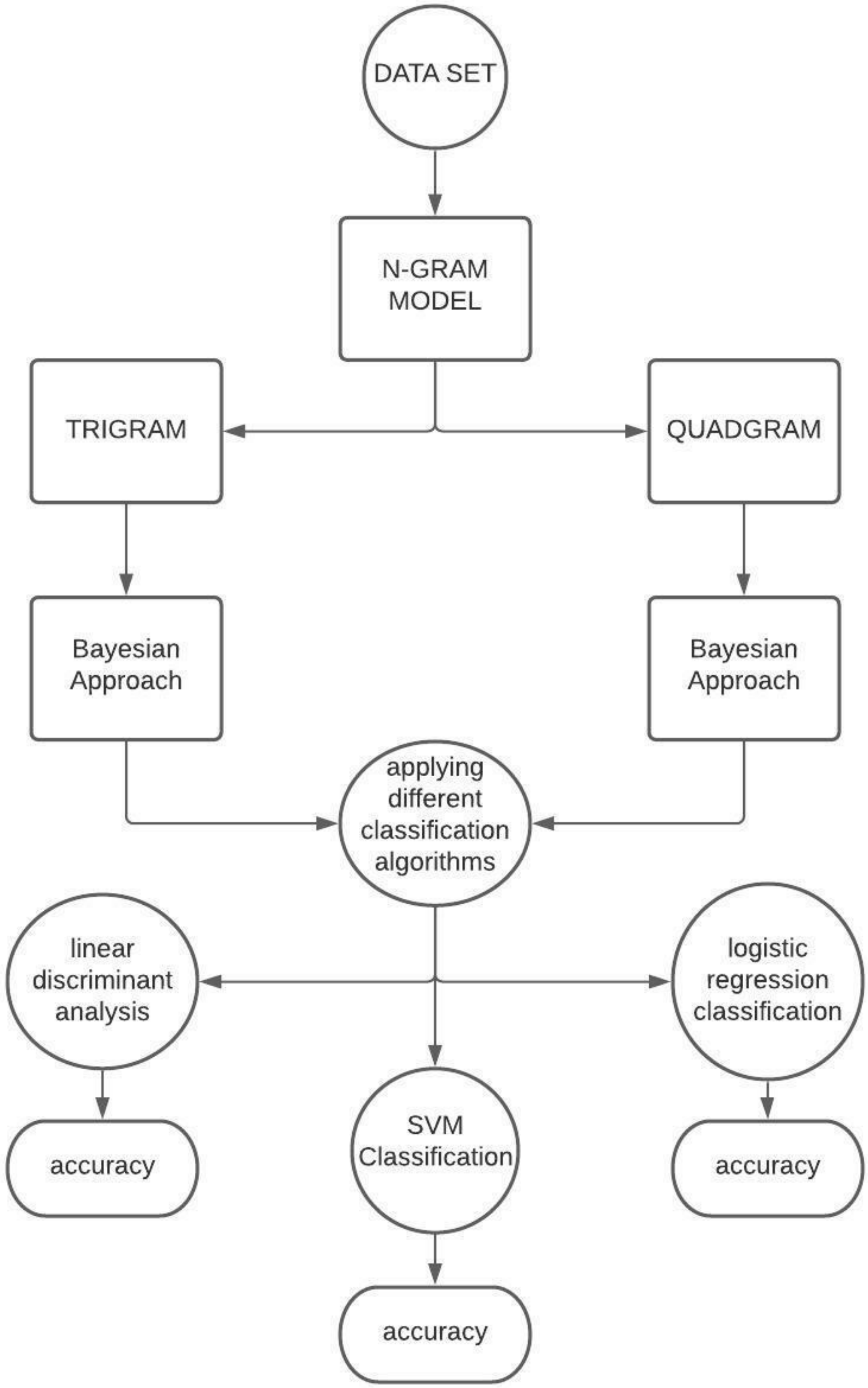
P(w1…...wn) = P(w1) . P(w2|w1) . P(w3|w1 w2) . P(w4|w1 w2 w3) ……….P(wn|w1 w2…...wn-1)

One problem arise is of cost of computation, In above chain rule, the cost of computation is not feasible for a big data. So in order to solve this problem we will use an assumption called Markov assumption which will reduce computation cost to a great extend.

Simplification assumption(Markov assumption)

P(wk | w1….wk-1) = P(wk | wk-1)

PROCESS FLOW



STEP-1:
TRAINING OF THE MODEL

1. **Model of all emails(for calculating the probability of evidence)**

In [141]:
```
import re
import nltk
from nltk import bigrams, trigrams
from collections import Counter, defaultdict

# Create a placeholder for model
model = defaultdict(lambda: defaultdict(lambda: 0))

# Count frequency of co-occurance
for a in range(0,5728):
    parahgraph = data.loc(0)[a]['text']
    sentences = re.split(", | . | : | ;",parahgraph)
    numSentences = len(sentences)
    for j in range(0,numSentences):
        array = sentences[j].split(" ")
        length = len(array)
        extra = []
        for k in range(0,length):
            if len(array[k])==0:
                extra.append(k)
        count = 0
        for n in extra:
            array.pop(n-count)
            count += 1
            length -= 1
        if(length >= 3):
            for i in range(0,length-2):
                model[(array[i],array[i+1])][array[i+2]] += 1
```

```
# Create a placeholder for quadgram_model
model_quadgram = defaultdict(lambda: defaultdict(lambda: 0))

# Count frequency of co-occurance
for a in range(0,5728):
    parahgraph = data.loc(0)[a]['text']
    sentences = re.split(", | . | : | ;",parahgraph)
    numSentences = len(sentences)
    for j in range(0,numSentences):
        array = sentences[j].split(" ")
        length = len(array)
        extra = []
        for k in range(0,length):
            if len(array[k])==0:
                extra.append(k)
        count = 0
        for n in extra:
            array.pop(n-count)
            count += 1
            length -= 1
        if(length >= 4):
            for i in range(0,length-3):
                model_quadgram[(array[i],array[i+1],array[i+2])][array[i+3]] += 1
```

2. Model of spam emails(for calculating the probability of likelihood)

```python
# Create a placeholder for spam model
model_spam = defaultdict(lambda: defaultdict(lambda: 0))

# Count frequency of co-occurance
for a in range(0,5728):
    m = data.loc(0)[a]['spam']
    if m == 1:
        parahgraph = data.loc(0)[a]['text']
        sentences = re.split(", | . | : | ;",parahgraph)
        numSentences = len(sentences)
        for j in range(0,numSentences):
            array = sentences[j].split(" ")
            length = len(array)
            extra = []
            for k in range(0,length):
                if len(array[k])==0:
                    extra.append(k)
            count = 0
            for n in extra:
                array.pop(n-count)
                count += 1
                length -= 1
            if(length >= 3):
                for i in range(0,length-2):
                    model_spam[(array[i],array[i+1])][array[i+2]] += 1
```

```python
# Create a placeholder for quadgram_model
model_spam_quadgram = defaultdict(lambda: defaultdict(lambda: 0))

# Count frequency of co-occurance
for a in range(0,5728):
    m = data.loc(0)[a]['spam']
    if m == 1:
        parahgraph = data.loc(0)[a]['text']
        sentences = re.split(", | . | : | ;",parahgraph)
        numSentences = len(sentences)
        for j in range(0,numSentences):
            array = sentences[j].split(" ")
            length = len(array)
            extra = []
            for k in range(0,length):
                if len(array[k])==0:
                    extra.append(k)
            count = 0
            for n in extra:
                array.pop(n-count)
                count += 1
                length -= 1
            if(length >= 4):
                for i in range(0,length-3):
                    model_spam_quadgram[(array[i],array[i+1],array[i+2])][array[i+3]] += 1
```

3. Prior probability for spam mails

In [13]:
```python
numSpam = 0
for a in range(0,5728):
    value = data.loc(0)[a]['spam']
    if value == 1:
        numSpam = numSpam + 1
```

In [14]:
```python
numSpam
```

Out[14]: 1368

In [15]:
```python
prior_spam = numSpam/len(data)
```

In [16]:
```python
prior_spam
```

Out[16]: 0.2388268156424581

STEP-2

Calculating posterior probabilities of emails which will quantify the N-gram model so that we can apply classification algorithms over it.

Bayesian approach

$$P(A|X=x1,x2,x3\ldots..xn) = \frac{P(X=x1,x2,x3\ldots..xn \mid A) * P(A)}{P(X=x1,x2,x3\ldots..xn)}$$

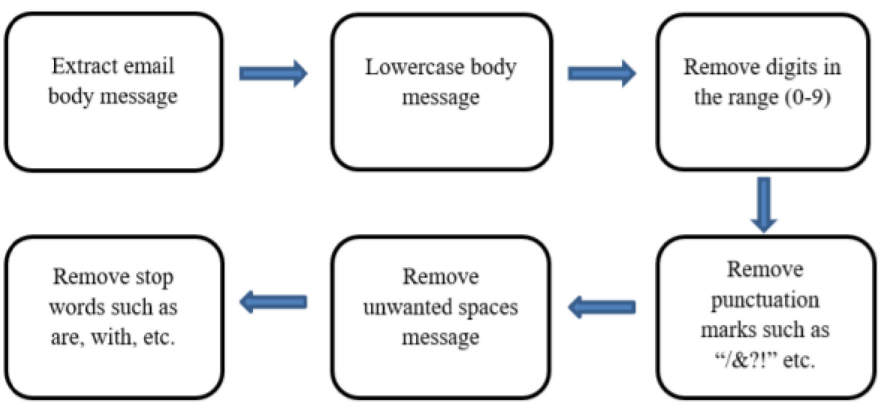
$$\text{Posterior} = \frac{likelihood * prior}{evidence}$$

Likelihood = calculated from trigram model for all the spam mails.
Evidence = calculated from trigram model for all mails including spam and ham.
Prior = calculated from probability of spam or ham in the training data.

Laplace smoothing

The trigram not present in trained model but present in test data will create problem just like in naive bayesian classification. The method to solve this problem is by adding extra count for missing data in the model.

Email preprocessing



1. **Likelihood function**

```
In [147]: import numpy as np
          def likelihood(text, model):
              p = 0
              num = 0
              size = len(model)
              array = text.split(" ")
              length = len(array)
              extra = []
              for k in range(0,length):
                  if len(array[k])==0:
                      extra.append(k)
              count = 0
              for n in extra:
                  array.pop(n-count)
                  count += 1
                  length -= 1
              if(length >= 3):
                  for i in range(0,length-2):
                      if model[(array[i],array[i+1])][array[i+2]] == 0:
                          num += 1
                  for j in range(0,length-2):
                      if model[(array[j],array[j+1])][array[j+2]] == 0:
                          p += np.log(1/(size+num))
                      else:
                          p += np.log(model[(array[j],array[j+1])][array[j+2]]/(size+num))
              return p
```

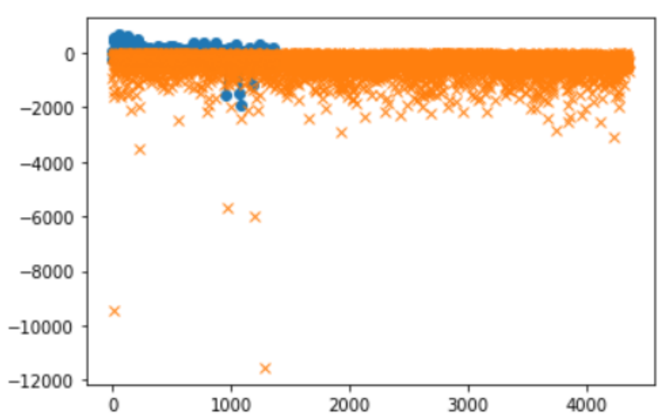
2. **Final data containing the values given by bayesian formula**

```
for a in range(0,5728):
    parahgraph = data.loc(0)[a]['text']
    p = (likelihood(parahgraph,model_spam)+np.log(prior_spam))-likelihood(parahgraph,model)
    label = data.loc(0)[a]['spam']
    ele = [parahgraph, label, p]
    new_data.append(ele)
```

3. **Creating final data file of probability value in log form(to consider value small values of probability)**

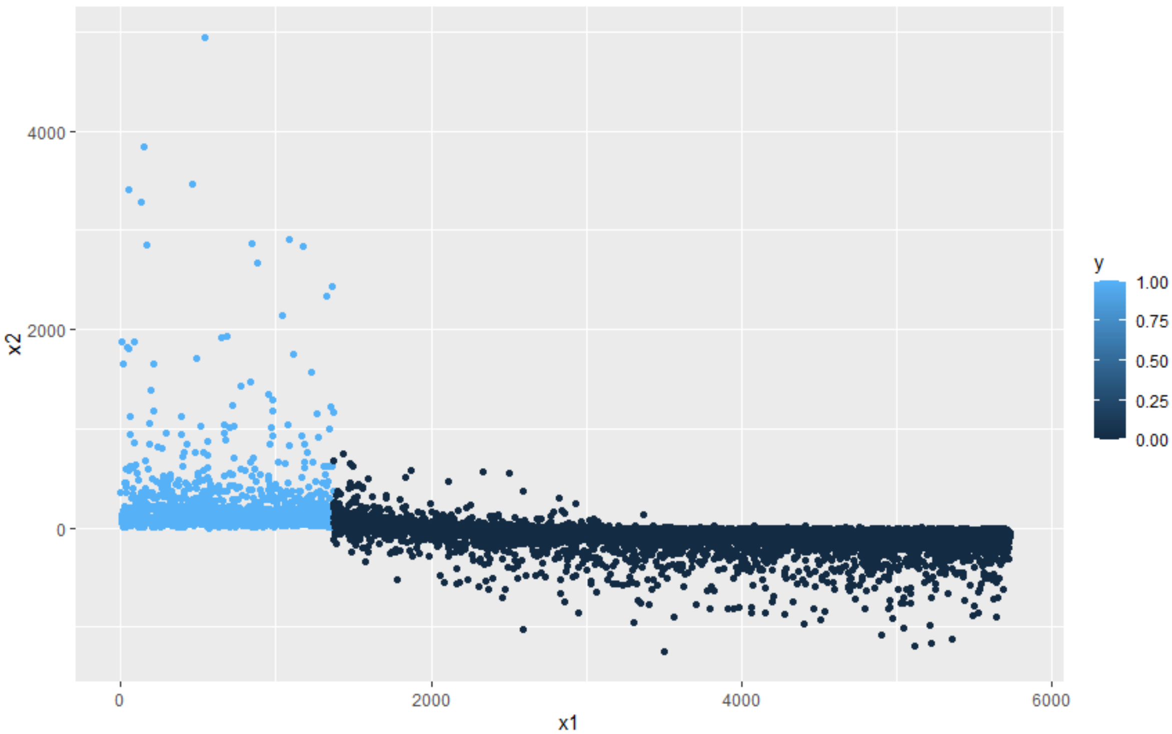
```
In [152]: df = pd.DataFrame(new_data)

In [153]: df.to_csv('processed_email.csv', index=False, na_rep='Unknown')
```

BIGRAM IS NOT INCLUDED IN FURTHER ANALYSIS BECAUSE IT IS VERY UNDERFITTING AS SHOWN BELOW

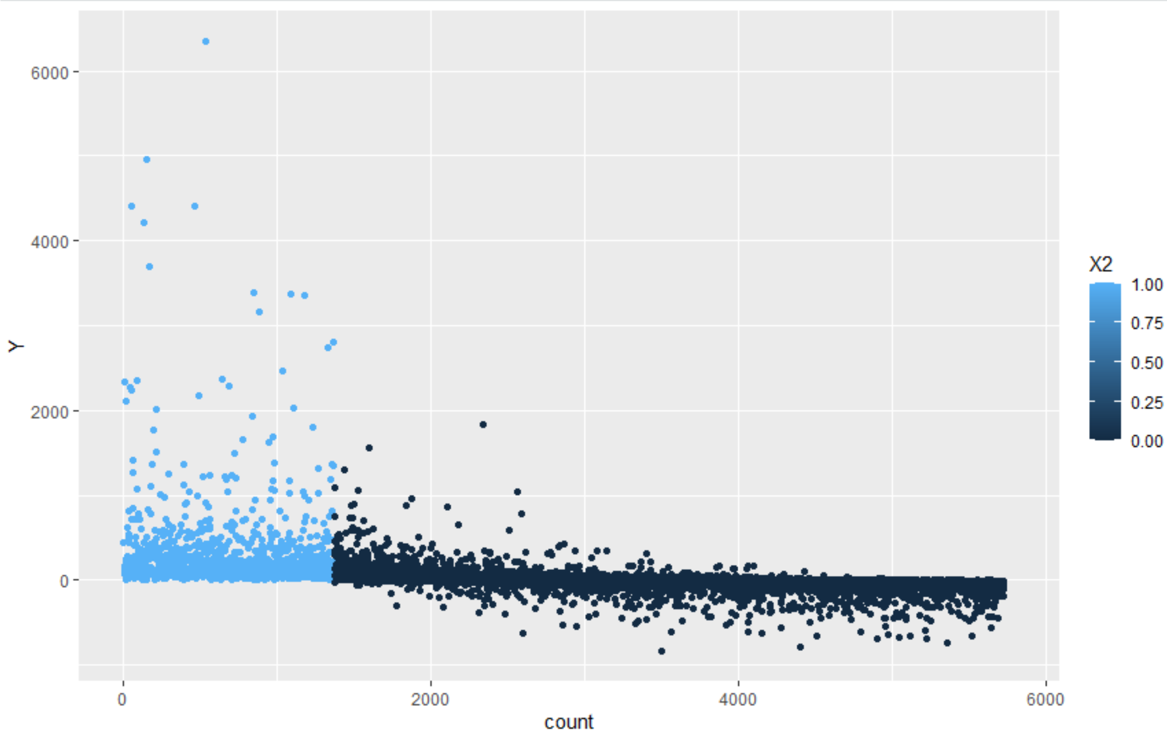


ORIGINAL LABELS GIVEN FOR TRAINING DATA PLOTTED AGAINST VALUES OF TRIGRAM MODEL AND QUADGRAM PROBABILITY VALUES.

TRIGRAM



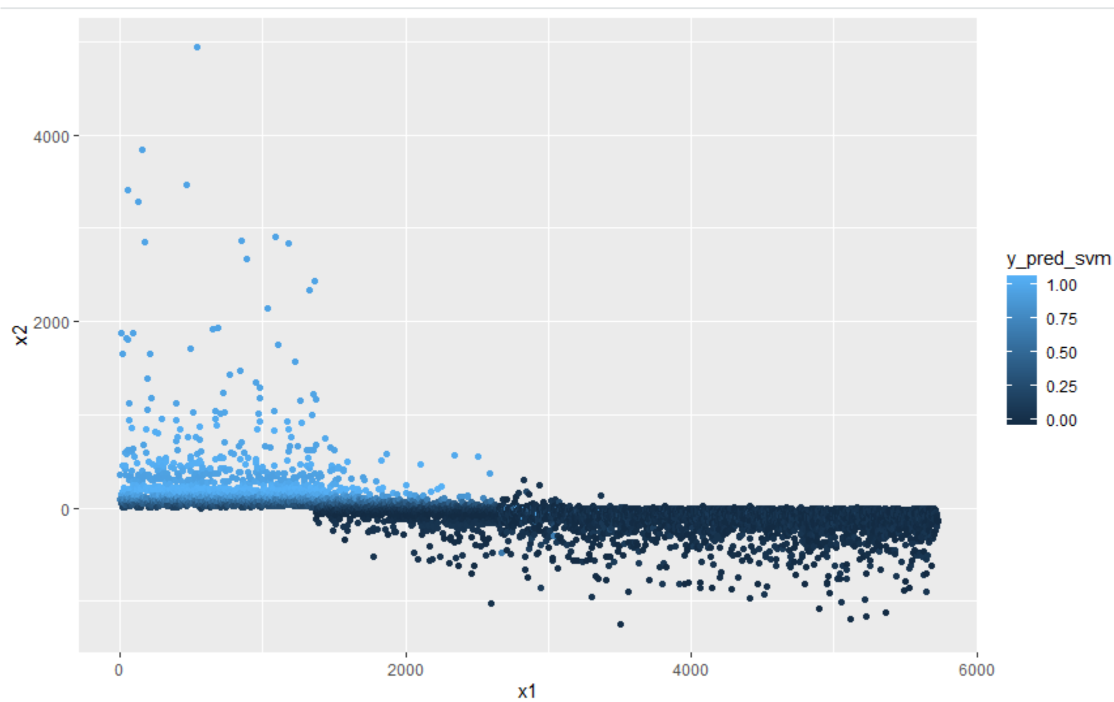
QUADGRAM



STEP-3

Applying various classification algorithms over the data set of trigram and quadgram both.

1. **After applying SVM**
   a. **TRIGRAM**

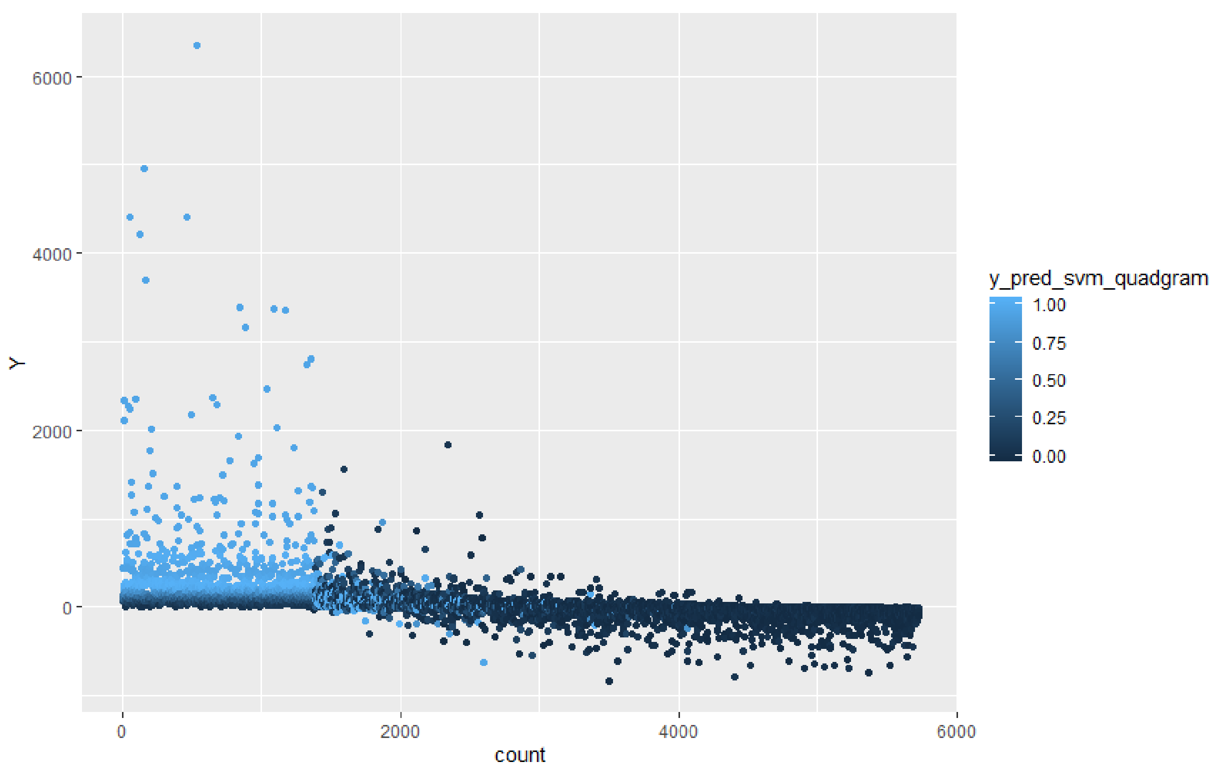


For Accuracy of model by using svm

```
Call:
svm(formula = X2 ~ Y, data = processed_email)


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  radial
       cost:  1
      gamma:  1
    epsilon:  0.1


Number of Support Vectors:  2833
```

Accuracy = 2833/5726

b. QUADGRAM

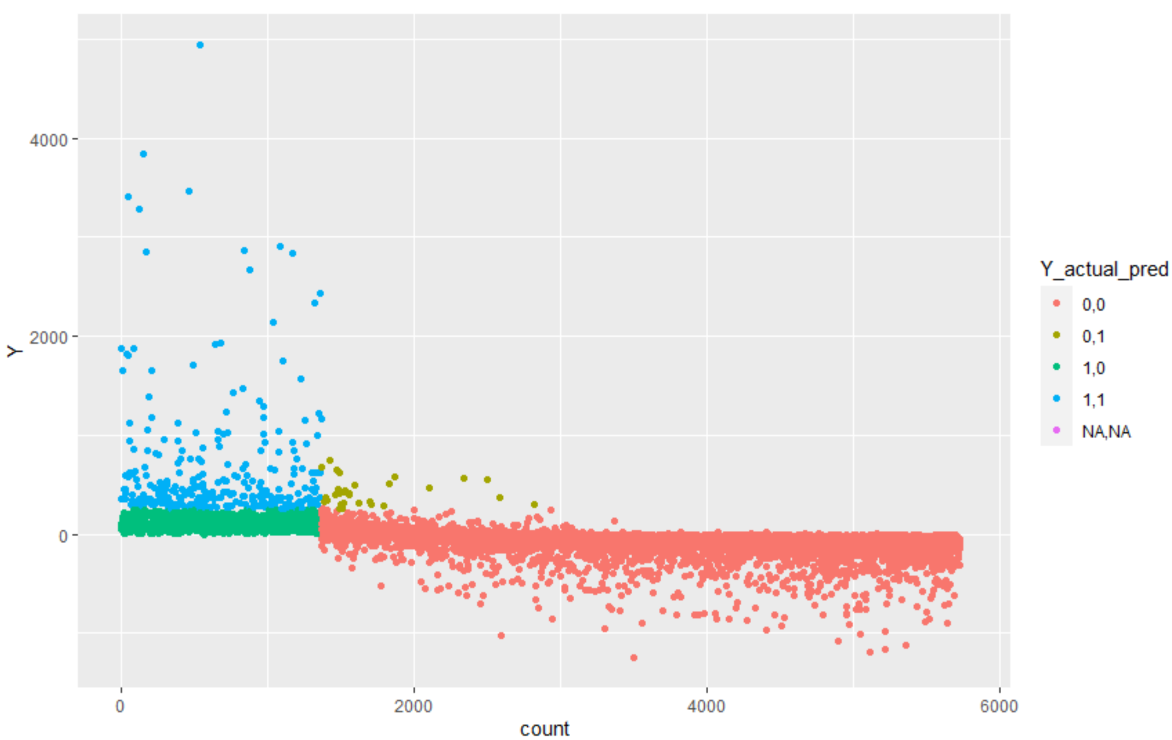


For Accuracy of model by using svm

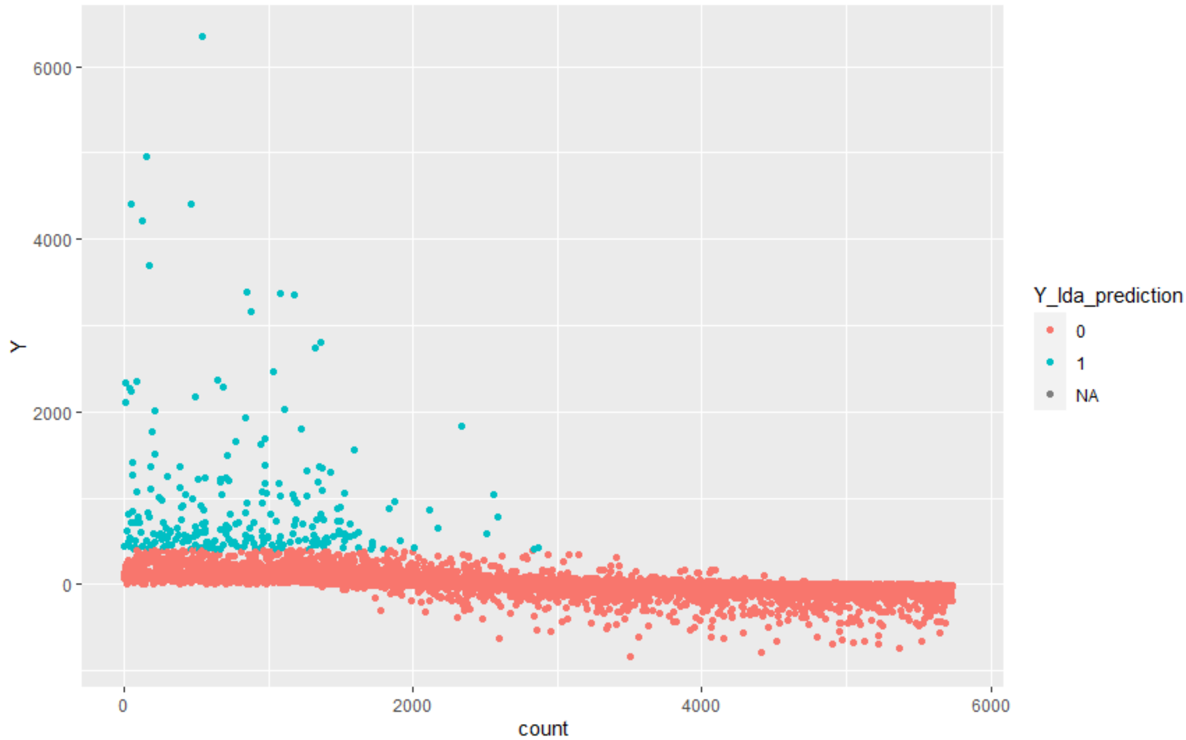
```
Call:
svm(formula = X2 ~ Y, data = processed_email_quadgram)


Parameters:
   SVM-Type:  eps-regression
 SVM-Kernel:  radial
       cost:  1
      gamma:  1
    epsilon:  0.1


Number of Support Vectors:  2473
```

Accuracy = 2833/5726

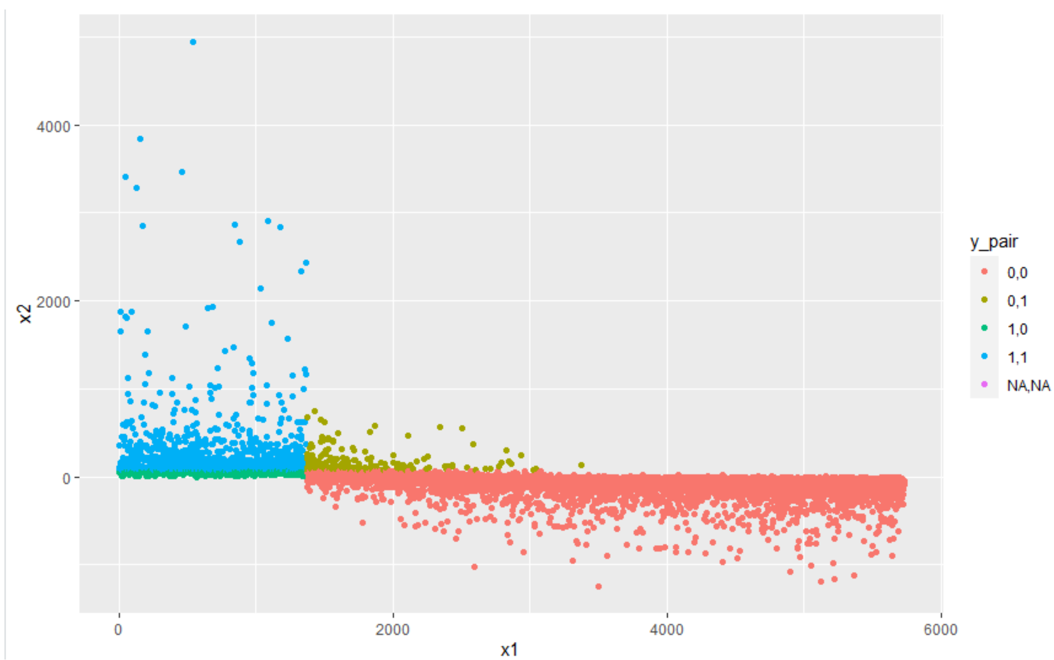
2. **After applying LDA**
    a. **TRIGRAM**



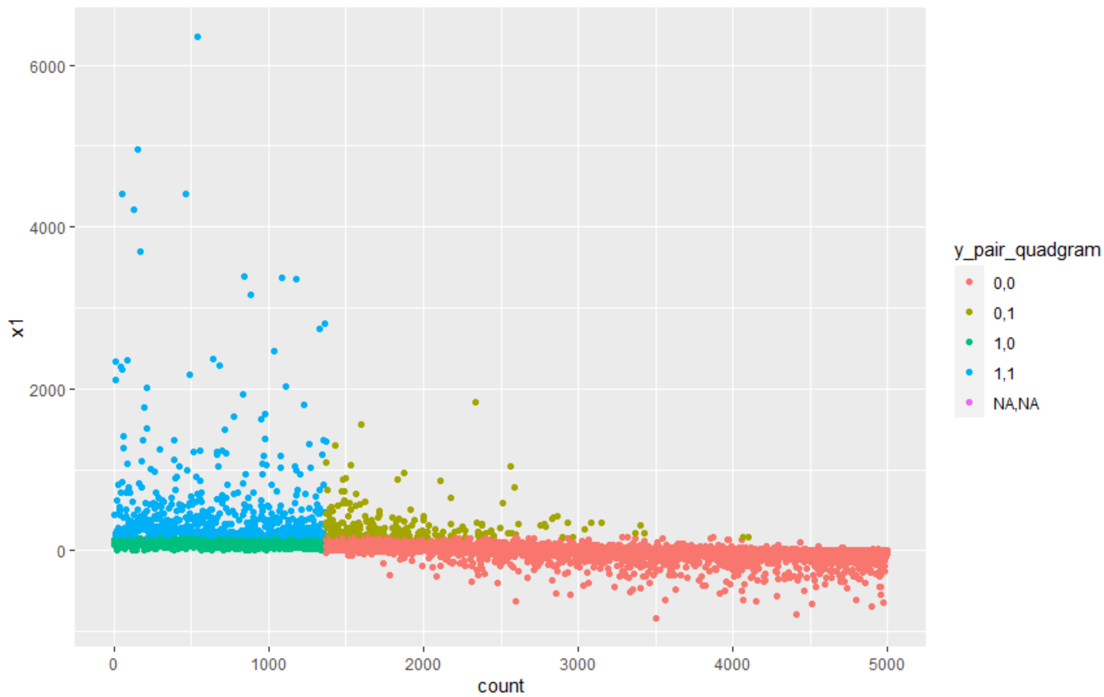
b. **QUADGRAM**



3. **After applying logistic regression**
TRIGRAM



QUADGRAM



Reference
https://www.analyticsvidhya.com/blog/2021/04/improve-naive-bayes-text-classifier-using-laplace-smoothing/
Language Model In NLP | Build Language Model in Python
https://towardsdatascience.com/email-spam-detection-1-2-b0e06a5c0472