Using Modified Term Frequency to Improve Term Weighting for Text Classification

National Institute of Technology Karnataka, Surathkal

Date: 23 April , 2021

Submitted To:
Dr. M. Venkatesan
Department of Computer Science and Engineering
NITK, Surathkal

Group Members:
Nishtha Kumari - 181CO236
Bhadra Giri- 181CO113

1. Introduction

Text classification (TC) is an essential task of natural language processing (NLP). In order to improve the performance of TC, term weighting is often used to obtain effective text representation by assigning appropriate weights to each term. A term weighting scheme is generally composed of term frequency factor, collection frequency factor and normalization factor. The normalization factor is commonly used as an optional factor to offset the influence of document length. Through the investigation of the existing term weighting schemes, we found that most of them focus on finding a more effective collection frequency factor, but rarely pay attention to finding a new term frequency factor. In our paper, authors first proposed a new term frequency factor called modified term frequency (MTF). Different from the normalization factor, MTF directly modifies the raw term frequency based on the length information ofUsing modified term frequency to improve term weighting for text classification all training documents. Then they proposed a new term weighting scheme by combining MTF with an existing collection frequency factor called modified distinguishing feature selector (MDFS). They denoted their scheme by MTF-MDFS (MDFS-based MTF). Extensive experimental results on 19 benchmark text datasets and 6 real-world text datasets show that their proposed MTF and MTF-MDFS are all much better than their state-of-the-art competitors in terms of the classification accuracy and the weighted average of $F1$ of widely used base classifiers, such as MNB, SVM and LR. So,The term frequency factor and the collection frequency factor are equally important for term weighting, both of which are helpful to improve the performance of TC. Currently, they had simply combined MTF and MDFS without considering whether they maybe counteract each other to generate inappropriate weights. For example, when the length difference of training documents is particularly large, there is likely to be a situation that the term frequency factor plays a dominant role in term weighting and the role of the collection frequency factor is greatly reduced. On the other hand, when the specificity scores of terms for classes varies too much, it may also cause adverse effects. We believe that the use of more sophisticated combination methods could improve the performance of the current MTF-MDFS and make its advantage stronger. This was a major direction for our contribution.

2. Dataset Description

20 Newsgroups: This dataset contains 19997 documents of newsgroup messages, which are divided into 20 classes. Except for 997 documents in one class, there are 1000 documents in each

of the remaining 19 classes. To save training time, numbers, punctuation marks and other non-alphabetic characters are removed. At the same time, the letters are converted to lower case. The classes are:

```
['alt.atheism',
 'comp.graphics',
 'comp.os.ms-windows.misc',
 'comp.sys.ibm.pc.hardware',
 'comp.sys.mac.hardware',
 'comp.windows.x',
 'misc.forsale',
 'rec.autos',
 'rec.motorcycles',
 'rec.sport.baseball',
 'rec.sport.hockey',
 'sci.crypt',
 'sci.electronics',
 'sci.med',
 'sci.space',
 'soc.religion.christian',
 'talk.politics.guns',
 'talk.politics.mideast',
 'talk.politics.misc',
 'talk.religion.misc']
```

Each record in the dataset is actually a text file (a series of words), for example, the first file looks like this:

```
From: lerxst@wam.umd.edu (where's my thing)
Subject: WHAT car is this!?
Nntp-Posting-Host: rac3.wam.umd.edu
Organization: University of Maryland, College Park
Lines: 15

 I was wondering if anyone out there could enlighten me on this car I saw
the other day. It was a 2-door sports car, looked to be from the late 60s/
early 70s. It was called a Bricklin. The doors were really small. In addition,
the front bumper was separate from the rest of the body. This is
all I know. If anyone can tellme a model name, engine specs, years
of production, where this car is made, history, or whatever info you
have on this funky looking car, please e-mail.

Thanks,
- IL
   ---- brought to you by your neighborhood Lerxst ----
```

3. Some Notations Used in Report

List of important notations and their descriptions

fik: the raw term frequency of term $t\ i$ in document dk .

q:	the total number of classes.
n:	the total number of training documents.
$d(t\ i)$	the number of documents containing term $t\ i$.
$d(t\ i\ ,\ cj)$	the number of documents belonging to class cj containing term $t\ i$.
$d(t\ i\_\text{bar}\ ,\ cj)$	the number of documents belonging to class cj not containing term $t\ i$.
$d(t\ i\ ,\ cj\ \_\text{bar})$	the number of documents not belonging to class cj containing term $t\ i$.
$d(t\ i\_\text{bar}\ ,\ cj\_\text{bar})$	the number of documents not belonging to class cj not containing term $t\ i$
$P\ (cj\ \|t\ i\ )$	the conditional probability of class cj given the presence of term $t\ i$.
$P\ (cj\ \|t\ i\ )$	the conditional probability of the absence of class cj given the absence of term $t\ i$. \
$P\ (t\ i\ \|cj\ )$	the conditional probability of the absence of term $t\ i$ given the presence of class cj .
$P\ (t\ i\ \|cj\ )$	the conditional probability of term $t\ i$ given the absence of class cj .

4. Existing term weighting schemes

Schemes	Term frequency factor	Collection frequency factor
binary	1	1
TF	f_{ik}	1
TF-IDF (Salton and Buckley, 1988)	f_{ik}	$\log_2\left(\frac{n}{d(t_i)}\right)$
TF-CHI (Debole and Sebastiani, 2003)	f_{ik}	$\frac{n\cdot\left(d(t_i,c_j)\cdot d(\overline{t_i},\overline{c_j})-d(\overline{t_i},c_j)\cdot d(t_i,\overline{c_j})\right)^2}{d(t_i)\cdot d(\overline{t_i})\cdot d(c_j)\cdot d(\overline{c_j})}$
TF-IG (Debole and Sebastiani, 2003)	f_{ik}	$\sum_{t\in\{t_i,\overline{t_i}\}}\sum_{c\in\{c_j,\overline{c_j}\}}\frac{d(t,c)}{n}\log_2\frac{n\cdot d(t,c)}{d(t)\cdot d(c)}$
TF-GR (Debole and Sebastiani, 2003)	f_{ik}	$\frac{\sum_{t\in\{t_i,\overline{t_i}\}}\sum_{c\in\{c_j,\overline{c_j}\}}\frac{d(t,c)}{n}\log_2\frac{n\cdot d(t,c)}{d(t)\cdot d(c)}}{-\sum_{c\in\{c_j,\overline{c_j}\}}\frac{d(c)}{n}\cdot\log_2\frac{d(c)}{n}}$
TF-RF (Lan et al., 2009)	f_{ik}	$\log_2\left(2+\frac{d(t_i,c_j)}{\max\left(1,d(t_i,\overline{c_j})\right)}\right)$
TF-PB (Liu et al., 2009)	f_{ik}	$\log_2\left(1+\frac{d(t_i,c_j)}{d(\overline{t_i},c_j)}\cdot\frac{d(t_i,c_j)}{d(t_i,\overline{c_j})}\right)$
TF-ICF (Wang and Zhang, 2013)	f_{ik}	$\log_2\left(1+\frac{q}{c(t_i)}\right)$
TF-ICF-Based (Wang and Zhang, 2013)	f_{ik}	$\log_2\left(2+\frac{d(t_i,c_j)}{\max\left(1,d(t_i,\overline{c_j})\right)}\cdot\frac{q}{c(t_i)}\right)$
TF-IDF-ICF (Ren and Sohrab, 2013)	f_{ik}	$\left(1+\log_2\frac{n}{d(t_i)}\right)\cdot\left(1+\log_2\frac{q}{c(t_i)}\right)$
TF-IDF-ICSDF (Ren and Sohrab, 2013)	f_{ik}	$\left(1+\log_2\frac{n}{d(t_i)}\right)\cdot\left(1+\log_2\frac{q}{\sum_{j=1}^{q}\frac{d(t_i,c_j)}{d(c_j)}}\right)$
TF-DC (Wang et al., 2015b)	f_{ik}	$1+\frac{1}{\log_2 q}\cdot\sum_{j=1}^{q}\frac{d(t_i,c_j)}{d(t_i)}\log_2\frac{d(t_i,c_j)}{d(t_i)}$
TF-BDC (Wang et al., 2015b)	f_{ik}	$1+\frac{1}{\log_2 q}\cdot\sum_{j=1}^{q}\frac{\frac{d(t_i,c_j)}{d(c_j)}}{\sum_{j=1}^{q}\frac{d(t_i,c_j)}{d(c_j)}}\log_2\frac{\frac{d(t_i,c_j)}{d(c_j)}}{\sum_{j=1}^{q}\frac{d(t_i,c_j)}{d(c_j)}}$
TF-IGM (Chen et al., 2016)	f_{ik}	$1+\lambda\frac{f_{ik}}{\sum_{r=1}^{q}f_{ir}\cdot r}$
TF-IGM$_{imp}$ (Dogan and Uysal, 2019)	f_{ik}	$1+\lambda\frac{f_{ik}}{\sum_{r=1}^{q}f_{ir}\cdot r+\log_{10}\left\lfloor\frac{D_{\max}}{f_{ik}}\right\rfloor}$
TF-MDFS (Chen et al., 2021)	f_{ik}	$\sum_{j=1}^{q}\log_2\left(1+\frac{d(t_i,c_j)}{\max\left(1,d(t_i,\overline{c_j})\right)}\cdot\frac{d(\overline{t_i},\overline{c_j})}{\max\left(1,d(\overline{t_i},c_j)\right)}\right)\cdot\frac{P(c_j\|t_i)+P(\overline{c_j}\|\overline{t_i})}{P(\overline{t_i}\|c_j)+P(t_i\|\overline{c_j})+1}$
LogTF-RF$_{max}$ (Xuan and Quang, 2013)	$\log_2(1+f_{ik})$	$\max_{j=1}^{q}\log_2\left(2+\frac{d(t_i,c_j)}{\max\left(1,d(t_i,\overline{c_j})\right)}\right)$
SqrtTF-IGM (Chen et al., 2016)	$\sqrt{f_{ik}}$	$1+\lambda\frac{f_{ik}}{\sum_{r=1}^{q}f_{ir}\cdot r}$
SqrtTF-IGM$_{imp}$ (Dogan and Uysal, 2019)	$\sqrt{f_{ik}}$	$1+\lambda\frac{f_{ik}}{\sum_{r=1}^{q}f_{ir}\cdot r+\log_{10}\left\lfloor\frac{D_{\max}}{f_{ik}}\right\rfloor}$

5. Methodology

5.1 Text pre-processing

5.1.1 Convert lower case:

During the text processing each sentence is split to words and each word is considered as a token after preprocessing. Programming languages consider textual data as sensitive, which means that The is different from the. we humans know that those both belong to same token but due to the character encoding those are considered as different tokens. Converting to lowercase is a very mandatory preprocessing step. As we have all our data in list, numpy has a method which can convert the list of lists to lowercase at once.

```
np.char.lower(data)
```

5.1.2 Remove Stopwords

Stop words are the most commonly occurring words which don't give any additional value to the document vector. in-fact removing these will increase computation and space efficiency. nltk library has a method to download the stopwords, so instead of explicitly mentioning all the stopwords ourselves we can just use the nltk library and iterate over all the words and remove the stop words. There are many efficient ways to do this, but ill just give a simple method.

```
1 print(stopwords.words('english'))
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you',
'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'sh
t's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves
t', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'b
ng', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'i
f', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into',
e', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'o
e', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'bo
me', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than',
'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 'r
n', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't
"isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "ne
n't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn',
```

We are going to iterate over all the stop words and not append to the list if it's a stop word.

```
new_text = ""
for word in words:
    if word not in stop_words:
        new_text = new_text + " " + word
```

5.1.3 Punctuation

Punctuation are the unnecessary symbols that are in our corpus documents, we should be little careful with what we are doing with this. There might be few problems such as U.S — us “United Stated” being converted to “us” after the preprocessing. hyphen and should usually be dealt little carefully. but for this problem statement we are just going to remove these.

```
symbols = "!\"#$%&()*+-./:;<=>?@[\]^_`{|}~\n"
for i in symbols:
    data = np.char.replace(data, i, ' ')
```

We are going to store all our symbols in a variable and iterate that variable removing that particular symbol in the whole dataset. we are using numpy here because our data is stored in list of lists, and numpy is our best best.

5.1.4 Apostrophe

Note that there is no ‘ apostrophe in the punctuation symbols. Because when we remove punctuation first it will convert don’t to dont, and it is a stop word which wont be removed. so what we are doing is we are first removing the stop words, and then symbols and then finally stopwords because few words might still have a apostrophe which are not stop words.

```
return np.char.replace(data, "'", "")
```

5.1.5 Single Characters

Single characters are not much useful in knowing the importance of the document and few final single characters might be irrelevant symbols, so it is always good be remove the single characters.

```
new_text = ""
for w in words:
    if len(w) > 1:
       new_text = new_text + " " + w
```

We just need to iterate to all the words and not append the word if the length is not greater than 1.

5.1.6 Stemming

This is the final and most important part of the preprocessing. stemming converts words to its stem.

For example **playing and played** are the same type of words which basically indicate an action **play.** Stemmer does exactly this, it reduces the word to its stem. we are going to use a library called porter-stemmer which is a rule based stemmer. Porter-Stemmer identifies and removes the suffix or affix of a word. The words given by the stemmer need note be meaningful few times, but it will be identified as the same for the model.

```
1 stemmer = PorterStemmer()
2 stemmer.stem("swimming")
```

```
'swim'
```

5.2 Calculating MTF

For each document ***dk*** (k=1,2,…,n), the density function ***ρ(l)*** of term ***ti*** is defined as:

$$\rho(l) = s \cdot \frac{f_{ik}}{l},$$

where ***s*** is a constant, and ***fik*** is the raw term frequency of term ***ti*** in ***dk***.

To resize the term frequency, we calculate the integral on the same interval [l(dk),l(dk)+avg_l], where ***l(dk)*** is the document length of dk, which is defined as the sum of all term frequencies, and ***avg_l*** is the average document length of all training documents. The resized term frequency (simply denoted as **RTF)** is defined as:

$$RTF\left(t_i, d_k\right) = \int_{l(d_k)}^{l(d_k)+avg\_l} \rho\left(l\right) dl.$$

the ***RTF*** of term ***ti*** in document ***dk*** is

$$\begin{aligned} &RTF_2\left(t_i, d_k\right) = \int_{l(d_k)}^{l(d_k)+avg\_l} s \cdot \frac{f_{ik}}{l} dl \\ &= s \cdot f_{ik} \cdot \int_{l(d_k)}^{l(d_k)+avg\_l} \frac{dl}{l} \\ &= s \cdot f_{ik} \cdot \log_e \left(1 + \frac{avg\_l}{l(d_k)}\right). \end{aligned}$$

Assume that the average length of all training documents is equal to the effective length of the document. When **l(dk)** is equal to ***avg_l, RTF(ti,dk)*** is equal to ***fik***. So in this case, we can figure out that s=1/loge2=log2e.

$$RTF_2\left(t_i, d_k\right) = \log_2 e \cdot f_{ik} \cdot \log_e \left(1 + \frac{avg\_l}{l(d_k)}\right)$$
$$= f_{ik} \cdot \log_2 \left(1 + \frac{avg\_l}{l(d_k)}\right).$$

uWe consider that when the document length differs greatly, it is easy to cause the RTF value to be too large or too small. So we also use a square root function to suppress this effect. Based on these premises, we propose a new term frequency factor called ***modified term frequency*** (MTF), which is defined as:

$$MTF\left(t_i, d_k\right) = \sqrt{f_{ik} \cdot \log_2 \left(1 + \frac{avg\_l}{l(d_k)}\right)}.$$

5.3 MDFS

MDFS makes full use of the distribution information of terms in all training documents, and it is more effective than other collection frequency factors for term weighting in most cases.

MDFS is derived from a widely accepted term selection method called ***distinguishing feature selector*** (DFS). The DFS argues that an ideal term (feature) selection method should assign high scores to distinctive terms and assign low scores to irrelevant terms. Specifically, DFS must meet four requirements:

1. **If a term occurs frequently in a single class and does not occur in other classes, it is distinctive and must be assigned a high score.**
2. **If a term occurs frequently in all classes, it is irrelevant and must be assigned a low score.**
3. **If a term occurs rarely in a single class and does not occur in other classes, it is irrelevant and must be assigned a low score.**
4. **If a term occurs in some of the classes,**

it is relatively distinctive and must be assigned a medium score.

$$DFS(t_i) = \sum_{j=1}^{q} \frac{P(c_j|t_i)}{P\left(\bar{t}_i|c_j\right)+P(t_i|\bar{c}_j)+1}.$$

By analyzing the formula of DFS, we found that the specificity of a term in a class has not been adequately demonstrated. To address this issue, we argue that an ideal term selection score should satisfy the fifth requirement:

"For class cj, d(ti,cj) must be large and d(t¯i,cj) must be small; For class c¯j (absence of class cj), d(t¯i,c¯j) must be large and d(ti,c¯j) must be small".

Then we modified the DFS and proposed a new collection frequency factor simply denoted by MDFS. In MDFS, each term ***ti*** has a class-specific score for each class ***cj.***

In addition, a weighting factor is used for class-specific scores to further reflect the contributions of terms to a single class. MDFS is defined as a global weighting factor, calculated by the weighted sum across all class-specific scores. The detailed formula is:

$$MDFS(t_i) = \sum_{j=1}^{q} w_{ij} \cdot MDFS_{cs}(t_i, c_j),$$

where ***wij*** and ***MDFScs(ti,cj)*** represent the weighting factor and class-specific score of term ***ti*** for class ***cj***, respectively. They are defined respectively as follows:

$$w_{ij} = \log_2\left(1 + \frac{d(t_i,c_j)}{\max(1,d(t_i,\bar{c}_j))} \cdot \frac{d\left(\bar{t}_i,\bar{c}_j\right)}{\max\left(1,d\left(\bar{t}_i,c_j\right)\right)}\right),$$

$$MDFS_{cs}(t_i, c_j) = \frac{P(c_j|t_i)P\left(\bar{c}_j|\bar{t}_i\right)}{P\left(\bar{t}_i|c_j\right)+P(t_i|\bar{c}_j)+1}.$$

5.4 W-MTF-MDFS

Finally, by combining MTF with MDFS, we propose a new term weighting scheme called ***MTF-MDFS (MDFS-based MTF).*** The detailed formula is:

$$W_{\text{MTF - MDFS}}(t_i, d_k) = MTF(t_i, d_k) \cdot MDFS(t_i).$$

6. Algorithm

6.1 MTF-MDFS learning in the training phase (D)

Input: D-a training document set
Output: All term weights WMTF-MDFS($t\ i$, dk), avg_l, $MDF\ S(t\ i$) (i = 1, 2, ⋯ , m)
1.for each document dk (k = 1, 2, ⋯ , n) do
2. Calculate $l(dk$)
3.end for
4.Calculate $avg_$l
5.for each term $t\ i$ (i = 1, 2, ⋯ , m) and each class cj (j = 1, 2, ⋯ , q) do
6. Calculate wij by Eq. (11)
7. Calculate $MDFS\ cs(ti$, cj)
8.end for
9.for each term ti (i = 1, 2, ⋯ , m) do
10. Calculate $MDFS(t\ i$) by Eq. (10)
11. for each document dk (k = 1, 2, ⋯ , n) do
12. Calculate $MTF(ti$, dk)
13. Calculate WMTF-MDFS(ti , dk)
14. end for
15.end for
16.return All term weights WMTF-MDFS($t\ i$, dk), avg_l, $MDF\ S(t\ i$) (i = 1, 2, ⋯ , m)

6.2 MTF-MDFS learning in the testing phase (d)

Input: d-a test document, avg_l, $MDF\ S(t\ i$) (i = 1, 2, ⋯ , m)
Output: All term weights WMTF-MDFS($t\ i$, d)
1.Calculate $l(d)$
2.for each term $t\ i$ (i = 1, 2, ⋯ , m) do
3. Calculate $MT\ F(t\ i$, d)
4. Calculate WMTF-MDFS($t\ i$, d)
5.end for
6.return All term weights

7. Experiment Results

Accuracy of MTF-MDFS using different models
Naive Bayes 86%

SVM	79%
Logistic Regression	80%
Xgboost	59%

Naive Bayes classifier gave the highest accuracy-86%

The classification report for Naive Bayes classifier is given below.

```
Accuracy achieved is 0.86
                        precision    recall  f1-score   support

           alt.atheism       0.82      0.70      0.76        20
         comp.graphics       0.85      1.00      0.92        29
               sci.med       1.00      0.76      0.86        25
soc.religion.christian       0.80      0.92      0.86        26

              accuracy                           0.86       100
             macro avg       0.87      0.85      0.85       100
          weighted avg       0.87      0.86      0.86       100


array([[14,  1,  0,  5],
       [ 0, 29,  0,  0],
       [ 1,  4, 19,  1],
       [ 2,  0,  0, 24]], dtype=int64)
```

7. Matching Score and Cosine Similarity

7.1 Ranking using Matching Score

Matching score is the most simplest way to calculate the similarity, in this method, we add tf_idf values of the tokens that are in query for every document. for example, if the query “hello world”, we need to check in every document if these words exists and if the word exists, then the mtf_mdfs value is added to the matching score of that particular doc_id. in the end we will sort and take the top k documents.

Matching Score gives relevant documents, but fails when we give long queries and will not be able to rank them properly.

7.2 Ranking using Cosine Similarity

When we have a perfectly working **Matching Score**, why do we need cosine similarity again? though **Matching Score** gives relevant documents, it quite fails when we give long queries, it

will not be able to rank them properly. what cosine similarly does is that it will mark all the documents as vectors of tf-idf tokens and plots them from the centre. what will happen is that, few times the query length would be small but it might be closely related to the document, in these cases cosine similarity is the best want to find relevance.

8. Contribution

Proposed two new weighting scheme by modifying ICF, MDFS and DFS.

8.1 MTF-ICF-DFS

•Accuracy with multinomial naive bayes: 88%.
DFS:

$$DFS(t) = \sum_{i=1}^{M} \frac{P(C_i|t)}{P(\bar{t}|C_i) + P(t|\bar{C}_i) + 1},$$

ICF:

$$\log_2\left(1 + \frac{q}{c(t_i)}\right)$$

New weighing scheme: weight(ti,dk)= MTF*ICF*DFS
The classification report for the weighting scheme with the Naive Bayes Classifier is given.

```
Accuracy achieved is 0.88
                        precision    recall  f1-score   support

           alt.atheism       0.88      0.70      0.78        20
         comp.graphics       0.88      1.00      0.94        29
               sci.med       1.00      0.80      0.89        25
soc.religion.christian       0.81      0.96      0.88        26

              accuracy                           0.88       100
             macro avg       0.89      0.87      0.87       100
          weighted avg       0.89      0.88      0.88       100


array([[14,  1,  0,  5],
       [ 0, 29,  0,  0],
       [ 1,  3, 20,  1],
       [ 1,  0,  0, 25]], dtype=int64)
```

8.2 MTF-ICF-MDFS

•Accuracy obtained with multinomial naives bayes is 89% (best so far)
•Formula:
MDFS:

$$\sum_{j=1}^{q} \log_2 \left(1 + \frac{d(t_i,c_j)}{\max\left(1, d(t_i,\overline{c_j})\right)} \cdot \frac{d(\overline{t_i},\overline{c_j})}{\max\left(1, d(\overline{t_i},c_j)\right)} \right) \cdot \frac{P(c_j|t_i)P(\overline{c}_j|\overline{t}_i)}{P(\overline{t}_i|c_j)+P(t_i|\overline{c}_j)+1}$$

ICF:

$$\log_2 \left(1 + \frac{q}{c(t_i)} \right)$$

New weighing scheme: weight(ti,dk) = MTF*ICF*MDFS

The classification report for the weighting scheme with the Naive Bayes Classifier is given in Fig3.

```
Accuracy achieved is 0.89
                         precision    recall  f1-score   support

           alt.atheism       0.88      0.75      0.81        20
         comp.graphics       0.88      1.00      0.94        29
               sci.med       1.00      0.80      0.89        25
soc.religion.christian       0.83      0.96      0.89        26

              accuracy                           0.89       100
             macro avg       0.90      0.88      0.88       100
          weighted avg       0.90      0.89      0.89       100


array([[15,  1,  0,  4],
       [ 0, 29,  0,  0],
       [ 1,  3, 20,  1],
       [ 1,  0,  0, 25]], dtype=int64)
```

9. Conclusion

Mtf-mdfs implemented through the the paper has the best accuracy with Naïve bayes classifier (**86%).**

As part of our contribution,mtf-icf, mtf-dfs-icf and mtf-mdfs-icf gives **89%**, **88%** and **89%** accuracy respectively.