# *Naive Bayes classifier for Fake News recognition*

Marija Mojsovska
Andrea Peri
Filippo Orlando

***Dipartimento di Fisica e Astronomia "G. Galilei"***

Università degli Studi di Padova

## Introduction

Being able to distinguish fake contents form real news, especially from social media platforms in the online world, is today one of the most serious challenges facing the news industry.

The goal of this project is to implement a Multinomial Naive Bayes classifier and test its performances in the classification of social media posts.

The data we use for training and testing the algorithm consists of a set with 10,240 instances and has the following labels for classifying the text:

| | |
|---|---|
| *Barely-True - 0* | *Mostly-True – 3* |
| *False – 1* | *Not-Known – 4* |
| *Half-True – 2* | *True – 5* |

# Naive Bayes text classification

The multinomial Naive Bayes model is a probabilistic learning method. The probability of a document *d* being in class *c* is computed as

$$P(c|d) \propto P(c) \prod_{1 \le k \le n_d} P(t_k|c)$$

*P(tₖ|c)* is the conditional probability of term *tₖ* occurring in a document of class *c*
*P(c)* is the prior probability of a document occurring in class *c*

The class of a document is determent finding the most likely class $c_{map}$

$$c_{\text{map}} = \arg\max_{c \in \mathbb{C}} [\log \hat{P}(c) + \sum_{1 \le k \le n_d} \log \hat{P}(t_k|c)].$$

$$\hat{P}(c) = \frac{N_c}{N}$$

$N_c$ is the number of documents in class *c*
*N* is the total number of documents

$$\hat{P}(t|c) = \frac{T_{ct} + 1}{\sum_{t' \in V}(T_{ct'} + 1)}$$

$T_{ct}$ is the number of occurrences of *t* in training documents from class *c* including multiple occurrences of a term in a document

## Multinomial Naive Bayes algorithm: Training

```
TrainMultinomialNB($\mathbb{C}$, $\mathbb{D}$)
  $V \leftarrow$ ExtractVocabulary($\mathbb{D}$)
  $N \leftarrow$ CountDocs($\mathbb{D}$)
  for each $c \in \mathbb{C}$
  do $N_c \leftarrow$ CountDocsInClass($\mathbb{D}, c$)
     $prior[c] \leftarrow N_c / N$
     $text_c \leftarrow$ ConcatenateTextOfAllDocsInClass($\mathbb{D}, c$)
     for each $t \in V$
     do $T_{ct} \leftarrow$ CountTokensOfTerm($text_c, t$)
     for each $t \in V$
     do $condprob[t][c] \leftarrow \frac{T_{ct}+1}{\sum_{t'}(T_{ct'}+1)}$
  return $V, prior, condprob$
```

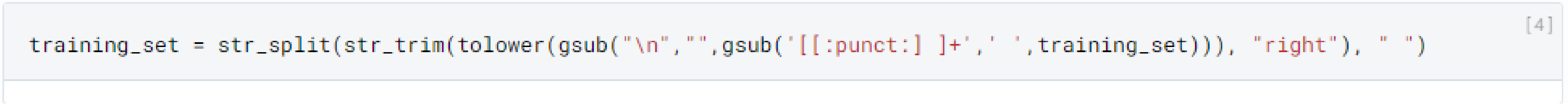
```
ApplyMultinomialNB($\mathbb{C}, V, prior, condprob, d$)
  $W \leftarrow$ ExtractTokensFromDoc($V, d$)
  for each $c \in \mathbb{C}$
  do $score[c] \leftarrow \log prior[c]$
     for each $t \in W$
     do $score[c] \mathrel{+}= \log condprob[t][c]$
  return $\arg\max_{c \in \mathbb{C}} score[c]$
```

# Loading and preprocessing data

- Total number of documents: 10234
- Raw data is a dataframe containing columns: Labels - Text - Text_Tag
- 85% as training set
- 15% as test set

In the next cell, different operations are applied at the raw set:

- **Inner gsub() function**: all punctuation element are replaced with a blanck
- **Outer gsub() function**: all the newline special characters (\n) are deleted
- **tolower()**: all characters are represented in lowercase
- **str_trim()**: the spaces at the end of every document are deleted
- **str_split()**: every document is splitted in single words



```
[4]
training_set = str_split(str_trim(tolower(gsub("\n","",gsub('[[:punct:] ]+',' ',training_set))), "right"), " ")
```

## Loading and preprocessing data

[1] "Sixteen thousand new Internal Revenue Service agents will be empowered to enforce the new health care legislation.“

[2] "The proposed mine in northern Wisconsin would be built without any government oversight, and will be nine miles long.“

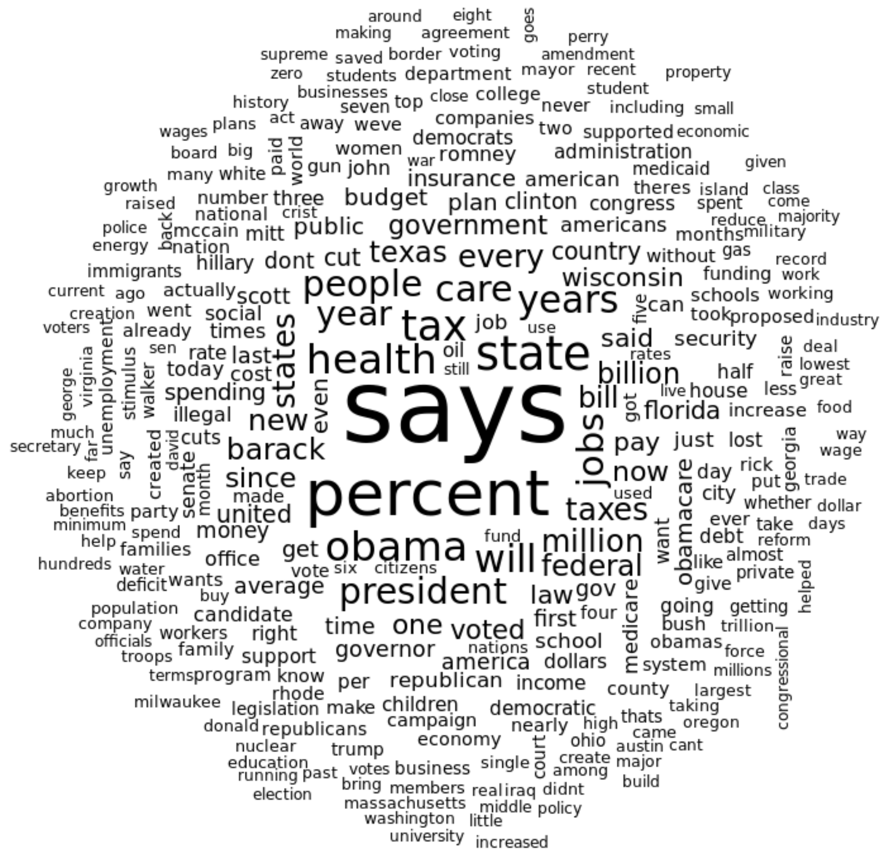
[3] "Says Williamson County Attorney Jana Duty has never prosecuted a single adult felony case."

```
[1] "sixteen"     "thousand"    "new"         "internal"    "revenue"     "service"     "agents"      "will"        "be"
    "empowered"   "to"          "enforce"     "the"         "new"         "health"      "care"        "legislation“

[2] "the"         "proposed"    "mine"        "in"          "northern"    "wisconsin"   "would"       "be"          "built"       "without"
    "any"         "government"  "oversight"   "and"         "will"        "be"          "nine"        "miles"       "long"

[3] "says"        "williamson"  "county"      "attorney"    "jana"        "duty"        "has"         "never"       "prosecuted"  "a"
    "single"      "adult"       "felony"      "case"
```

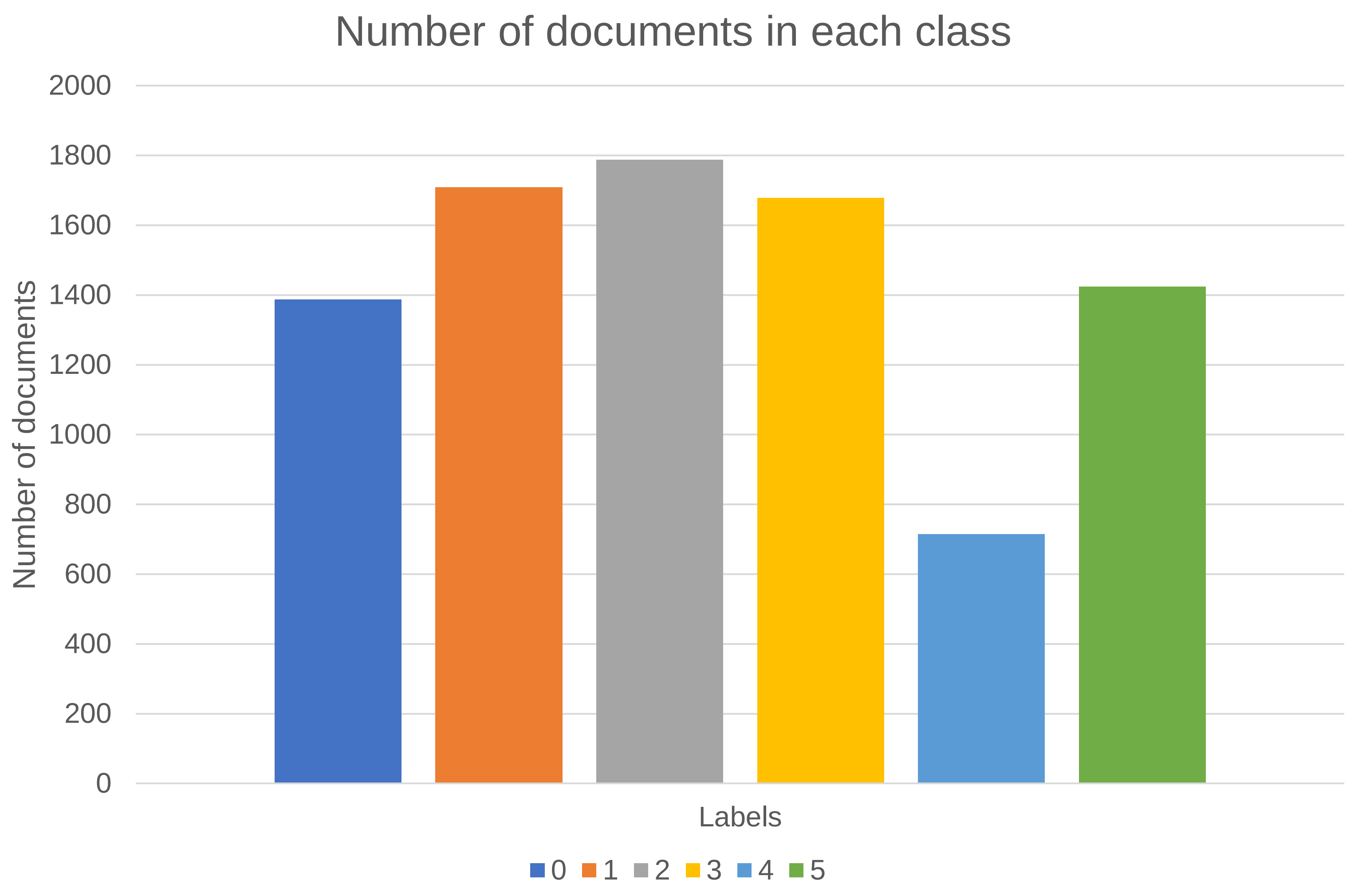
## Extract vocabulary

- Total number of words in the training set: 11276

```r
start_time <- Sys.time()

voc <- character()

for (i in training_set){
    for(j in i){
        if (!(j %in% voc))
            voc <- c(voc,j)
    }
}

end_time <- Sys.time()
end_time - start_time
```

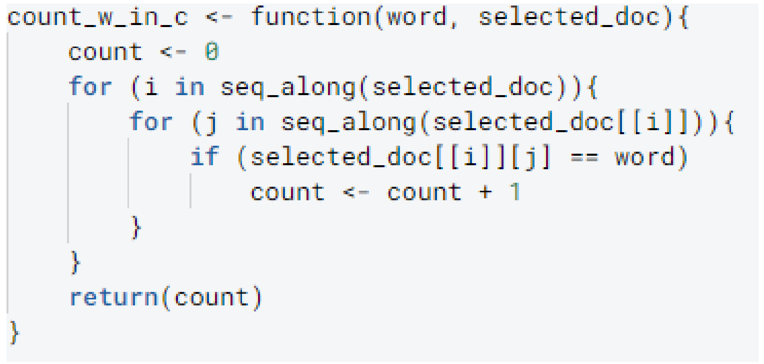


# Number of documents in each class

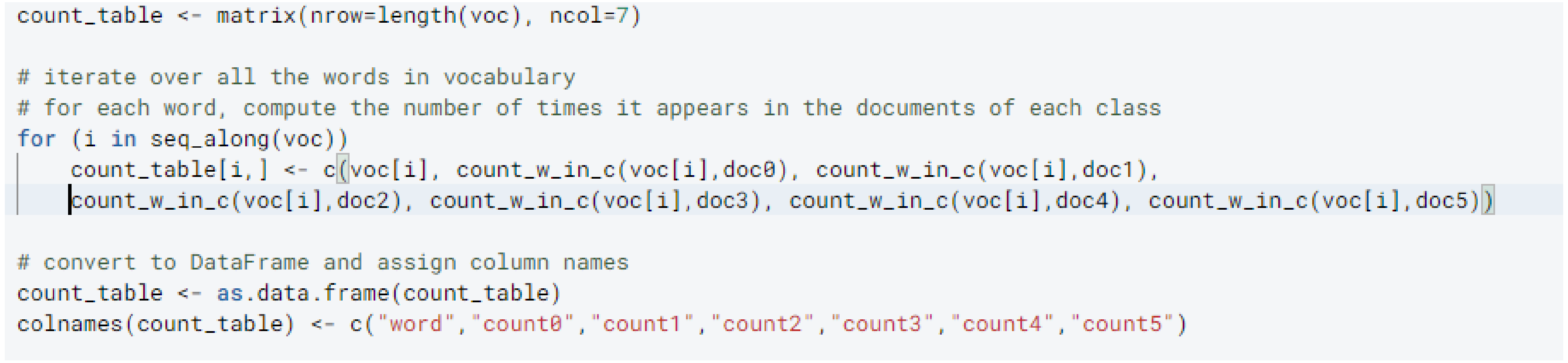


# Compute count_table

- The table will contain the number of appearances for each word in all the documents in each class.

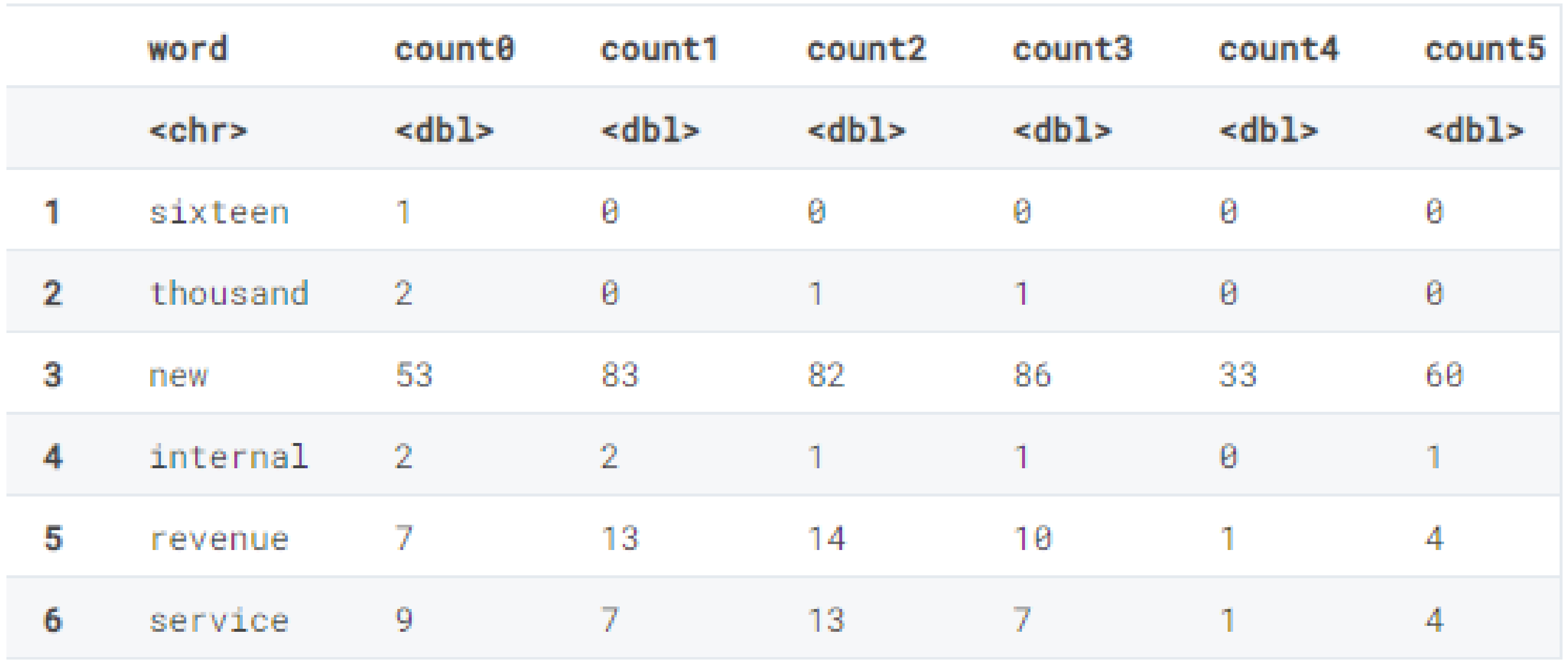


```r
count_w_in_c <- function(word, selected_doc){
    count <- 0
    for (i in seq_along(selected_doc)){
        for (j in seq_along(selected_doc[[i]])){
            if (selected_doc[[i]][j] == word)
                count <- count + 1
        }
    }
    return(count)
}
```



```r
count_table <- matrix(nrow=length(voc), ncol=7)

# iterate over all the words in vocabulary
# for each word, compute the number of times it appears in the documents of each class
for (i in seq_along(voc))
    count_table[i,] <- c(voc[i], count_w_in_c(voc[i],doc0), count_w_in_c(voc[i],doc1),
    count_w_in_c(voc[i],doc2), count_w_in_c(voc[i],doc3), count_w_in_c(voc[i],doc4), count_w_in_c(voc[i],doc5))

# convert to DataFrame and assign column names
count_table <- as.data.frame(count_table)
colnames(count_table) <- c("word","count0","count1","count2","count3","count4","count5")
```

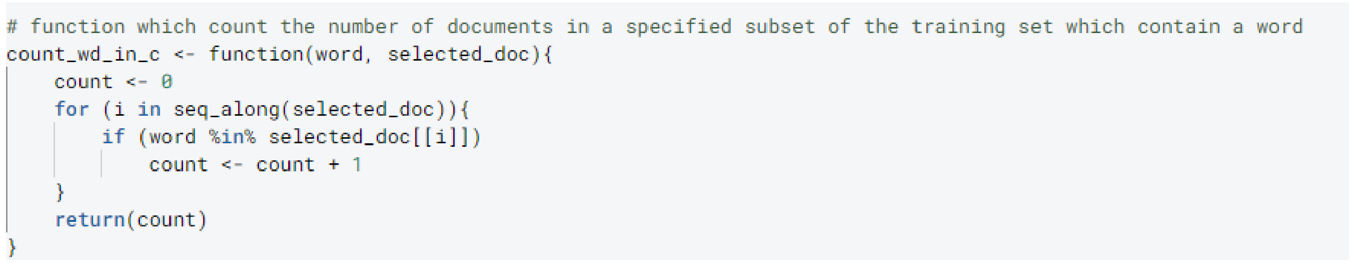
# Compute count_table



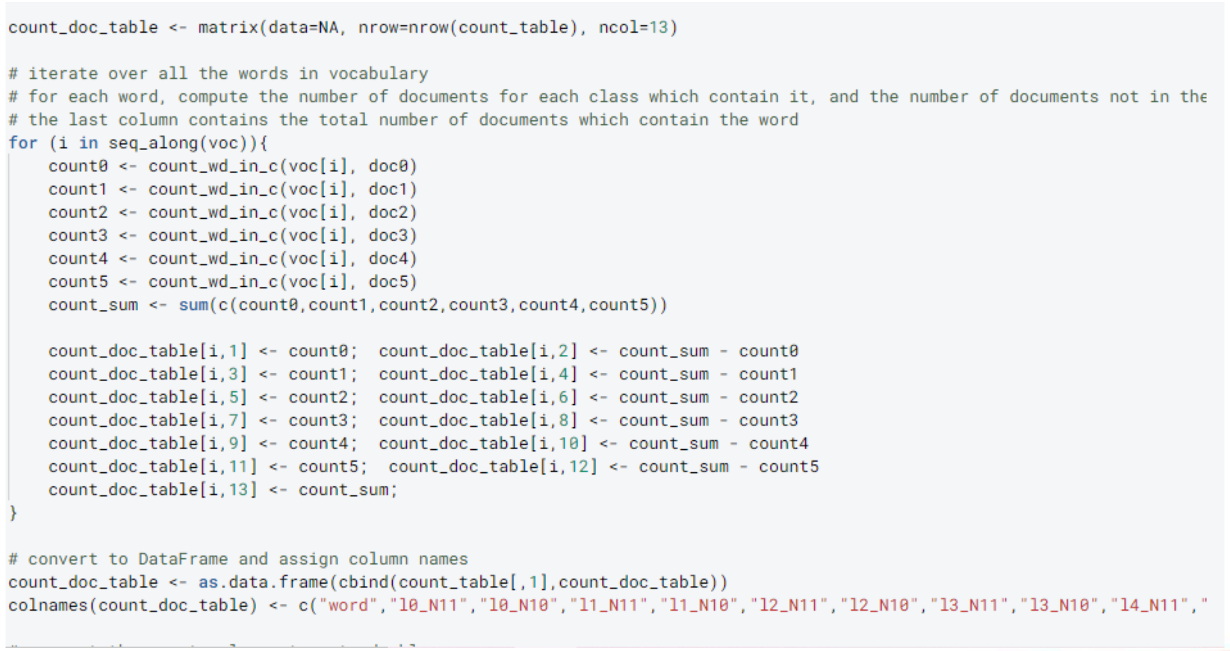
| | word | count0 | count1 | count2 | count3 | count4 | count5 |
|---|---|---|---|---|---|---|---|
| | <chr> | <dbl> | <dbl> | <dbl> | <dbl> | <dbl> | <dbl> |
| 1 | sixteen | 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | thousand | 2 | 0 | 1 | 1 | 0 | 0 |
| 3 | new | 53 | 83 | 82 | 86 | 33 | 60 |
| 4 | internal | 2 | 2 | 1 | 1 | 0 | 1 |
| 5 | revenue | 7 | 13 | 14 | 10 | 1 | 4 |
| 6 | service | 9 | 7 | 13 | 7 | 1 | 4 |

## Feature selection – count_doc_table

- This table will contain the number of documents for each class which contain a specific word
- This information is needed to use Mutual Information and as feature selection strategies

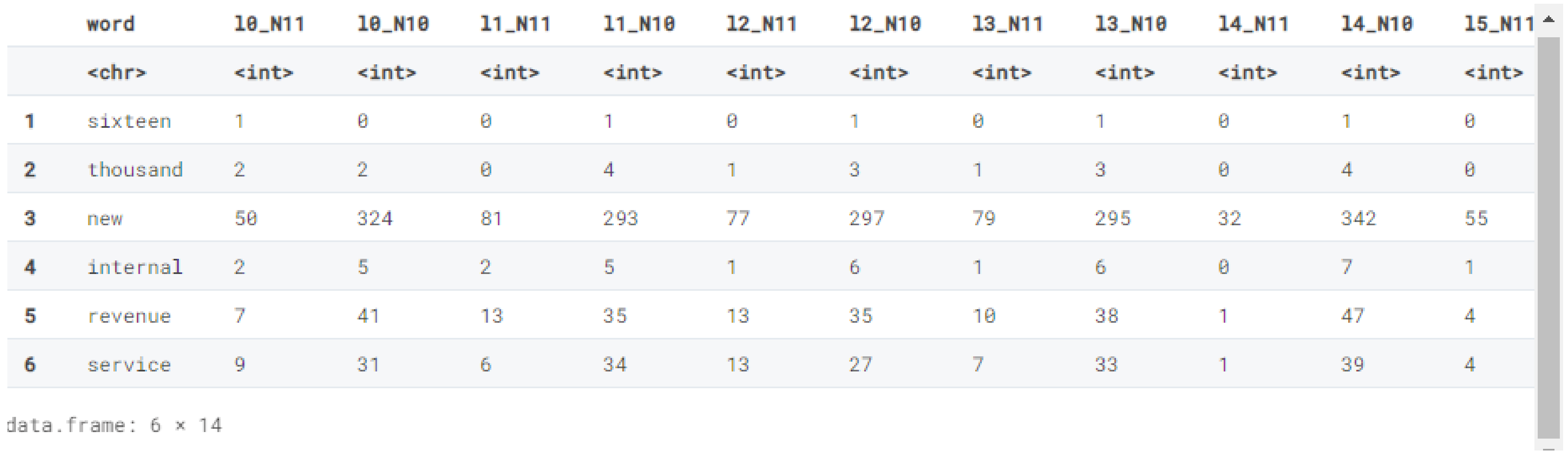


```r
# function which count the number of documents in a specified subset of the training set which contain a word
count_wd_in_c <- function(word, selected_doc){
    count <- 0
    for (i in seq_along(selected_doc)){
        if (word %in% selected_doc[[i]])
            count <- count + 1
    }
    return(count)
}
```



```r
count_doc_table <- matrix(data=NA, nrow=nrow(count_table), ncol=13)

# iterate over all the words in vocabulary
# for each word, compute the number of documents for each class which contain it, and the number of documents not in the
# the last column contains the total number of documents which contain the word
for (i in seq_along(voc)){
    count0 <- count_wd_in_c(voc[i], doc0)
    count1 <- count_wd_in_c(voc[i], doc1)
    count2 <- count_wd_in_c(voc[i], doc2)
    count3 <- count_wd_in_c(voc[i], doc3)
    count4 <- count_wd_in_c(voc[i], doc4)
    count5 <- count_wd_in_c(voc[i], doc5)
    count_sum <- sum(c(count0,count1,count2,count3,count4,count5))

    count_doc_table[i,1] <- count0;  count_doc_table[i,2] <- count_sum - count0
    count_doc_table[i,3] <- count1;  count_doc_table[i,4] <- count_sum - count1
    count_doc_table[i,5] <- count2;  count_doc_table[i,6] <- count_sum - count2
    count_doc_table[i,7] <- count3;  count_doc_table[i,8] <- count_sum - count3
    count_doc_table[i,9] <- count4;  count_doc_table[i,10] <- count_sum - count4
    count_doc_table[i,11] <- count5;  count_doc_table[i,12] <- count_sum - count5
    count_doc_table[i,13] <- count_sum;
}

# convert to DataFrame and assign column names
count_doc_table <- as.data.frame(cbind(count_table[,1],count_doc_table))
colnames(count_doc_table) <- c("word","l0_N11","l0_N10","l1_N11","l1_N10","l2_N11","l2_N10","l3_N11","l3_N10","l4_N11","
```

# Feature selection – count_doc_table



| | word | 10_N11 | 10_N10 | 11_N11 | 11_N10 | 12_N11 | 12_N10 | 13_N11 | 13_N10 | 14_N11 | 14_N10 | 15_N11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | <chr> | <int> | <int> | <int> | <int> | <int> | <int> | <int> | <int> | <int> | <int> | <int> |
| 1 | sixteen | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 2 | thousand | 2 | 2 | 0 | 4 | 1 | 3 | 1 | 3 | 0 | 4 | 0 |
| 3 | new | 50 | 324 | 81 | 293 | 77 | 297 | 79 | 295 | 32 | 342 | 55 |
| 4 | internal | 2 | 5 | 2 | 5 | 1 | 6 | 1 | 6 | 0 | 7 | 1 |
| 5 | revenue | 7 | 41 | 13 | 35 | 13 | 35 | 10 | 38 | 1 | 47 | 4 |
| 6 | service | 9 | 31 | 6 | 34 | 13 | 27 | 7 | 33 | 1 | 39 | 4 |

data.frame: 6 × 14

## Feature selection

*Feature selection* is the process of selecting a subset of the terms occurring in the training set and using only this subset as features in the classification.

For a given class *c*, a utility measure *A(t,c)* is computed for each term of the vocabulary and the *k* terms that have the highest values of *A(t,c) are* selected. All other terms are discarded and not used in the classification.

We will use two utility measures:

- **mutual information**

  $$A(t, c) = I(U_t ; C_c)$$

- **χ2 test**

  $$A(t, c) = X^2 (t, c)$$

# Mutual Information

The *mutual information* measures how much information the presence/absence of a term contributes to making the correct classification decision on the class.

The utility measure for mutual information is computed using the following formula

$$I(U;C) = \sum_{e_t \in \{1,0\}} \sum_{e_c \in \{1,0\}} P(U = e_t, C = e_c) \log_2 \frac{P(U = e_t, C = e_c)}{P(U = e_t) P(C = e_c)}$$

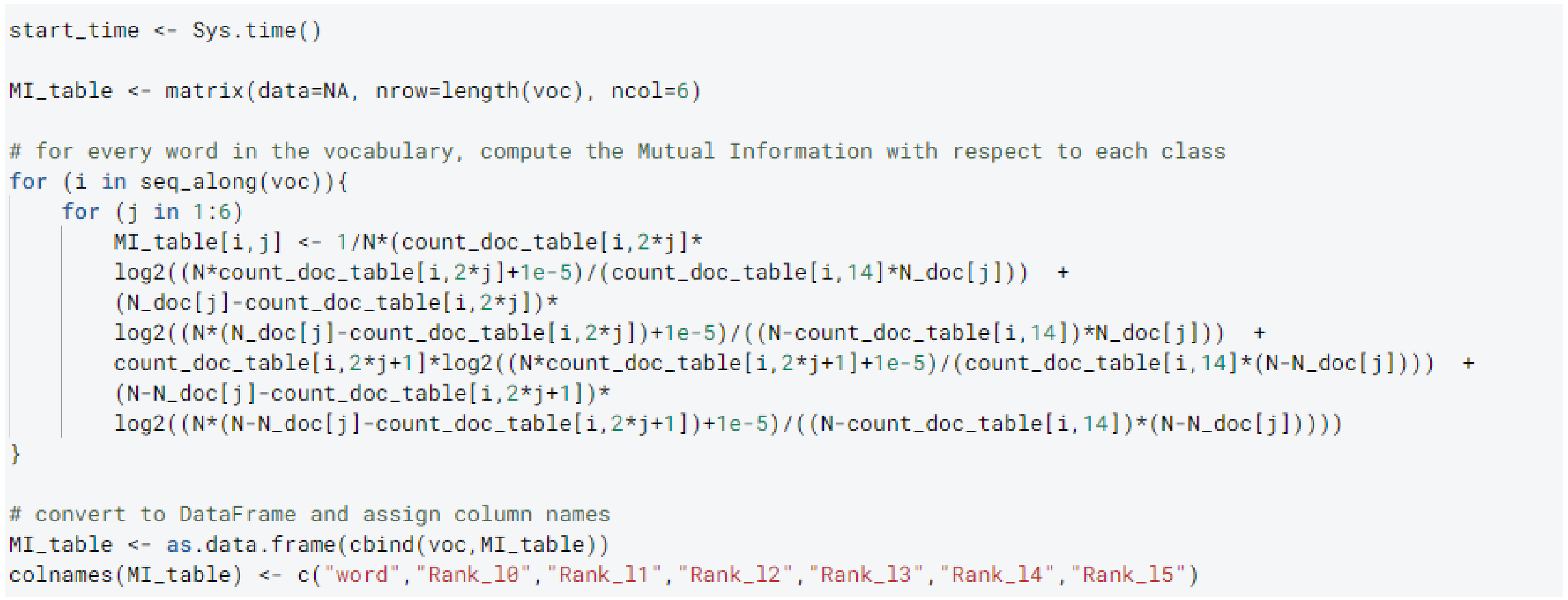
*U* is a random variable that takes values

| | | |
|---|---|---|
| $e_t = 1$ | $\rightarrow$ | the document contains term *t* |
| $e_t = 0$ | $\rightarrow$ | the document does not contain *t* |

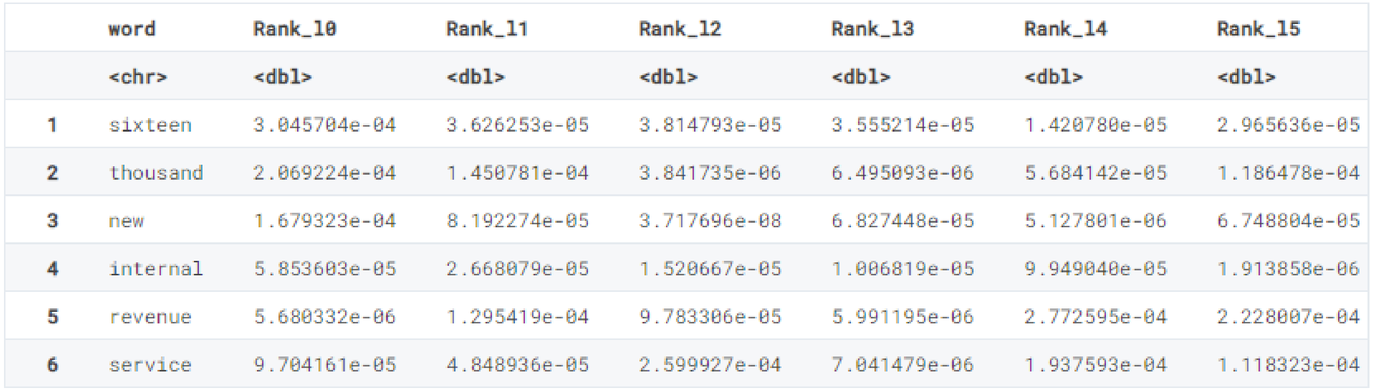
*C* is a random variable that takes values

| | | |
|---|---|---|
| $e_c = 1$ | $\rightarrow$ | the document is in class *c* |
| $e_c = 0$ | $\rightarrow$ | the document is not in class *c* |

# Compute Mutual Information



```
start_time <- Sys.time()

MI_table <- matrix(data=NA, nrow=length(voc), ncol=6)

# for every word in the vocabulary, compute the Mutual Information with respect to each class
for (i in seq_along(voc)){
    for (j in 1:6)
        MI_table[i,j] <- 1/N*(count_doc_table[i,2*j]*
        log2((N*count_doc_table[i,2*j]+1e-5)/(count_doc_table[i,14]*N_doc[j]))  +
        (N_doc[j]-count_doc_table[i,2*j])*
        log2((N*(N_doc[j]-count_doc_table[i,2*j])+1e-5)/((N-count_doc_table[i,14])*N_doc[j]))  +
        count_doc_table[i,2*j+1]*log2((N*count_doc_table[i,2*j+1]+1e-5)/(count_doc_table[i,14]*(N-N_doc[j])))  +
        (N-N_doc[j]-count_doc_table[i,2*j+1])*
        log2((N*(N-N_doc[j]-count_doc_table[i,2*j+1])+1e-5)/((N-count_doc_table[i,14])*(N-N_doc[j]))))
}

# convert to DataFrame and assign column names
MI_table <- as.data.frame(cbind(voc,MI_table))
colnames(MI_table) <- c("word","Rank_l0","Rank_l1","Rank_l2","Rank_l3","Rank_l4","Rank_l5")
```



| | word | Rank_l0 | Rank_l1 | Rank_l2 | Rank_l3 | Rank_l4 | Rank_l5 |
|---|---|---|---|---|---|---|---|
| | <chr> | <dbl> | <dbl> | <dbl> | <dbl> | <dbl> | <dbl> |
| 1 | sixteen | 3.045704e-04 | 3.626253e-05 | 3.814793e-05 | 3.555214e-05 | 1.420780e-05 | 2.965636e-05 |
| 2 | thousand | 2.069224e-04 | 1.450781e-04 | 3.841735e-06 | 6.495093e-06 | 5.684142e-05 | 1.186478e-04 |
| 3 | new | 1.679323e-04 | 8.192274e-05 | 3.717696e-08 | 6.827448e-05 | 5.127801e-06 | 6.748804e-05 |
| 4 | internal | 5.853603e-05 | 2.668079e-05 | 1.520667e-05 | 1.006819e-05 | 9.949040e-05 | 1.913858e-06 |
| 5 | revenue | 5.680332e-06 | 1.295419e-04 | 9.783306e-05 | 5.991195e-06 | 2.772595e-04 | 2.228007e-04 |
| 6 | service | 9.704161e-05 | 4.848936e-05 | 2.599927e-04 | 7.041479e-06 | 1.937593e-04 | 1.118323e-04 |

## Feature selection χ2

In statistics, the *χ2* test is applied to test the independence of two events. In feature selection, the two events are occurrence of the term and occurrence of the class.

We rank terms with respect to the following quantity:

$$X^2(\mathbb{D}, t, c) = \sum_{e_t \in \{0,1\}} \sum_{e_c \in \{0,1\}} \frac{(N_{e_t e_c} - E_{e_t e_c})^2}{E_{e_t e_c}}$$

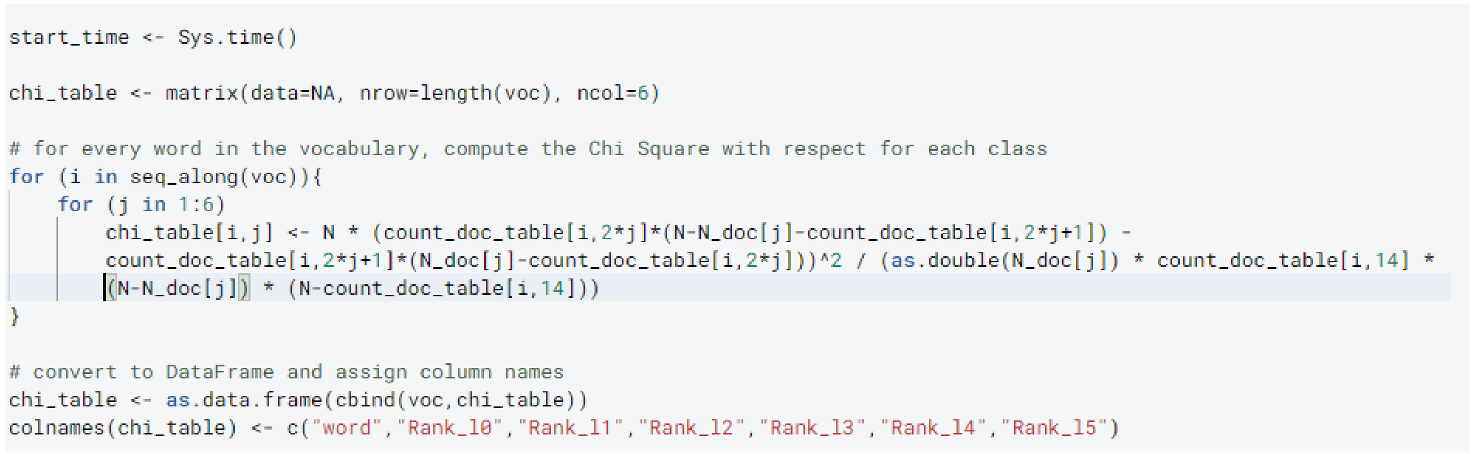
Where $N_{e_t e_c}$ is the observed frequency in the training set and $E_{e_t e_c}$ is the expected frequency.
For example,

$$E_{11} = N \times P(t) \times P(c)$$

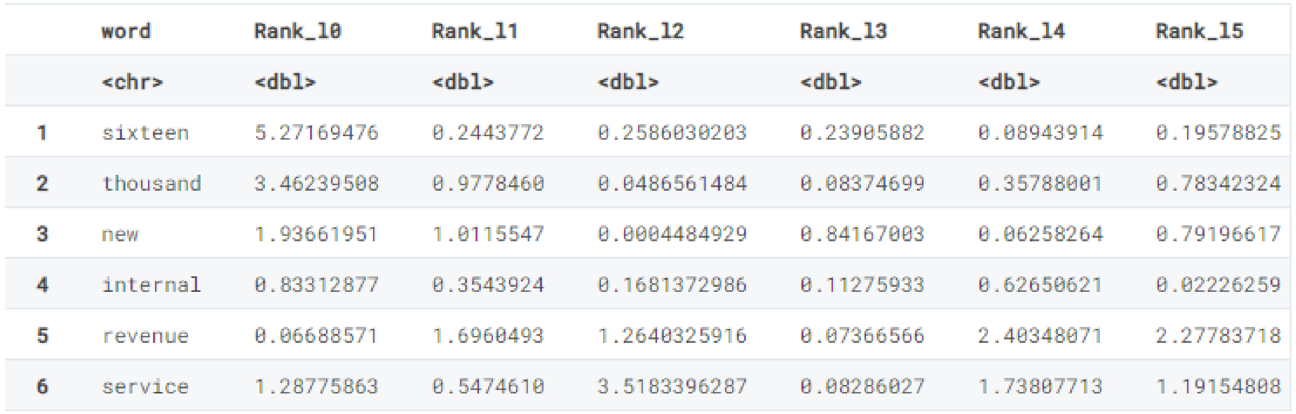
$E_{11}$ is the expected frequency of t and c occurring together in a document assuming that term and class are independent.

$X^2$ is a measure of how much expected counts and observed counts deviate from each other.

## Compute χ2

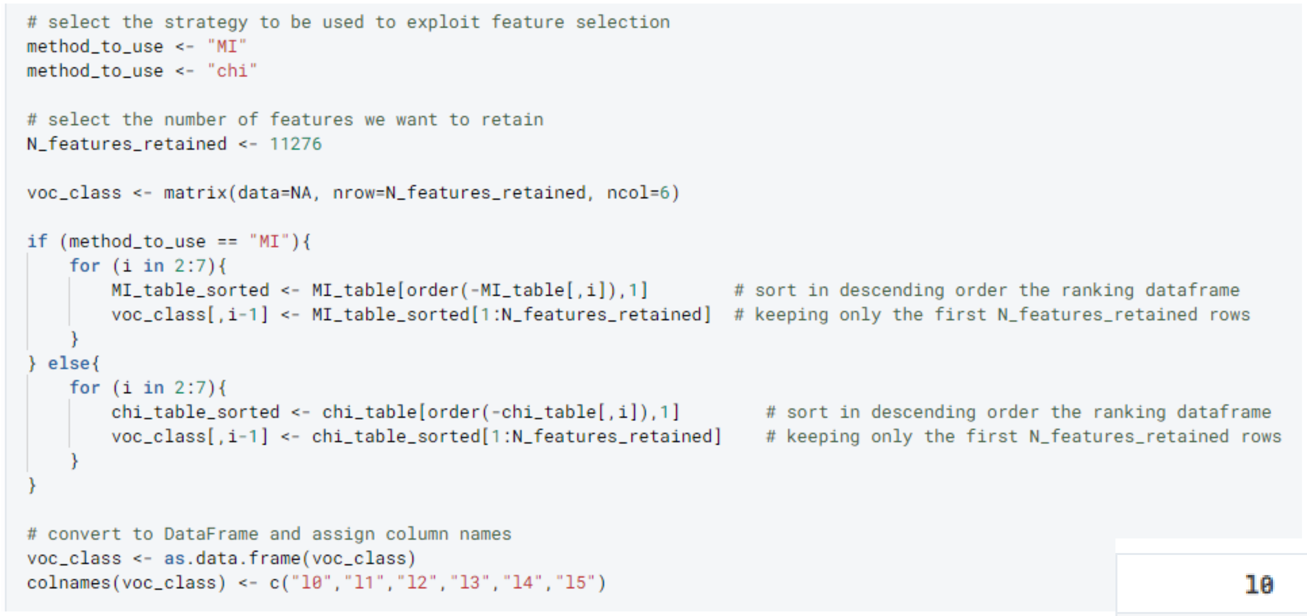


```r
start_time <- Sys.time()

chi_table <- matrix(data=NA, nrow=length(voc), ncol=6)

# for every word in the vocabulary, compute the Chi Square with respect for each class
for (i in seq_along(voc)){
    for (j in 1:6)
        chi_table[i,j] <- N * (count_doc_table[i,2*j]*(N-N_doc[j]-count_doc_table[i,2*j+1]) -
        count_doc_table[i,2*j+1]*(N_doc[j]-count_doc_table[i,2*j]))^2 / (as.double(N_doc[j]) * count_doc_table[i,14] *
        (N-N_doc[j]) * (N-count_doc_table[i,14]))
}

# convert to DataFrame and assign column names
chi_table <- as.data.frame(cbind(voc,chi_table))
colnames(chi_table) <- c("word","Rank_10","Rank_11","Rank_12","Rank_13","Rank_14","Rank_15")
```

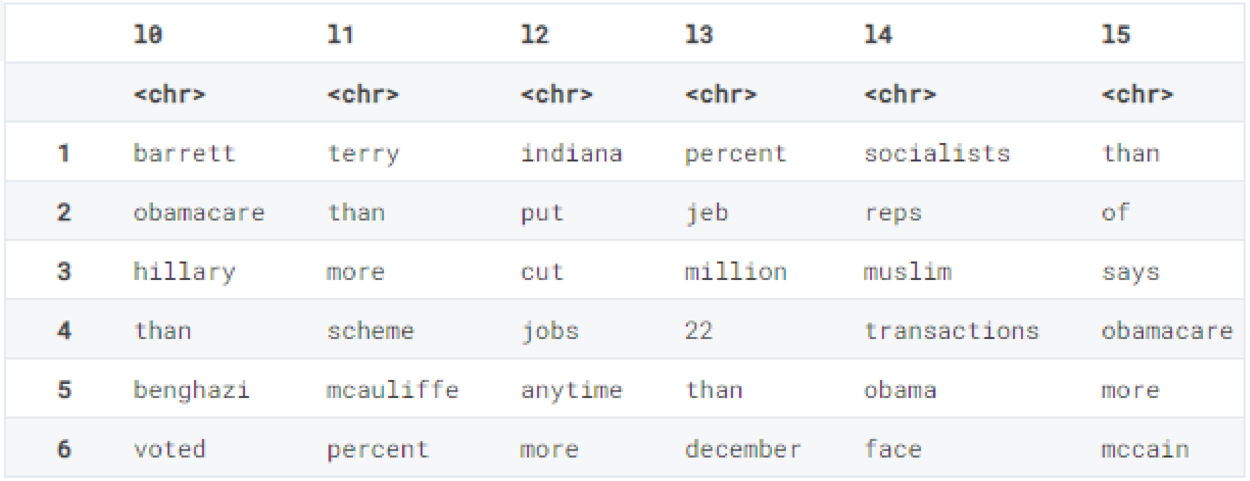


| | word | Rank_10 | Rank_11 | Rank_12 | Rank_13 | Rank_14 | Rank_15 |
|---|---|---|---|---|---|---|---|
| | <chr> | <dbl> | <dbl> | <dbl> | <dbl> | <dbl> | <dbl> |
| 1 | sixteen | 5.27169476 | 0.2443772 | 0.2586030203 | 0.23905882 | 0.08943914 | 0.19578825 |
| 2 | thousand | 3.46239508 | 0.9778460 | 0.0486561484 | 0.08374699 | 0.35788001 | 0.78342324 |
| 3 | new | 1.93661951 | 1.0115547 | 0.0004484929 | 0.84167003 | 0.06258264 | 0.79196617 |
| 4 | internal | 0.83312877 | 0.3543924 | 0.1681372986 | 0.11275933 | 0.62650621 | 0.02226259 |
| 5 | revenue | 0.06688571 | 1.6960493 | 1.2640325916 | 0.07366566 | 2.40348071 | 2.27783718 |
| 6 | service | 1.28775863 | 0.5474610 | 3.5183396287 | 0.08286027 | 1.73807713 | 1.19154808 |

# Extract a vocabulary for each class



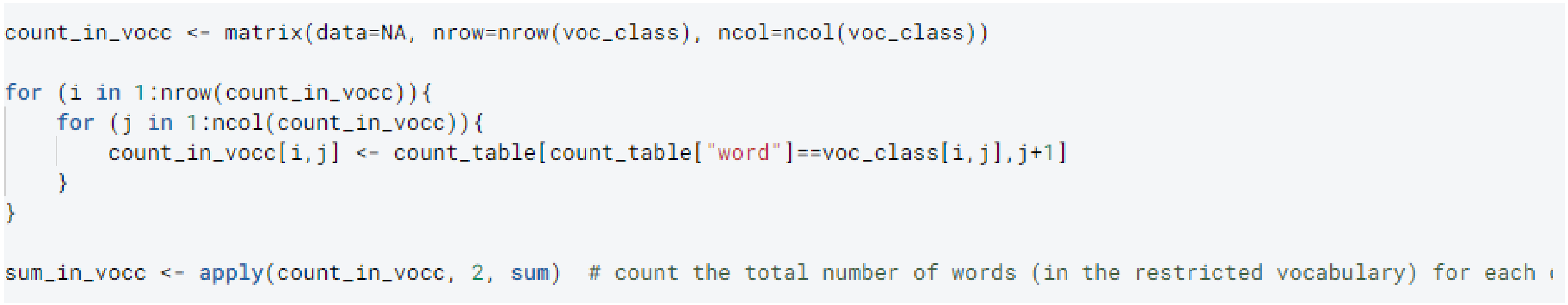
```
# select the strategy to be used to exploit feature selection
method_to_use <- "MI"
method_to_use <- "chi"

# select the number of features we want to retain
N_features_retained <- 11276

voc_class <- matrix(data=NA, nrow=N_features_retained, ncol=6)

if (method_to_use == "MI"){
    for (i in 2:7){
        MI_table_sorted <- MI_table[order(-MI_table[,i]),1]       # sort in descending order the ranking dataframe
        voc_class[,i-1] <- MI_table_sorted[1:N_features_retained]  # keeping only the first N_features_retained rows
    }
} else{
    for (i in 2:7){
        chi_table_sorted <- chi_table[order(-chi_table[,i]),1]       # sort in descending order the ranking dataframe
        voc_class[,i-1] <- chi_table_sorted[1:N_features_retained]   # keeping only the first N_features_retained rows
    }
}

# convert to DataFrame and assign column names
voc_class <- as.data.frame(voc_class)
colnames(voc_class) <- c("l0","l1","l2","l3","l4","l5")
```



| | l0 | l1 | l2 | l3 | l4 | l5 |
|---|---|---|---|---|---|---|
| | <chr> | <chr> | <chr> | <chr> | <chr> | <chr> |
| 1 | barrett | terry | indiana | percent | socialists | than |
| 2 | obamacare | than | put | jeb | reps | of |
| 3 | hillary | more | cut | million | muslim | says |
| 4 | than | scheme | jobs | 22 | transactions | obamacare |
| 5 | benghazi | mcauliffe | anytime | than | obama | more |
| 6 | voted | percent | more | december | face | mccain |

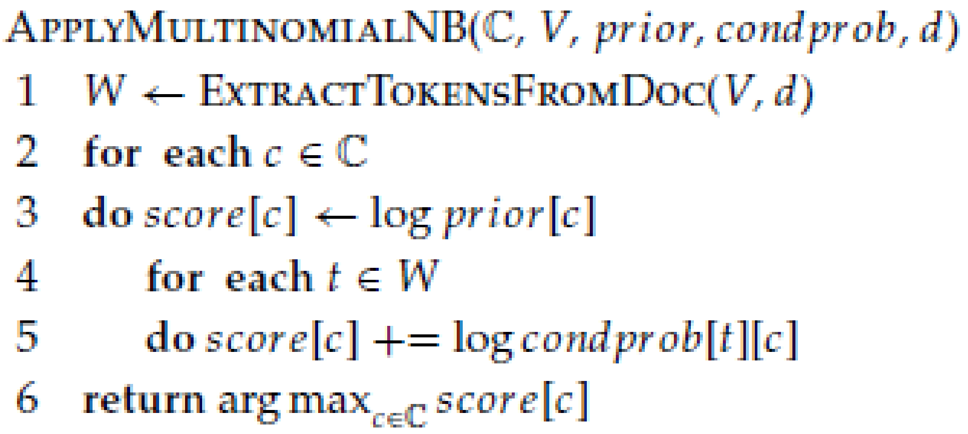
# Test the algorithm

- "count_in_vocc" matrix is a restriction of "count_doc_table" only to the words retained as features

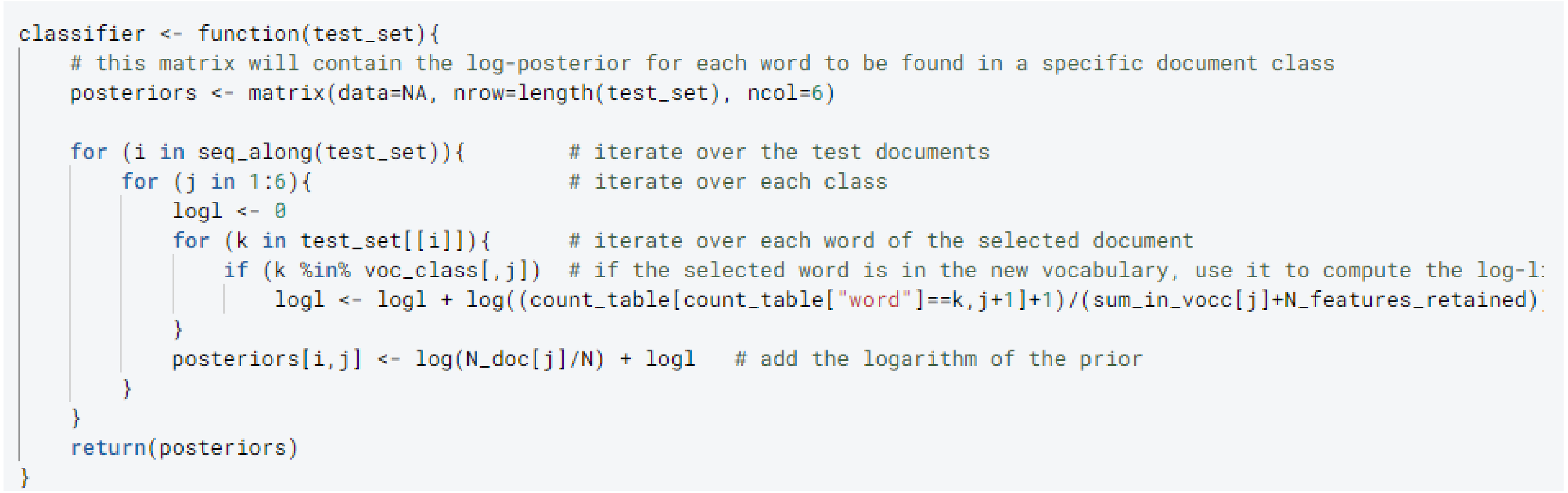


```r
count_in_vocc <- matrix(data=NA, nrow=nrow(voc_class), ncol=ncol(voc_class))

for (i in 1:nrow(count_in_vocc)){
    for (j in 1:ncol(count_in_vocc)){
        count_in_vocc[i,j] <- count_table[count_table["word"]==voc_class[i,j],j+1]
    }
}

sum_in_vocc <- apply(count_in_vocc, 2, sum)  # count the total number of words (in the restricted vocabulary) for each
```

# Multinomial Naive Bayes algorithm: Classifier

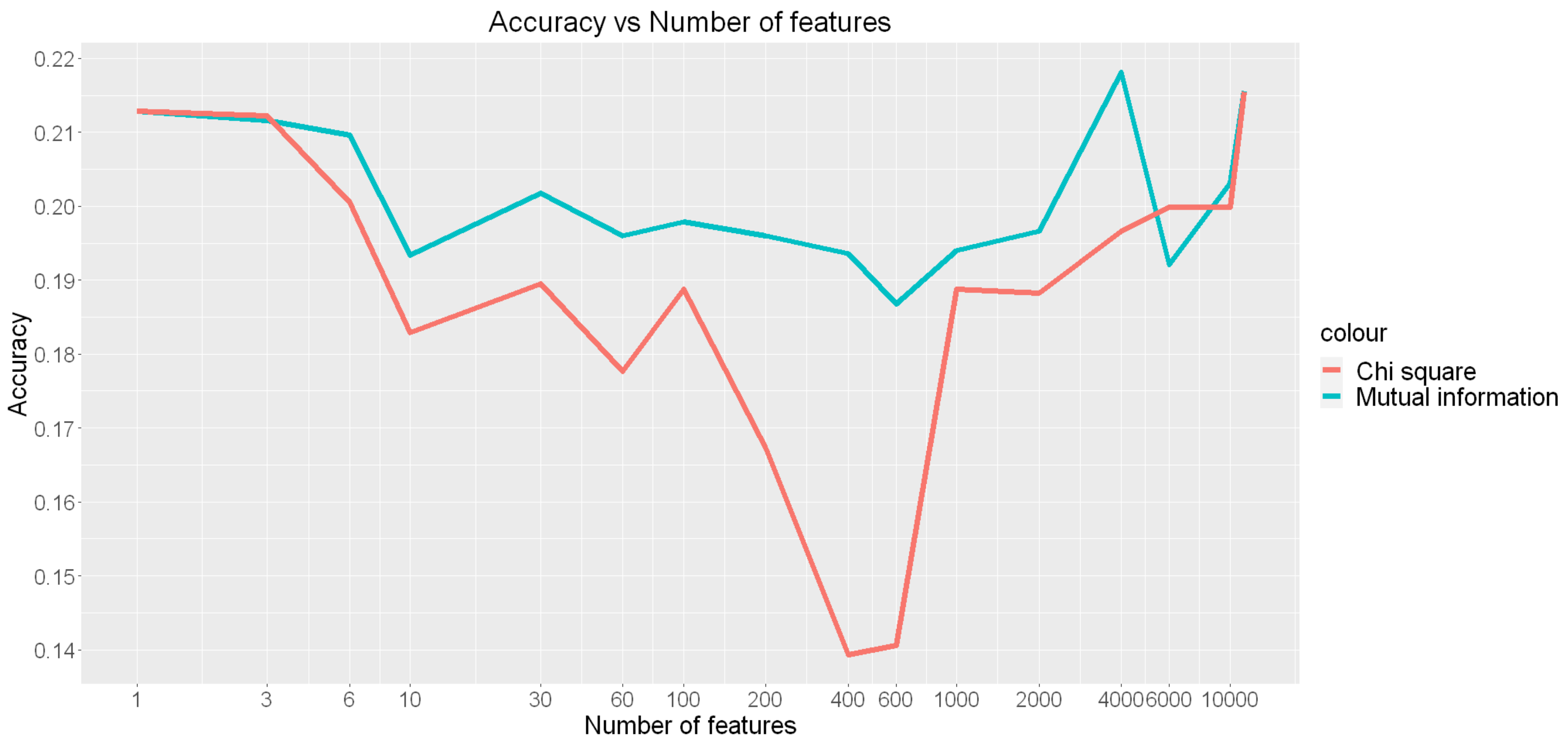


```
ApplyMultinomialNB($\mathbb{C}$, $V$, $prior$, $condprob$, $d$)
1  $W \leftarrow$ ExtractTokensFromDoc($V$, $d$)
2  for each $c \in \mathbb{C}$
3  do $score[c] \leftarrow \log prior[c]$
4      for each $t \in W$
5      do $score[c] \mathrel{+}= \log condprob[t][c]$
6  return $\arg\max_{c \in \mathbb{C}} score[c]$
```

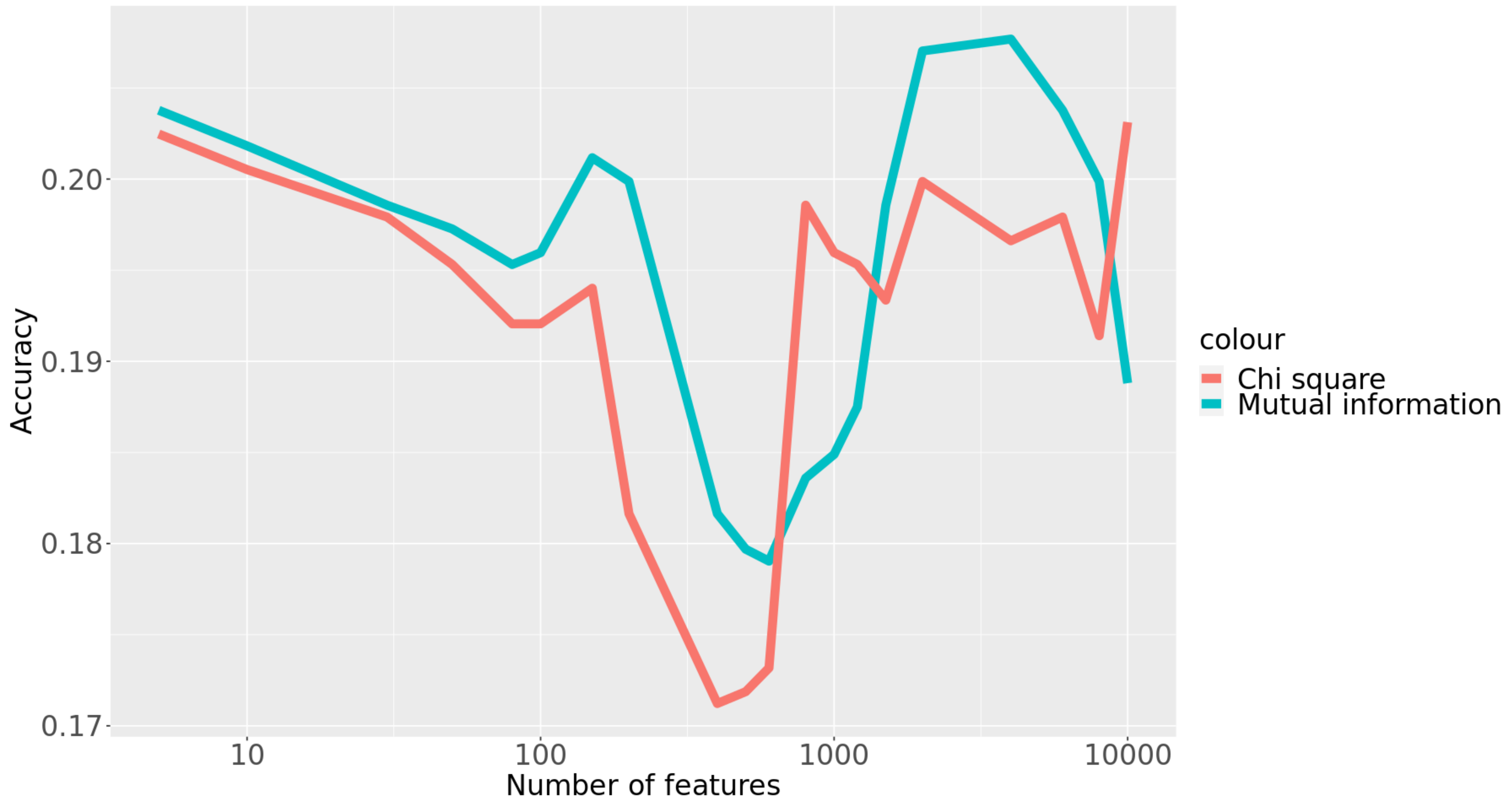


```
classifier <- function(test_set){
    # this matrix will contain the log-posterior for each word to be found in a specific document class
    posteriors <- matrix(data=NA, nrow=length(test_set), ncol=6)

    for (i in seq_along(test_set)){        # iterate over the test documents
        for (j in 1:6){                    # iterate over each class
            logl <- 0
            for (k in test_set[[i]]){      # iterate over each word of the selected document
                if (k %in% voc_class[,j])  # if the selected word is in the new vocabulary, use it to compute the log-l
                    logl <- logl + log((count_table[count_table["word"]==k,j+1]+1)/(sum_in_vocc[j]+N_features_retained)
            }
            posteriors[i,j] <- log(N_doc[j]/N) + logl    # add the logarithm of the prior
        }
    }
    return(posteriors)
}
```

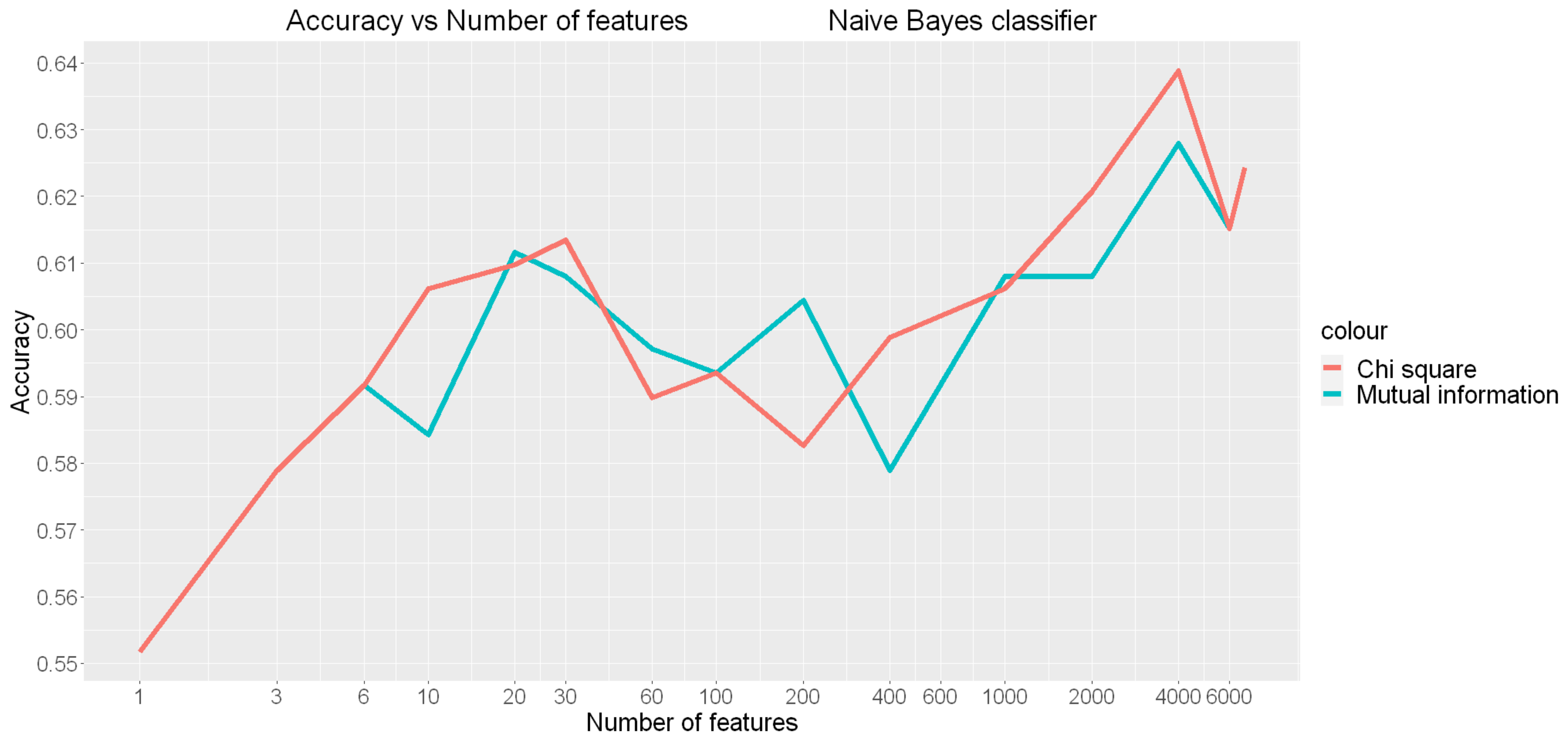
# Accuracy plot



## Accuracy plot without stopwords



# Accuracy plot – Two-classes Naive Bayes classifier



*Thank you for your attention*

UNIVERSITÀ DEGLI STUDI DI PADOVA