
# MET CS688 C1

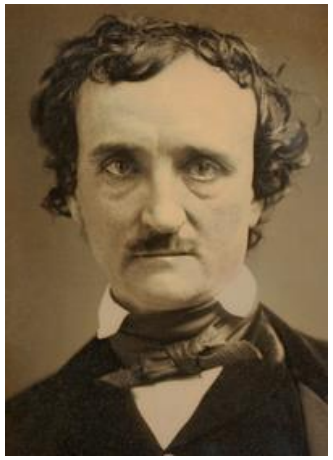
# *WEB ANALYTICS AND MINING*

ZLATKO VASILKOSKI

## ZIPF'S LAW

# Edgar Allan Poe “The gold bug”

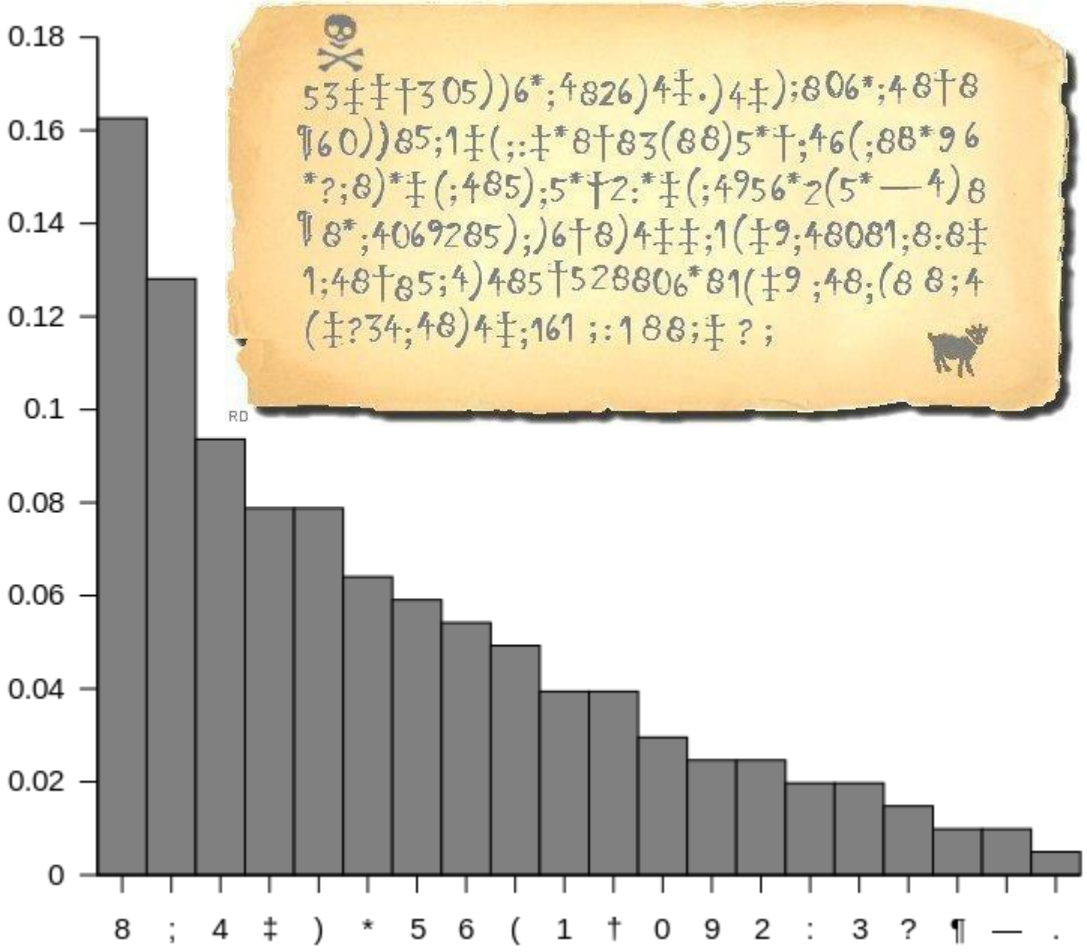
The Gold-Bug" is a short story by Edgar Allan Poe published in 1843. https://en.wikipedia.org/wiki/The_Gold-Bug



Edgar Allan Poe (1809 – 1849)

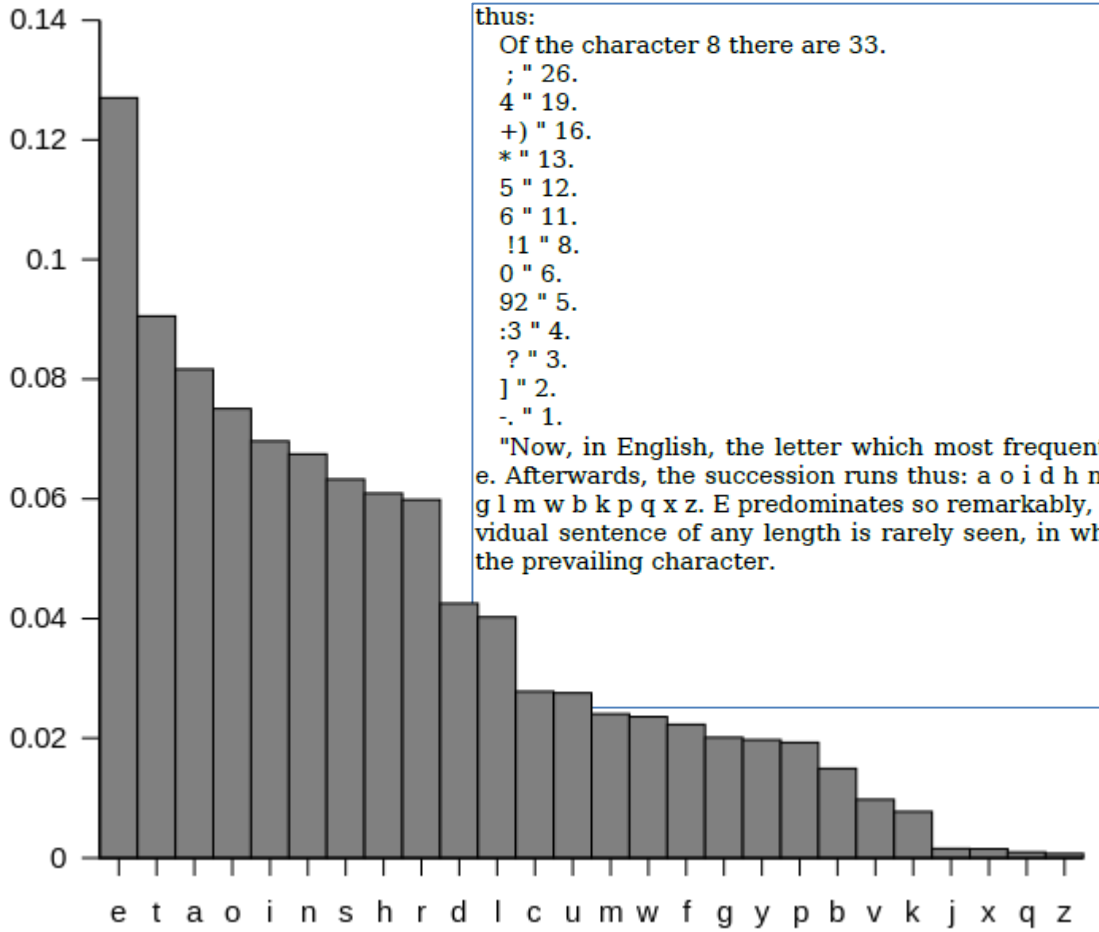
- Though he did not invent cryptography Poe certainly popularized it during his time.
- He was probably inspired by an interest in Daniel Defoe's Robinson Crusoe.
- His short story “The gold Bug” contains a great deal of text mining techniques still used today.

# The cryptography of the “The gold bug”










thus:
Of the character 8 there are 33.
; " 26.
4 " 19.
+) " 16.
* " 13.
5 " 12.
6 " 11.
!1 " 8.
0 " 6.
92 " 5.
:3 " 4.
? " 3.
] " 2.
-. " 1.
"Now, in English, the letter which most frequently occurs is e. Afterwards, the succession runs thus: a o i d h n r s t u y c f g l m w b k p q x z. E predominates so remarkably, that an individual sentence of any length is rarely seen, in which it is not the prevailing character.

30

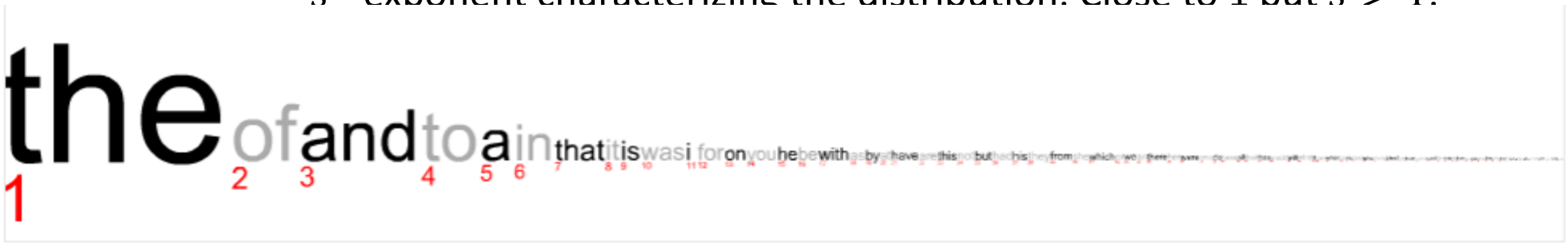
The Gold-Bug’s "secret writing" includes a cipher that uses a simple substitution cipher.

# Zipf's law - How terms are distributed across documents.

- Zipf's law is an observation that the most frequent (English) word will occur approximately twice as often as the second most frequent word, three times as often as the third most frequent word, etc.
- Named after George Zipf (1902–1950), Harvard University linguist.
- An empirical law formulated (1935) by studying data in the physical and social sciences using mathematical statistics.

$$f_k \sim 1/k^s$$

- $f_k$ - Frequency of a word ranked $k^{\text{th}}$. The word “and” is ranked third so k=3
- $s$ - exponent characterizing the distribution. Close to 1 but $s > 1$.



# Zipf's law links

Familiarize yourself with Zipf's law by visiting:

https://en.wikipedia.org/wiki/Zipf%27s_law

Look at these following links:

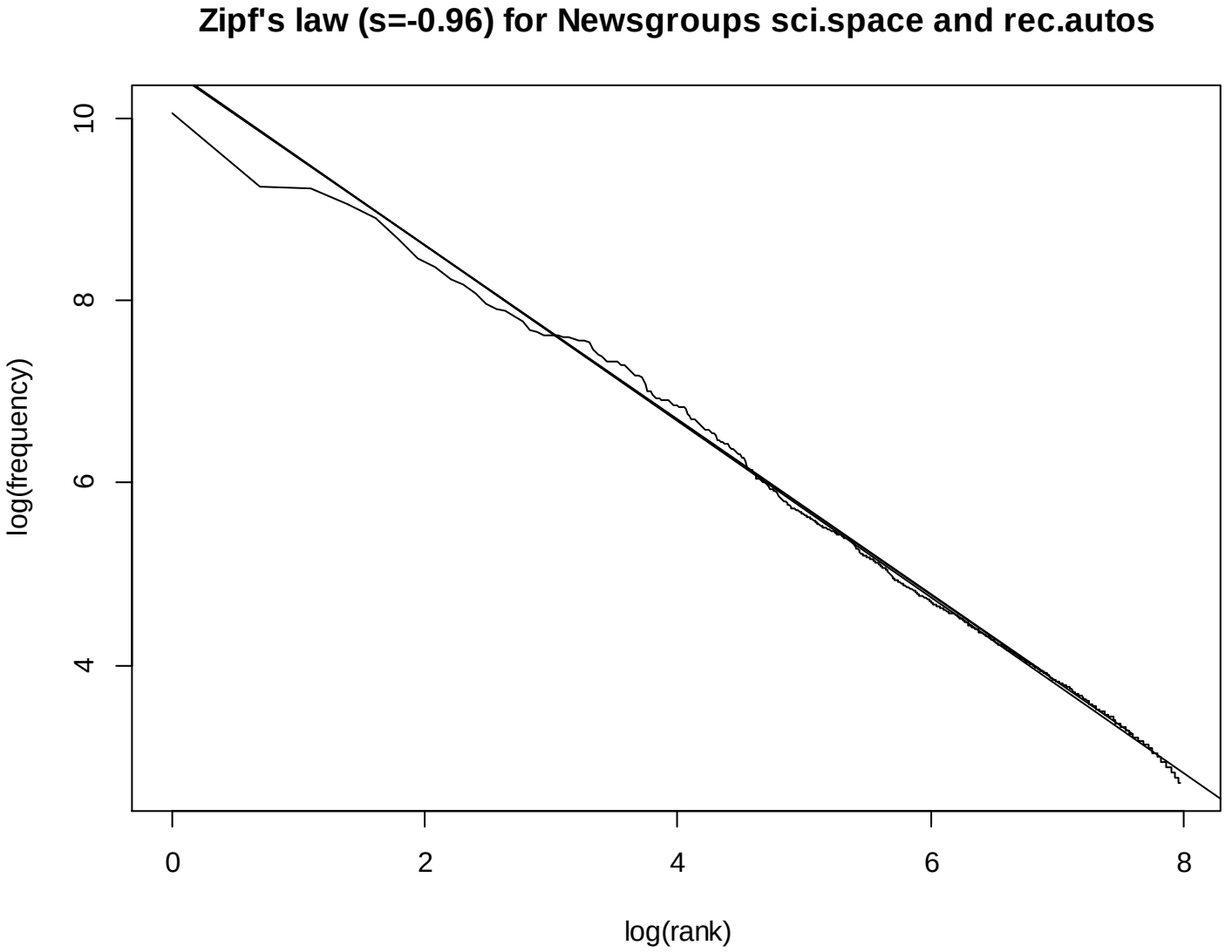
https://www.youtube.com/watch?v=fCn8zs912OE

http://www.wordcount.org/

**Zipf's Law**: $f=1/k^s$

The frequency ***f*** of any word is inversely proportional to its rank ***k*** in the frequency table, (with some exponent ***s***).

All the files in the newsgroup test and train folders were used. Word length of more than 2 and frequency 15 were used.



# Lab Project: Confirm Zipf's law

Task: Find out how much Zipf's law holds for the frequency words of newsgroup text

1. Create a Corpus from all of the text for the 2 newsgroups you used in the Newsgroops assignment. (alternatively use the code on Blackboard)
2. Implement just the following preprocessing:
   - Remove Whitespace
   - Convert to Lowercase
   - Remove Punctuation
   - Remove Numbers
3. Create a document term matrix and find the 20 most frequent words.
   - Find their frequency
   - Find their frequency %
4. Create a linear model from the data and find what is Zipf's exponent for this dataset.
   - Familiarize yourself with the R fitting function "lm()" from "stats" package. Use lm(Y ~ X , data)
   - Take X to be the natural logarithm of the Frequency
   - Take Y to be the natural logarithm of the Rank (the order from most to list popular words)
5. Plot Percentage of all words vs unique words.
   - What is the percentage of single words
6. Check is half of ANY text is 50 to 100 same words.
   - Plot Cumulative Percent vs the 100 most frequent words
7. Submit your R script code.

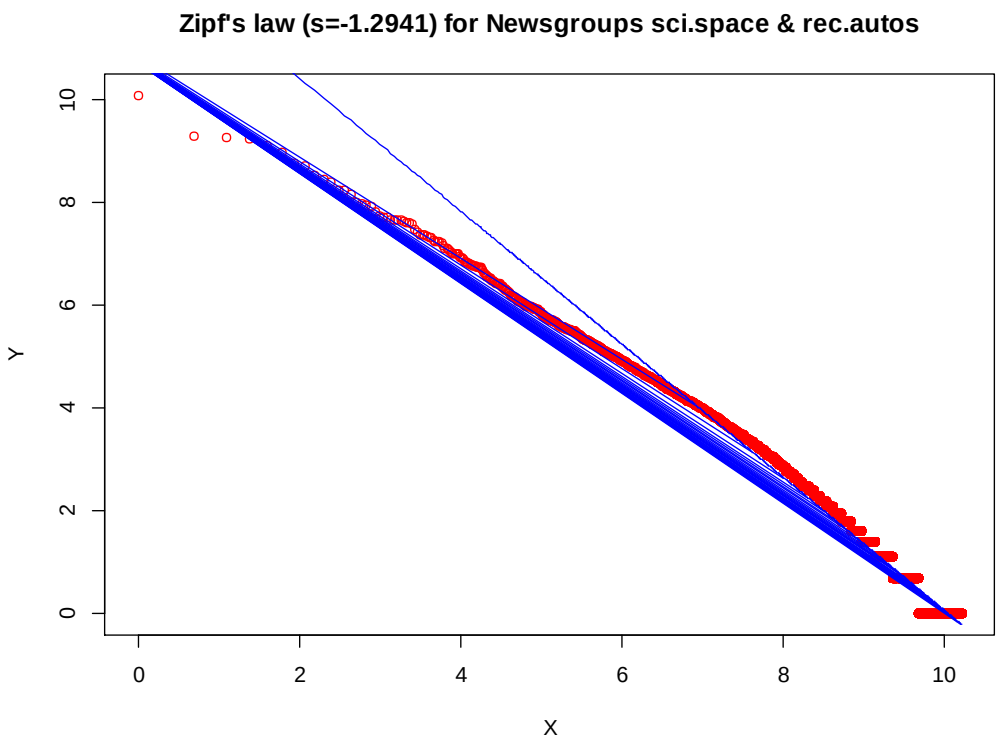
Predictive Task (do it only if you find it challenging):

1. Find rank of particular word ("military") from http://www.wordcount.org/ and predict the frequency of that word in your corpus,

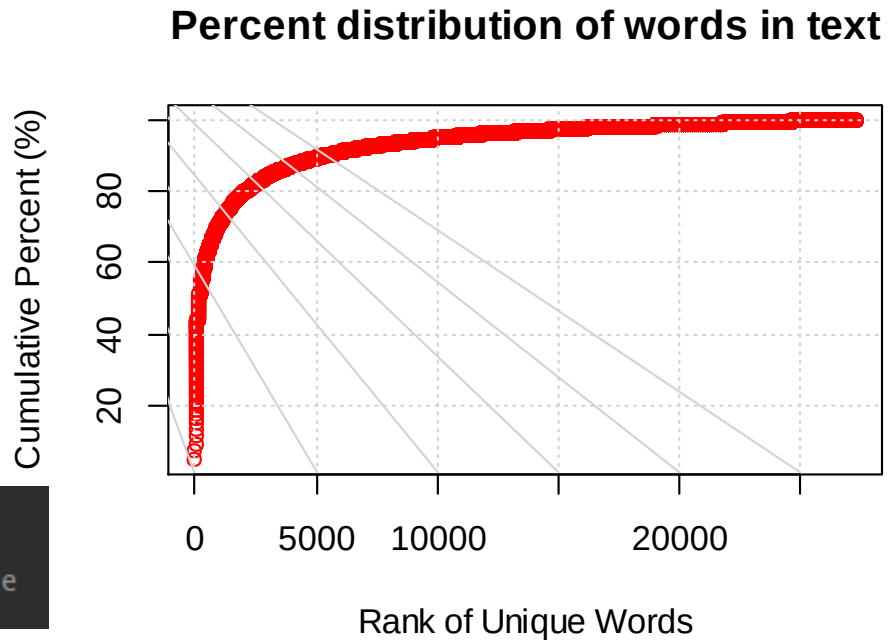
# Hints Lab Project: Confirm Zipf's law

```r
# 3) Create a document term matrix and find the 20 most frequent words.
dtm <- as.matrix( DocumentTermMatrix(Doc.Corpus, control=list(wordLengths=c(1,Inf), bounds=list(global=c(1,Inf))) ))
freq <- colSums(dtm) # Term frequencies
ord <- order(freq,decreasing = TRUE) # Indices of frequencies order
frequency <- freq[ord] # Ordered Freq.
Freq.Percent <- frequency/sum(frequency)*100
```

```r
# Zipf's Variables
TotalNum_Words <- dim(dtm)[2]
rank <- 1:TotalNum_Words
x <- log(rank) # Natural Logarithm of the Rank
y <- log(frequency) # Natural Logarithm of the Frequency

data <- data.frame(X=x,Y=y)
plot(data, col = "red")

# Fitting
# linear model ==============================================
model <- lm(Y ~ X , data) # Create a linear model from the Data
model$coefficients
s <- round(model$coefficients[2],4)
predictedY <- predict(model, data) # Regression - predicted values from the model
lines(data$X, predictedY, col = "blue", pch=4)

Title <- paste0("Zipf's law (s=",s,") for Newsgroups ",newsgroup1," & ",newsgroup2)
title(Title) # Display Title

# Using tm package
Zipf_plot(dtm,main=Title, type = "p", col = "red")
# (Intercept)           x
# 13.008821   -1.294067
```

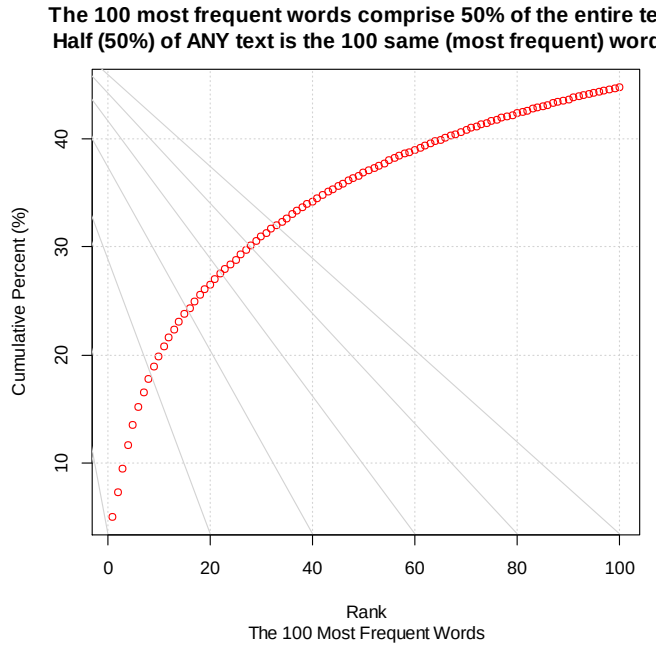


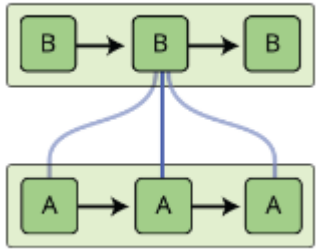
```r
# 5) Plot Peercentage of all words vs unique words
yy <- cumsum(Freq.Percent) # Cumulative Percentage
xx <- rank

data1 <- data.frame(X=xx,Y=yy)
plot(data1, col = "red", xlab = "Rank of Unique Words", ylab = "Cumulative Percent (%)")
grid(nx = NULL, ny = NULL, col = "lightgray", lty = "dotted")
Title <- "Percent distribution of words in text"
title(Title) # Display Title
```




```r
# 5a) What is the Percentage of Single words
ix <- which(frequency<2) # Indices of Single Words
Single.Words <- sum(frequency[ix])/TotalNum_Words*100 # Percent of Words appearing ony once
```

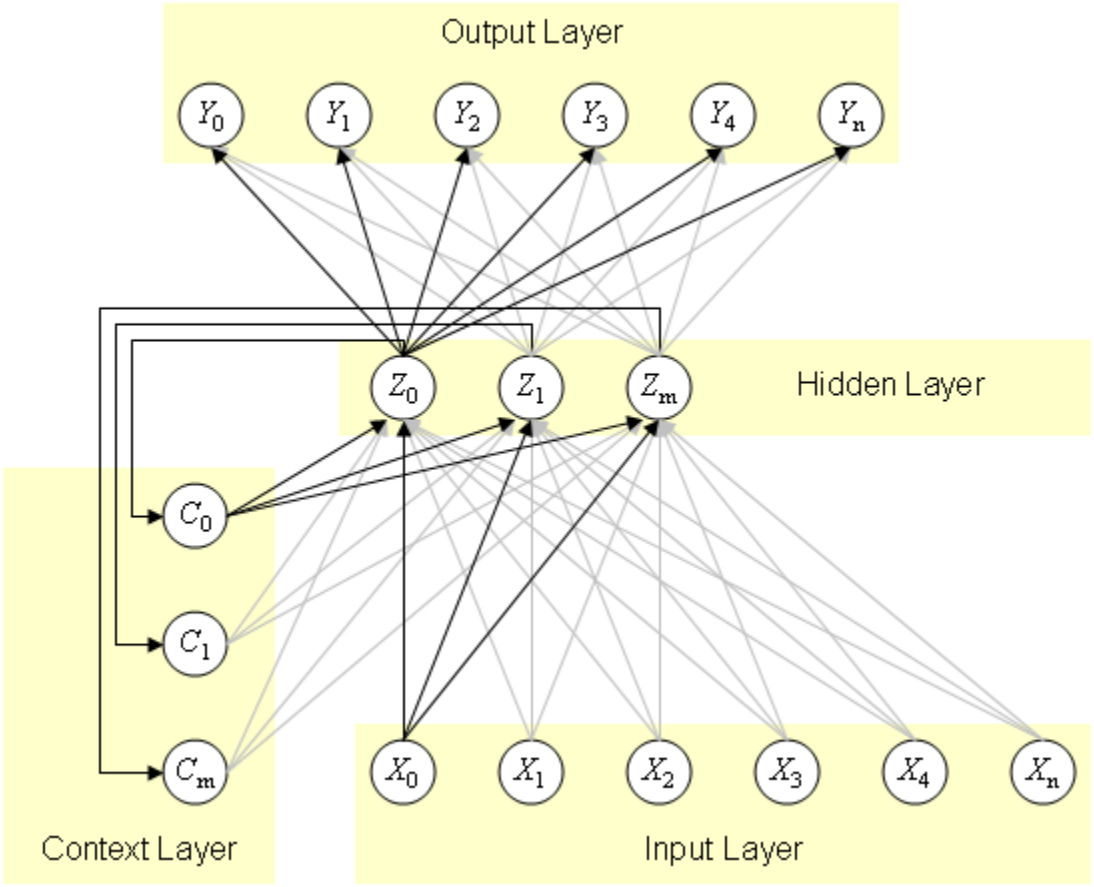
```r
# 6) Check is Half of ANY text is 50 to 100 same words
Title1 <- "The 100 most frequent words comprise 50% of the entire text!" # Title
Title2 <- "Half (50%) of ANY text is the 100 same (most frequent) words!" # Sub Title
Title <- paste(Title1,Title2,sep = "\n")
plot(data1[1:100,], col = "red", xlab = "Rank", ylab = 'Cumulative Percent (%)')
grid(nx = NULL, ny = NULL, col = "lightgray", lty = "dotted")
title(main = Title, sub = "The 100 Most Frequent Words") # Display Title
```

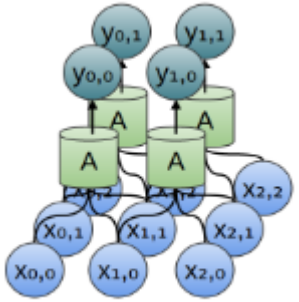





# Recurrent Neural Networks

- Attention and Augmented Recurrent Neural Networks work with sequences of data like text, audio and video.
- Contain Context layer
- A special variant – “long short-term memory” LSTM networks
- Very powerful, remarkable results in many tasks including translation, voice recognition, and image captioning.






# Convolutional Neural Nets

- Feed-forward neural network, uses multilayer perceptrons that require minimal preprocessing, great for image classification.
- Uses many identical copies of the same.
- Analogous to the abstraction of functions in mathematics and computer science.
- Allows the network to have lots of neurons and express computationally large models while keeping the number of actual parameters – the values describing how neurons behave – that need to be learned fairly small.

# Neural Nets in R – RSNNS package

- Enables a use of wide variety of network types and possible applications.
- A convenient feature in R is the inclusion of datasets along with software packages.
- This is important for NNS standardized tests as well as for examples of the usage of the package.
- Use the list *snnsData()* for all of the available datasets in RSNNS.

- These functions can be used to pick the right columns according to their names.
  - *inputColumns()* Extracts all columns from a matrix whose column names begin with "in"
  - *outputColumns()* Extracts all columns from a matrix whose column names begin with "out"
- Functions *splitForTrainingAndTest()* can be used to split the data in a training and a test set.

# RSNNS – Access & Splitting the Data

| | Dat Table | |
|---|---|---|
| Inputs | | Outouts |
| 1 | | 1 |
| 2 | Train | 2 |
| - | Data | - |
| - | | - |
| 850 | | 850 |
| 851 | | 851 |
| - | Test | - |
| - | Data | - |
| 1000 | | 1000 |

- Access and split the data into Train and Test data – Note how the data is split.

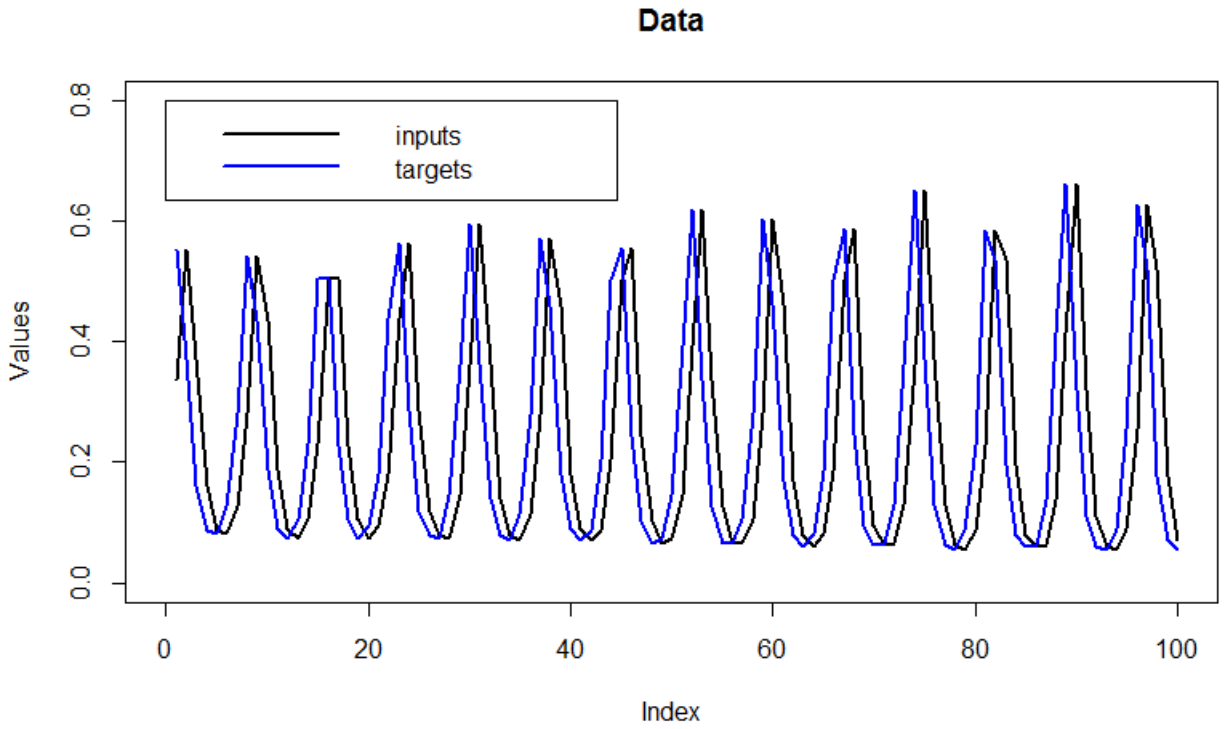
```
# Example: RSNNS Loading data
library("RSNNS")
# Load the Data (more data @  http://sci2s.ugr.es/keel/datasets.php)
data("snnsData")
laser <- snnsData$laser_1000.pat

# Spliting the Data
inputs <- laser[,inputColumns(laser)] # extract all col. names begin with "in"
targets <- laser[,outputColumns(laser)] # extract all col with “out"
# Split the input data (1000) into train (850) and test (150) specified
# by the argument ratio=0.15
patterns <- splitForTrainingAndTest(inputs, targets, ratio = 0.15)
```

```
> laser[1:5,]
             in1      out1
pattern1 0.337255 0.552941
pattern2 0.552941 0.372549
pattern3 0.372549 0.160784
pattern4 0.160784 0.086275
pattern5 0.086275 0.082353
> inputs[1:5]
pattern1 pattern2 pattern3 pattern4 pattern5
0.337255 0.552941 0.372549 0.160784 0.086275
> targets[1:5]
pattern1 pattern2 pattern3 pattern4 pattern5
0.552941 0.372549 0.160784 0.086275 0.082353
>
```

```
> length(patterns$inputsTrain)
[1] 850
> patterns$inputsTrain[1:5]
pattern1 pattern2 pattern3 pattern4 pattern5
0.337255 0.552941 0.372549 0.160784 0.086275
> patterns$targetsTrain[1:5]
pattern1 pattern2 pattern3 pattern4 pattern5
0.552941 0.372549 0.160784 0.086275 0.082353
>
```

```
> length(patterns$inputsTest)
[1] 150
> patterns$inputsTest[1:5]
pattern851 pattern852 pattern853 pattern854 pattern855
  0.109804   0.090196   0.125490   0.254902   0.482353
> patterns$targetsTest[1:5]
pattern851 pattern852 pattern853 pattern854 pattern855
  0.090196   0.125490   0.254902   0.482353   0.454902
>
```

# RSNNS –Data Overview



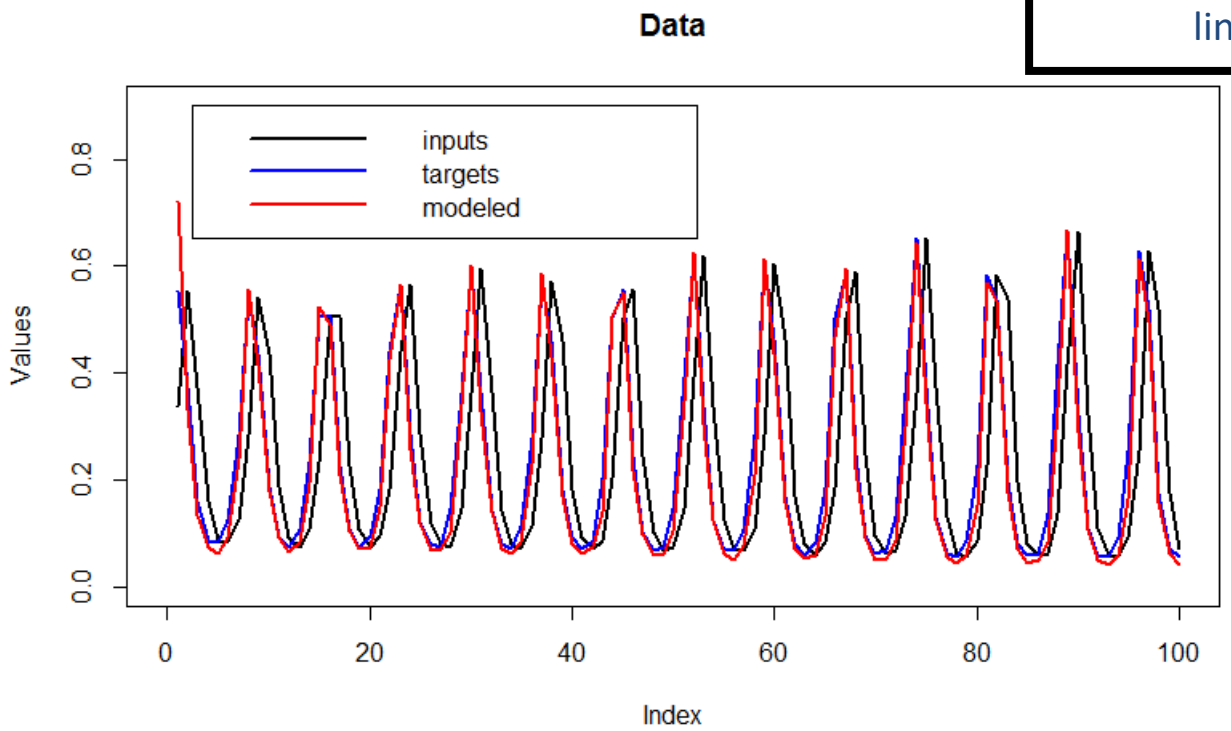

R is a powerful tools for data visualization.
Very useful visualizing the data used in neural net modeling.

Note how the target data lags after the input data.

# RSNNS - 1. Regression

- Illustration of how to use an Elman network (Elman 1990) for time series regression.
- The result (fitted values) is stored in *model$fitted.values* – same size as *inputs* and *targets*.
- Note how the fitted values (red) match the target values (blue)

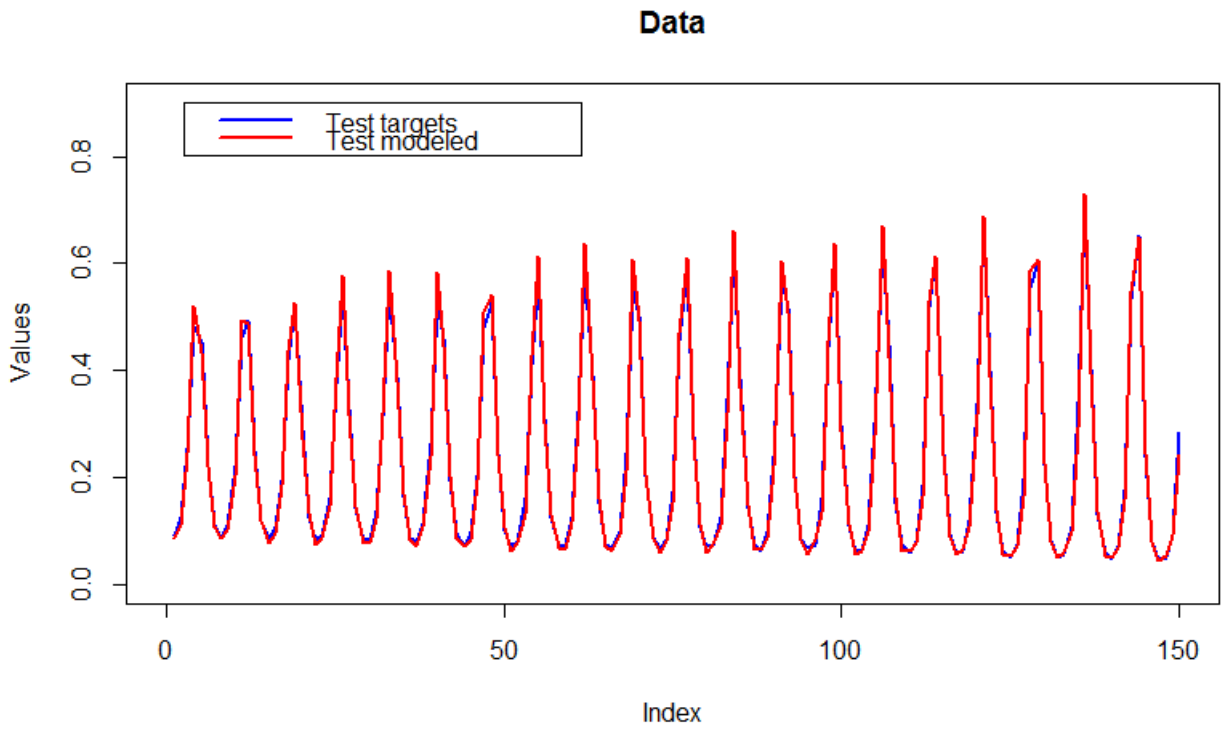
```
# Example: RSNNS Prediction using Elman network
model <- elman(patterns$inputsTrain, patterns$targetsTrain,
               size = c(8, 8), learnFuncParams = c(0.1), maxit = 500,
               inputsTest = patterns$inputsTest, targetsTest = patterns$targetsTest,
               linOut = FALSE)
```




```
> length(model$fitted.values)
[1] 850
> model$fitted.values[1:5]
[1] 0.72094250 0.33228073 0.13483816 0.07415414 0.06115905
>
```

# RSNNS - 1. Regression

- Test data vs. test fitted values (indices 851 to 1000)



# RSNNS - 1. Regression

Error plots

```
# Example: RSNNS Elman network – Plot Errors
plotRegressionError(patterns$targetsTrain, model$fitted.values)
plotRegressionError(patterns$targetsTest, model$fittedTestValues)
```

Regression plots

- For the **training data** - showing an optimal linear fit (black line), and linear fit (red line) to the **training data** (dots).
- Same for the **test data**. Note how the error for the test data (indices 851 to 1000) is larger.