**VILNIUS UNIVERSITY**

**FACULTY OF MATHEMATICS AND INFORMATICS**

**Article**

# Ridge and Lasso logistic regression in natural language processing

Eligijus Bujokas

**VILNIUS (2018.12.18)**

# Logistic Ridge and Lasso regression in NLP


## Abstract

The aim of this document is to give a brief introduction to natural language processing (NLP), apply Ridge, Lasso and elastic net binary logistic regression to an NLP related problem and to write a reproducible code.

The calculations and visuals are made using the statistical software **R**. The main framework for text preprocessing is the **text2vec** (Selivanov and Wang, 2018) package.

The logistic regression is implemented using the package **glmnet** (Simon, Friedman, Hastie, and Tibshirani, 2011).



**Key words :** Logistic regression, Ridge, Lasso, Elastic Net, NLP, machine learning, tfidf

# Contents

# Introduction

There are many classification tasks that revolve around classifying textual data. There are algorithms that based on a user written string can talk with a person, after reading lyrics of a song it can label the genre of the song and so on. The field of computer science that deals with text which comes from users (humans) is called natural language processing.

Natural language processing (NLP) is a sub field of computer science concerned with the interactions between computers and human (natural) languages, in particular how to program computers to process and analyze large amounts of natural language data.

One part of NLP tasks is to deal with binary responses based on text. A computer must determine whether a written text has a positive or negative sentiment, whether this is hate speech or not and so on. It is very time consuming (and almost impossible) for humans to cover the vast amount of textual data and give it a label thus computer scientists implemented binary logistic regression in various NLP frameworks to deal with the labeling problem. Instead of going through millions of documents, a group of people need to label only a couple of thousand of documents and have the computer (machine) fit the binary logistic model (learn) and fit the model to other texts.

## Introductory example

Let us assume that our $\mathbb{Y}$ variable is binary. Each class of $\mathbb{Y}$ could indicate whether a string is 'positive' or 'negative' in a semantic sense, 'sincere' or 'not sincere' and so on. Each column in the $\mathbb{X}$ matrix are also binary. Each column represent a unique word and each row value represent whether a word was observed in a given string or not.

For example, let us say we have a couple of strings regarding a car review and some regarding other news:

| text | is_car_review |
|---|---|
| the new car | 1 |
| opel is good automobile | 1 |
| the weather is dreadful | 0 |
| the stocks are rising | 0 |

The matrix that is used for computations is often called the document term matrix (dtm for short). The dtm of our raw document would look like this:

| are | stocks | car | weather | automobile | rising | new | opel | dreadful | good | is | the |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |

Intuitively, we would like the words 'car' and 'automobile' to indicate a review being from a car review and the words 'weather', 'stocks' and others to indicate a non car related review.

Binary logistic regression is a method that can assign each feature (word in our case) a positive or a negative weight depending on the response variable (variable **is car review** in our case).

# NLP definitions and techniques

## Document corpus

A text corpus is a collection of texts of written (or spoken) language presented in electronic form. It is usually denoted as $\mathbb{D}$. Typically, the corpus is the vector containing the text data. In the introduction chapter, the text corpus would be ['the new car', 'opel is good automobile', 'the weather is dreadful', 'the stocks are rising'].

Each document are made of several tokens.

## Tokenization

Electronic text is a linear sequence of symbols (characters or words or phrases). Naturally, before any real text processing is to be done, text needs to be segmented into linguistic units such as words, punctuation, numbers, alpha-numeric, etc. This process is called tokenization (Trim, 2013).

In English, words are often separated from each other by blanks (white space), but not all white space is equal. Both 'Los Angeles' and 'rock 'n' roll' are individual thoughts despite the fact that they contain multiple words and spaces. We may also need to separate single words like 'I'm' into separate words 'I' and 'am'.

Tokenization is a kind of pre-processing in a sense; an identification of basic units to be processed. It is conventional to concentrate on pure analysis or generation while taking basic units for granted. Yet without these basic units clearly segregated it is impossible to carry out any analysis or generation.

For example, the string 'I'm a from the city of Vilnius' can be tokenized into several tokens: 'i', 'am', 'from', 'the', 'city', 'of', 'vilnius'.

Each term (token) in a given corpus is denoted as $t$.

## N - grams

An n-gram is a contiguous sequence of n items from a given sequence of text (Jurafsky and Martin, 2018 , pp. 37). Given a sentence, we can construct a list of n-grams from the sentence by finding pairs of words that occur next to each other. For example, given the sentence 'my name is Eligijus' you can construct bigrams (n-grams of length 2) by finding consecutive pairs of words: 'my name', 'name is' and 'is Eligijus'. Each of these bigrams would be considered a token and would be considered as a separate feature in the $\mathbb{X}$ matrix.

## Tf-idf tranformation

Tf-idf, short for term frequency-inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. The tf-idf is the product of two statistics, term frequency and inverse document frequency.

### Term frequency

$tf()$ is a function of two arguments - the term t and document d. If we denote the term frequency in a document as $f_{t,d}$ then

$$tf(t, d) = f_{t,d}$$

In simpler terms, the function tf() takes every unique token in our corpus and counts how many times it appeared in a document.

### Inverse document frequency

The inverse document frequency ($idf()$) is a measure of how much information the word provides, in other words, if it is common or rare across all documents.

$$idf(t, D) = log\left(\frac{N}{1 + |\{d \in D : t \in d\}|}\right)$$

N - total number of documents in our corpus. The corpus in the introduction chapter has two documents.

$|\{d \in D : t \in d\}|$ number of documents where the term t appears. We add 1 because if no document has a certain term, then we would be dividing by zero.

**Tf - idf calculation**

The tf-idf transformation for each term is a product of term frequency and inverse document frequency.

$$tfidf(t, d, D) = tf(t, d)\, idf(t, D)$$

Recall our example case. Lets us calculate the tfidf statistic for the word 'car'.

Term frequency:

$tf('car', d_1) = 1$

$tf('car', d_2) = 0$

$tf('car', d_3) = 0$

$tf('car', d_4) = 0$

Inverse document frequency:

$idf('car', D) = log(\frac{4}{2}) = 0.69$

Term frequency - inverse document frequency:

$tfidf('car', d_1, D) = 0.69$

$tfidf('car', d_2, D) = 0$

$tfidf('car', d_3, D) = 0$

$tfidf('car', d_4, D) = 0$

The transformation would yield that in the first document the term 'car' is equal to 0.69 and 0 in others. After transforming our dtm using this transformation we would diminish the weights of words that appear in every document and enhance the weights of specific words that are only in few documents.

Additionally, we would not have to artificially drop words that are very common because the more times a term appears in documents, the less important it will be, because the dtm value for that feature will be close to 0.

# Logistic regression for textual data

## General case for binary dependant variable

Let us assume that our data is the set:

$$D_{\mathbb{Y},\mathbb{X}} = \{\mathbb{Y}_i \in \{0,1\}, \mathbb{X} = [1, X_{i1}, X_{i2}, ..., X_{ik}], \forall i \in \{1, ..., n\}\}$$

In most practical cases, the intercept is added to the design matrix.

In matrix form:

$$\mathbb{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \mathbb{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & ... & x_{1k} \\ 1 & x_{21} & x_{22} & ... & x_{2k} \\ ... & ... & ... & ... & \\ 1 & x_{n1} & x_{n2} & ... & x_{nk} \end{bmatrix}$$

In practice, all $x_{ij}$ are either 0 or 1, indicating that a feature appeared in a certain document of the corpus, or the count of occurrence in a given document. The number of columns in the $\mathbb{X}$ matrix is equal to the number of unique terms in our document.

The terms could be unigrams, bigrams, stemmed words, whole sentences. It all depends on the problem and the insights of the researcher.

The general model for the binary response variable is:

$$log\frac{P(Y=1)}{P(Y=0)} = \sum_{i=0}^{k} \theta_i X_i \, (1)$$

## Maximum likelihood for estimating the coefficients

Let us define:

$$\theta := \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_k \end{bmatrix}$$

From (1), the probability of 'success' for each observation i can be rewritten as (Czepiel, 2017 , pp. 3-5):

$$\pi_i := P(\mathbb{Y}_i = 1) = \frac{e^{\theta^T \mathbb{X}_i}}{1 + e^{\theta^T \mathbb{X}_i}} = \frac{1}{1 + e^{-\theta^T \mathbb{X}_i}}$$

$$P(\mathbb{Y}_i = 0) = 1 - \pi_i$$

In every binary case, the $\mathbb{Y}$ can be encoded as a vector consisting of 0 and 1. Thus, we want $\theta$ that maximizes the product:

$$l(\theta) = \prod_{i=1}^{n} \pi_i^{y_i} (1 - \pi_i)^{1 - y_i} \ (2)$$

Logarithm is a monotone function thus the maximum of $l(\theta)$ is the same as $log(l(\theta))$ (Jose Manuel Pereira Mario Bastoa, 2016 , pp. 636 - 637).

$$L(\theta) := log(l(\theta)) = \sum_{i=1}^{n} [y_i log(\pi_i) + (1 - y_i) log(1 - \pi_i)] = \sum_{i=1}^{n} \left[ y_i log(\frac{\pi_i}{1 - \pi_i}) + log(1 - \pi_i) \right]$$

$$L(\theta) = \sum_{i}^{n} \left[ y_i \theta^T \mathbb{X}_i + log(1 + e^{\theta^T \mathbb{X}_i}) \right] \ (3)$$

The $\widehat{\theta}$ that maximizes the (3) equation will give a weight to every unique word in our text. This, in practice, may lead to some computational problems, because even a small text document can have thousands of unique words.

## Ridge logistic regression

Ridge logistic regression introduces an additional term to the (3) equation - the L2 penalty.

$$L^R(\theta, \alpha) = \sum_i^n \left[ y_i \theta^T \mathbb{X}_i + log(1 + e^{\theta^T \mathbb{X}_i}) \right] - \alpha \sum_{j=1}^k \theta_j^2$$

Often in practice, the $\alpha$ parameter is fixed to a certain value. As the parameter $\alpha$ increases, the ridge coefficient estimates will tend to approach zero. However, the penalty introduced in the log-likelihood function will shrink all of the coefficients towards zero, but it will not set any of them exactly to zero (van Wieringen, 2018 , pp.8). Hence, ridge regression has the disadvantage over model selection, of including all the predictors in the final model.

On the other hand, Ridge regression estimates gives us more uniformally distributed weights to all words. Depending on the problem, one may view this as an advantage.

## Lasso logistic regression

Lasso logistic regression is very similar to that of Ridge, but the penalty term is L1:

$$L^L(\theta, \alpha) = \sum_i^n \left[ y_i \theta^T \mathbb{X}_i + log(1 + e^{\theta^T \mathbb{X}_i}) \right] - \alpha \sum_{j=1}^k |\theta_j|$$

The L1 penalty used in the lasso is used for both variable selection and shrinkage, since it has the effect, when the $\alpha$ is sufficiently large, of forcing some of the coefficient estimates to be exactly equal to zero (Jose Manuel Pereira Mario Bastoa, 2016 , pp.637). In Lasso regression, the final model may involve only a subset of the predictors, which in turn improves model interpretability.

Depending on the research subject and the problem, having less predictors is beneficial.

## Elastic net regression

One is not confined to just using either Ridge or Lasso regression. The elastic net procedure tries to implement both of these methods:

$$L^{EN}(\theta, \alpha) = \sum_i^n \left[ y_i \theta^T \mathbb{X}_i + log(1 + e^{\theta^T \mathbb{X}_i}) \right] - (\alpha \sum_{j=1}^k |\theta_j| + (1 - \alpha) \sum_{j=1}^k \theta_j^2)$$

$\alpha \in (0, 1)$.

This approach is particularly useful when the number of predictors is much larger than the number of observations (Zou and Hastie, 2005).

## Formula in glmnet

The general notion of optimizing a function in the computer science world is to minimize instead of maximize the objective function. This is the case in glmnet package. The general formula which yealds the $\widehat{\theta}$ is the following:

$$-\frac{1}{N}\sum_{i}^{n}\left[y_i\theta^T\mathbb{X}_i + log(1+e^{\theta^T\mathbb{X}_i})\right] + \lambda\left(\alpha\sum_{j=1}^{k}|\theta_j| + (1-\alpha)\sum_{j=1}^{k}\theta_j^2\right)$$

If $\alpha = 0$ then we have Ridge regression. If $\alpha = 1$ then we have Lasso regression. $\lambda$ is a parameter that tell us how much weight should be put on the penalty of the parameters.

# Quora sincere question example

We will use the various types of logistic regression and text preprocessing techniques to a data set from the website Quora. On that website users can ask any question and other users can answer those question for recognition or special points. While most of the questions are sincere, some of the questions are dubious and it is not easy to keep track of all the questions for a human. Thus, machine learning algorithms can be used to determine whether a written question is genuine or not.

## Explanatory data analysis

| question_text | target |
|---|---|
| How did Quebec nationalists see their province as a nation in the 1960s? | 0 |
| Do you have an adopted dog, how would you encourage people to adopt and not shop? | 0 |
| Why does velocity affect time? Does velocity affect space geometry? | 0 |
| How did Otto von Guericke used the Magdeburg hemispheres? | 0 |
| Can I convert montra helicon D to a mountain bike by just changing the tyres? | 0 |
| Is Gaza slowly becoming Auschwitz, Dachau or Treblinka for Palestinians? | 0 |

| question_text | target |
|---|---|
| Has the United States become the largest dictatorship in the world? | 1 |
| Which babies are more sweeter to their parents? Dark skin babies or light skin babies? | 1 |

```
dim(D)
```

[1] 1306122       2

There are more than one million documents in the corpus.

The target column obtains two values: 0 if the question is sincere and 1 if the question is not sincere.

```
D$target %>%
  table() %>%
  data.table() %>%
  kable()
```

| . | N |
|---|---|
| 0 | 1225312 |
| 1 | 80810 |

Only about 6 percent of the documents in the corpus have a negative sentiment. We will balance the data by taking the same number of sincere sentiment rows as the insincere ones.

```
set.seed(1)
no_negative <- D[target == 1] %>%
  nrow()
D <- D[,.SD[sample(1:.N, no_negative, replace = FALSE)] , by = .(target)]
```

## Training and test set split

We will hold out some data when creating a model in order to see if our model will generalize well. Ideally, the proportion of two classes in the test data set must be similar to the overall data set.

We will hold out 30 percent of the original data set.

```
prop <- 0.3
rows_in_test <- round(nrow(D) * prop, 0)

## Sampling the row indices that will be in the test set
test_index <- sample(1:nrow(D), rows_in_test, replace = FALSE)

## Test set
test <- D[test_index]

## Train set
train <- D[setdiff(1:nrow(D), test_index)]
```

## Text preprocesing

We will drop the punctuation, make everything lowercase and remove the stop words (this, the, a, etc).

```
train <- train[, index := 1:.N]
prep_fun = function(x){
  x <- tolower(x)
  x <- str_replace_all(x, '[[:punct:]]', ' ')
  x <- removeNumbers(x)
  x <- removePunctuation(x)
  return(x)
}

it_train = itoken(train$question_text,
             preprocessor = prep_fun,
             tokenizer = word_tokenizer,
             ids = train$index,
             progressbar = FALSE)
vocab = create_vocabulary(it_train,
                          stopwords = stopwords(source = 'smart'))
```

We will prune the vocabulary to include only words that appear in more than 10 documents and do not appear more than in 50 percent of the documents. Finally, we will transform our document term matrix using the tf idf transformation.

```
pruned_vocab = prune_vocabulary(vocab, doc_count_min = 10,
                                doc_proportion_max = 0.5)
vectorizer = vocab_vectorizer(pruned_vocab)

dtm_train  = create_dtm(it_train, vectorizer, type = 'dgCMatrix')

tfidf = TfIdf$new()
dtm_train_tfidf = fit_transform(dtm_train, tfidf)
```

## Number of features

We will create several models with different alpha values using the glmnet package . The first model ($\alpha = 0$) will represent the Ridge logistic regression while the last one ($\alpha = 1$) will represent the Lasso logistic regression.

```
## Defining the range of alpha

alpha_range <- seq(0, 1, length.out = 20) %>%
  round(3)

## Constructing a list to store the values and the models in

result_coef <- vector('list', length(alpha_range))
names(result_coef) <- alpha_range

result_model <- vector('list', length(alpha_range))
names(result_model) <- alpha_range

## Iterating over all alphas

for(alpha in alpha_range){
  model <- glmnet(x = dtm_train, y = train$target,
                  family = 'binomial',
                  alpha = alpha,
                  lambda = 0.01)
  coef_table <- data.table(feature = rownames(coef(model)),
                           coefficient = coef(model)[, 1]) %>%
    .[order(-coefficient)] %>%
    .[coefficient!=0]
  result_coef[[as.character(alpha)]] <- coef_table %>%
    .[, alpha := alpha]

   result_model[[as.character(alpha)]] <- model
}
```

```
no_ft <- lapply(result_coef, nrow) %>%
  unlist()
result_frame <- data.table(alpha = alpha_range, no_features = no_ft)
kable(result_frame)
```

| alpha | no_features |
|---|---|
| 0.000 | 8790 |
| 0.053 | 6334 |
| 0.105 | 4623 |
| 0.158 | 3410 |
| 0.211 | 2534 |
| 0.263 | 1928 |
| 0.316 | 1509 |
| 0.368 | 1234 |
| 0.421 | 1024 |
| 0.474 | 867 |
| 0.526 | 748 |
| 0.579 | 654 |
| 0.632 | 589 |
| 0.684 | 521 |
| 0.737 | 466 |
| 0.789 | 410 |
| 0.842 | 367 |
| 0.895 | 338 |
| 0.947 | 303 |
| 1.000 | 277 |

```
barplot(result_frame$no_features, result_frame$alpha, xlab = 'alpha',
        ylab = 'Number of features',
        names.arg = result_frame$alpha, col = 'royalblue')
```

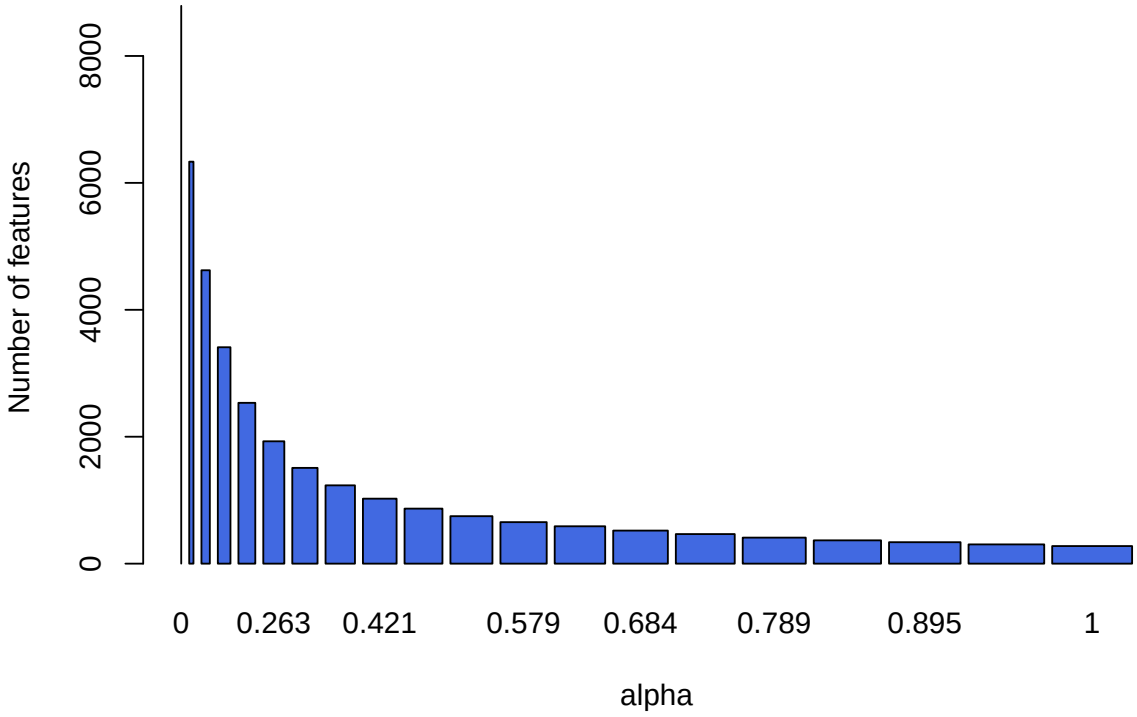


Figure 1: Barplot of features

## Accuracy calculation

```
## Preprocesing of text

test <- test[, index := 1:.N]

it_test = itoken(test$question_text,
             preprocessor = prep_fun,
             tokenizer = word_tokenizer,
             ids = test$index,
             progressbar = FALSE)

vocab = create_vocabulary(it_test, stopwords = stopwords(source = 'smart'))
vectorizer = vocab_vectorizer(vocab)
dtm_test  = create_dtm(it_test, vectorizer, type = 'dgCMatrix')

## Tf-idf

tfidf = TfIdf$new()
dtm_test_tfidf = fit_transform(dtm_test, tfidf)

## Adding features from the train frame

features <- colnames(dtm_train_tfidf)
missing_ft <- setdiff(features, colnames(dtm_test_tfidf))
mm <- sparseMatrix(i = nrow(dtm_test_tfidf), j = length(missing_ft))
colnames(mm) <- missing_ft
dtm_test_tfidf <- cbind(dtm_test_tfidf, mm)
dtm_test_tfidf <- dtm_test_tfidf[,features]

## Creating an object to store the results

acc_scores <- vector('list', length(alpha_range))
names(acc_scores) <- alpha_range

## Iterating

for(alpha in alpha_range){
  result <- data.table(target = test$target,
                       fit = predict(result_model[[as.character(alpha)]],
                                     dtm_test_tfidf, type = 'class'))

  acc_scores[[as.character(alpha)]] <- sum(result$target == result$fit.s0)/nrow(result)
}
```

```
## Outputing scores

acc_scores <- acc_scores %>%
  unlist() %>%
  round(3)
result_frame <- data.table(alpha = alpha_range, accuracy = acc_scores)
kable(result_frame)
```

| alpha | accuracy |
|---|---|
| 0.000 | 0.822 |
| 0.053 | 0.829 |
| 0.105 | 0.830 |
| 0.158 | 0.824 |
| 0.211 | 0.817 |
| 0.263 | 0.812 |
| 0.316 | 0.805 |
| 0.368 | 0.799 |
| 0.421 | 0.794 |
| 0.474 | 0.788 |
| 0.526 | 0.782 |
| 0.579 | 0.778 |
| 0.632 | 0.771 |
| 0.684 | 0.767 |
| 0.737 | 0.762 |
| 0.789 | 0.757 |
| 0.842 | 0.753 |
| 0.895 | 0.749 |
| 0.947 | 0.745 |
| 1.000 | 0.741 |

# References

[1] S. A. Czepiel. *Maximum Likelihood Estimation of Logistic Regression Models: Theory and Implementation*. Article. 2017. <URL: https://czep.net/stat/mlelr.pdf>.

[2] A. F. d. S. Jose Manuel Pereira Mario Bastoa. *The logistic lasso and ridge regression in predicting corporate failure*. Article. 2016. <URL: https://www.sciencedirect.com/science/article/pii/S2212567116303100>.

[3] D. Jurafsky and J. H. Martin. *Speech and Language Processing An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition* . 2018. <URL: https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf>.

[4] D. Selivanov and Q. Wang. *text2vec: Modern Text Mining Framework for R*. R package version 0.5.1. 2018. <URL: https://CRAN.R-project.org/package=text2vec>.

[5] N. Simon, J. Friedman, T. Hastie, et al. "Regularization Paths for Cox's Proportional Hazards Model via Coordinate Descent". In: *Journal of Statistical Software* 39.5 (2011), pp. 1-13. <URL: http://www.jstatsoft.org/v39/i05/>.

[6] C. Trim. *Tokenization*. Article. 2013. <URL: https://www.ibm.com/developerworks/community/blogs/nlp/entry/tokenization?lang=en>.

[7] W. N. van Wieringen. *Lecture notes on ridge regression*. 2018. <URL: https://arxiv.org/pdf/1509.09169;Lecture>.

[8] H. Zou and T. Hastie. *Regularization and variable selection via the elastic net*. Article. 2005. <URL: https://web.stanford.edu/~hastie/Papers/B67.2%20(2005)%20301-320%20Zou%20&%20Hastie.pdf}.>