Visualisation Research Center (VISUS)

University of Stuttgart
Almandring 19
70569 Stuttgart
Germany

Diplomarbeit Nr. 3489

Analyzing Textual Data by Multiple Word Clouds

Nils Rodrigues

Course of Study:	Softwaretechnik
Examiner:	Prof. Dr. Daniel Weiskopf
Supervisor:	Dr. rer. nat. Michael Burch, M. Sc. Lorenzo Di Silvestro
Commenced:	May 1, 2013
Completed:	October 31, 2013
CR-Classification:	H.3.3

Abstract

Word clouds are a good way to represent textual data with meta information. However, when it comes to analyzing multiple data sources, they are difficult to use. This is due to their poor comparability. The proposed RadCloud merges multiple standard clouds into a single one, while retaining the information about the origin of the words. Using separate clouds as well as this new visualization technique, a tool for textual data analysis is created: MuWoC. The resulting software is then evaluated by using it with multiple data sources ranging from encyclopedia articles, over literature to custom CSV files.

Contents

List of Figures

List of Listings

List of Algorithms

1 Introduction

Personal and public information is often recorded and archived as textual data. For further processing at a later point in time, it is necessary to read high amounts of the documents to get an overview of the most important information and the relations between the individual sources of data. This method is accurate but also very time consuming in comparison to a computer-based analysis.

To be of any help, the computation results will have to be understood by humans. Much of the information processed in daily life is of visual nature. Someone can safely cross a street when hearing music with headphones but when the eyes are covered, the task is more difficult. Representing extracted information as images makes use of the relatively sophisticated visual system. The visualization pipeline [HM90] describes the course steps from getting data to creating displayable images. Most forms of information visualization work according to this pipeline, but vary in the implementation of the individual steps.

Word clouds represent a possible way of visually displaying textual data. As Thomas Gottron has shown they provide a quick way of quickly finding the importance of texts [Got09]. They are readable and understandable while at the same time working with only a reduced context. With the advances of the world wide web and the mass emergence of personal blogs, word clouds often appear as a guidance system for visitors to find related or popular information.

However, when this visualization is to be used on multiple data sources, for instance a selection of groups of metals (see section 4.1), it does not scale to the task. It is possible to create a single cloud for all of them, but the user looses the information about the origin of the words. In the context of a textual analysis task, this information in itself could be very relevant. To find out what makes stands out in each group of metals, it is necessary how the words relate to the individual groups. Therefore such a single word cloud can not be used for the comparison of the different data sources.

The use of multiple independent clouds solves the issue of origin, but performs poorly when the observer wants to compare them. The algorithms for this kind of visualization can place and style the elements randomly. Thus the same word can appear with different font families, sizes, colors and even rotations in the individual clouds. In the case of the metals, it would be necessary to analyze each word clouds individually and to perform their comparison manually.

A possible solution for problems with the use of multiple data sources and their comparability is the development of a single visualization that still shows the words' provenance. However, a visual display alone sometimes is not enough for an in-depth analysis. It has to be embedded into a tool that provides the data to be displayed as well as ways of interaction. This way the integrated tools can be of help to a broader variety of users.

The use case of the groups of metals serves as main use case test for the tool. It must provide a method of finding out what distinguishes the individual groups and what they have in common. This could even give a hint at the qualities that are responsible for the classification of the different chemical elements as metals.

Other use cases include:

- The Reuters' 21578 data set (section 4.2) as often used text classification target.
- Translations of a selection of Jule Verne's stories (section 4.3) as English literature.
- Three versions of Goethe's Faust (section 4.4) as German literature.
- A selection of actors that participated in movies by Wes Anderson (section 4.5) as manual input source.

Outline

This document contains the fundamental ideas and provides insight into the development of the software MuWoC. The name originated from the merging of "Multiple Word Clouds" and is pronounced like *Mooh-vok*. The tool was designed and developed to answer an unasked question by employing word clouds in the context of textual data from multiple sources. Chapter 2 contains information about the development of several algorithms for the creation of the visualizations. Chapter 3 explains different aspects of the software that exploits the novel approach of multiple word clouds. In chapter 4, the tool is used on sample data to evaluate the results.

2 Word Clouds

There are several ways to visually represent textual data. One is in the form of word clouds that are composed of single words. The restriction to extracted tokens is their biggest difference from plain text. Those words are then presented with a reduced context, while introducing new information in the form of abundant attributes [LL07]. These also include styling as for example the words' color, size or font. Such attributes can carry information or – not mutually exclusive – catch the beholder's attention.

Furthermore words in plain text have a fixed position inside sentences; their order is provided by their semantic meaning and grammatical rules and is therefore often unambiguous. Word clouds instead, have no fixed rules of placement, so that their elements can appear anywhere.

Since the clouds have a reduced context, the represented words have to be chosen carefully in order to be able to summarize a complete text.

At the beginning of the development of MuWoC stood the separate word clouds as they already existed. To provide more information and a better analytical tool for the user, these clouds were then merged into a new representation.

2.1 Standard word clouds

Over time there have been multiple implementations of word clouds. The currently most prominent one is Wordle [Fei13]. It is implemented in Java and embedded on a website for free public use (http://www.wordle.net/). On this site the user can upload any text and let the service create a word cloud from it. The result depends mostly on the frequency with which each word appears. The more often a word is used, the higher its importance is rated. This relevance is then used to select the topmost n words and calculate their font size. The more important a word is, the bigger it will be drawn [Fei10].

Up to now Wordle's results are strictly determined by the input, but this changes as colors and the word's position inside the cloud are concerned. The user can provide a palette from which the software will assign a color to each word. Furthermore, the algorithm can introduce small variations to the palettes.

The user can also select to have some words rotated, so that they are not horizontal anymore. This helps to reduce whitespace and packs them tighter together. However, a word's position inside a cloud is not always predetermined by grammatical rules and therefore it can differ from implementation to implementation.

2.1.1 Wordle's Spiral

Wordle's algorithm measures the size of each word and then estimates the size of a target area that will probably be necessary to place them all in a coherent cloud. An intended position for each word is then randomly chosen from within this target area [Fei10]. The random nature of this distribution allows – among other possibilities – for more important words to be placed closer to the edges of the cloud, letting the less relevant ones stay in the middle.

If a word is placed in a position where it intersects with another, it is moved along a spiral that grows outward and starts at the word's originally intended position. This movement is performed until a free space is found that is preferably not outside the initially estimated area. To test for intersections, the Wordle algorithm uses hierarchical bounding boxes to optimize performance in comparison to a purely vector based approach [Fei10].

2.1.2 MuWoC's Implementations

As stated by the author, Wordle "was designed for pleasure" [Fei10]. MuWoC however, follows a pragmatic approach to text analysis. Thus the user has to be able to easily find information. To achieve this goal guidelines were developed for all of MuWoC's visualizations to follow:

Letter casing

Words are more readable if their letters are not converted to lower or upper casing. The cause being that the human brain first recognizes whole words and only later individual letters [Joh75]. At the same time, peripheral vision can recognize the shapes of words before the main visual focus gets to them [Ray75]. As a result, the word *key* is first seen as ▙. If a word is completely capitalized to only use upper case letters, the shape is also changed. The same word *KEY* would then appear to the peripheral vision in the shape of a rectangle ▬. This means that humans would take longer to read because of the need to use the visual focus to recognize the words.

Separation

Words that have space around them can be read more easily [RFP98]. If the vertical distance between the words *key* and *lock* is removed, the *y* from the former invades the space between the *l* and *k* of the latter. As a result the words don't form the shapes they normally would and the brain is forced to recognize the letters themselves.

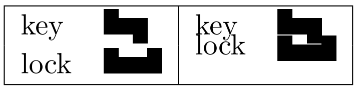


Figure 2.1: The effect of missing space space between words

Rotation

Words can be read best, when they are presented in the same direction we are accustomed to [Hue98]. Since the majority of languages this means that horizontal words are easier to read than oblique ones. Therefore the cloud should not rotate them, as readability is more important than space utilization.

Placement of important words
In many word clouds the more important words can be found by observing their font size. This is sufficient in smaller clouds that occupy little space. However, if the words are distributed throughout large clouds, it becomes more complicated for the user as it is difficult to compare the sizes of objects that are farther away from each other [CM86]. Even situations similar to the ones in the Ebbinghaus Illusion [RHY05] can occur, which then also lead to false size impressions. Spatial distance also forces the user to scan the entire cloud to find the most relevant words. Therefore important words should be drawn near each other.

Following these guidelines, allows for two simplifications of Wordle's algorithm for placing a word in the cloud:

- no rotation
- only one bounding box around the entire word

To accommodate different users needs, MuWoC implements multiple layout algorithms. In spite of their different approaches to the word placement, they still provide some continuity by only assigning a position to a word but not changing its appearance.

2.1.2.1 Spiral

The first one was derived from Wordle and therefore has similarities as for example the use of a spiral. Other researches have also recently selected this spiral algorithm as it is relatively simple to implement and can produce satisfactory images [BLW13]. MuWoC also follows Feinberg's "greedy" approach [Fei10] as once a word has been placed, it will not move. It also uses a spiral that starts at the intended position of a word and goes outwards.

However MuWoC's implementation also has differences. There is no estimated cloud area, that is used to control the shape. This results in the placement of the words near their initial position, and the growth of the cloud into any shape. A second difference, is that the algorithm tries to put every word in the center of the cloud. Therefore, as the cloud grows, its shape becomes circular.

The approach with a common starting point also presents a problem in combination with the algorithm's greedy nature. The first word, would be in the center, whether it was important or not. Subsequent, maybe more relevant, words would be placed nearer to the outside and would be randomly distributed because they would have followed the spiral. This is not a wanted feature. The human vision has a focal point where it is sharpest, the so called *fovea*[RB79]. This point is at the center of the field of vision. The farther something is from the center, the more blurred it appears [Ray75].
To keep important words together and most readable when looking at the entire cloud, they should be in the center. To achieve this, they have to be ordered in a descending fashion,

before they are placed in the cloud. This solves the previous problem and also add the distance from the center as an additional measure of importance for the user.

The archimedean spiral can be considered a circle, that grows, as the perimeter is drawn. Therefore the known trigonometrical functions to be used in algorithm 2.1 to calculate individual points along the spiral.

Algorithm 2.1 Spiral coordinates

```
1: function SpiralCoordinates(point $center$, float $angle$, float $radius$)
2:     radius ← $radius \times (angle \div 360)$        // $angle$ can be bigger than 360
3:     coords.x ← $center.x + (radius \times \cos angle)$
4:     coords.y ← $center.y + (radius \times \sin angle)$
5:     return coords
6: end function
```

For algorithm 2.1 a point on the spiral is represented by an angle. Since a spiral can have more than one turn, this angle can be greater than 360 degrees. The constant increase in distance from the center is represented by the increase in radius for every turn. This algorithm further needs a given center. MuWoC's implementation always starts with the origin of the coordinate system in the center, which means that two addition operations can be saved. The constant center also allows further optimization by reusing previously calculated positions. In fact, these points only have to be recalculated when the parameters of the spiral change and can otherwise be stored in an array that grows dynamically as needed. For the placement of subsequent words, the points can then be retrieved by array arithmetic instead of trigonometrical functions.

Since the spiral itself is a curved line, it contains an infinite number of points. For the placement of words only discrete points are selected as possible positions. Their selection is performed by walking the spiral in discrete steps of angles. However the step size has to decrease as the spiral grows or otherwise all words would be placed along straight rays that originate at the center. To avoid this problem the size of the steps is decreased for every new point:

$$(2.1)\quad \alpha_{n+1} = \alpha_n * \omega \quad (\text{where } \omega \in\,]0,1[\text{ and } \omega \in \mathbb{R})$$

ω has to be big enough, to not make the steps too small too quickly and small enough to have an effect. However this approach has to use an ω that fits the distance between two subsequent turns of the spiral. If it is too small, the new positions will be too near, thus essentially stopping the spiral from growing and creating an endless loop in the program.

Before the words are finally placed the calculated coordinates have to be transformed. Because MuWoC is developed with Microsoft's Windows Presentation Foundation (WPF), the user interface (UI) elements are not drawn manually during the layout process, but just positioned. The framework then takes care of the rendering. The origin of the WPF coordinate system is always at the upper left corner of the parent UI element. Therefore all calculated positions for the words have to be offset, so that the cloud's bounding box starts at the container's origin. For this, the final size has to be known and therefore the translation of coordinates is only performed when all words have been placed in the cloud.

To fit the different placement algorithms into WPF's two-stage layout logic two methods have to be implemented (see algorithm 2.2). First the size of each contained UI element (child) has to be measured. In a second step, they have to be arranged into their final position. In this case the children are the words.

Algorithm 2.2 Word placement in standard clouds

```
procedure MEASURE(vector $availableSize$)
    for all child $\in$ this.Children do
        child.Measure($availableSize$)          // Gets the size the words would like to have
        Placement.Add(child, child.DesiredSize)
    end for
    Placement.MoveToOrigin()
    this.DesiredSize $\leftarrow$ Placement.Bounds.Size
end procedure

function ARRANGE(rectangle $finalPosition$)
    for all child $\in$ this.Children do
        position $\leftarrow$ Placement.GetPosition(child)
        child.Arrange(position)                 // Put the words at their calculated position
    end for
    return this.DesiredSize
end function
```

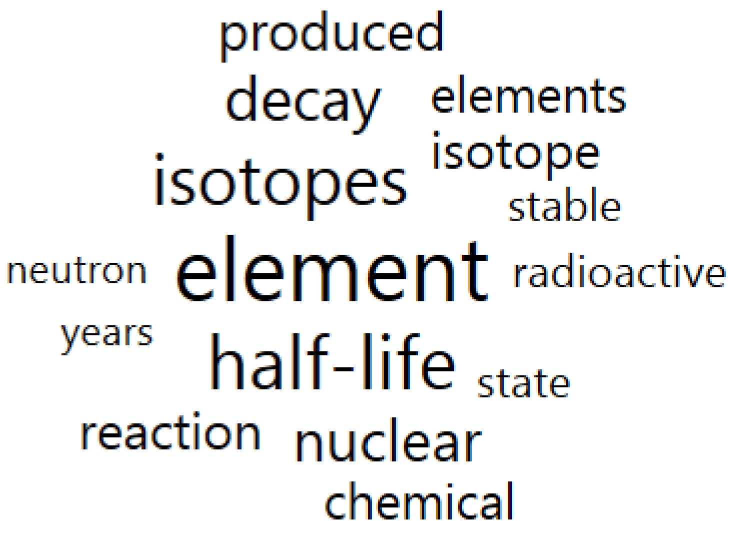


Figure 2.2: Sample result of the spiral layout algorithm.

2.1.2.2 Layered Rings

The approach with layered rings approximates the spiral while eliminating its problem with the coefficient ω (equation (2.1)) that could lead to endless loops. Multiple circles are used instead of one spiral. The radius of each circle is calculated as follows:

$$layerRadius = radiusPerLayer \times \lfloor \alpha \div 360 \rfloor \tag{2.2}$$

The step size between the points of one layer is equal but for each layer, more points are selected:

$$\begin{aligned} \alpha_{n+1} &= \frac{360}{m_{n+1}} && \alpha \text{ represents the step sizes for points on the rings} \\ m_{n+1} &= m_n + d && (m_n = \text{ number of points in circle } n\text{; constant } d \in \mathbb{N}) \end{aligned} \tag{2.3}$$

Similarly to the spiral approach, the performance of this algorithm can be improved if the calculated points of insertion are reused between the words.

2.1.2.3 Space station

The Spiral and the Layered Rings algorithms place words relatively near to the position they were supposed to be. However, following the spiral or rings also means, that there might be space between the words, that could be filled.

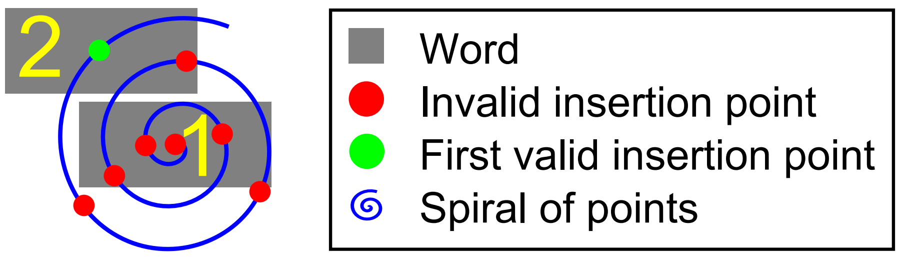


Figure 2.3: Placing words at selected points on a spiral leads to gaps.

Therefore a third layout algorithm was developed. The first basic rule is that a word's bounding box should always touch at least one other word's box to avoid wasting space. Further more, since the bounding box is a rectangle, it has four sides with which is can connect to other words.

To accommodate these rules, the 2-dimensional projection of a simplified space station can be used:
Each word represents a module that has four docking ports: top, right, bottom and left. These ports are situated at the center of their respective sides of the bounding box. When words are placed in the cloud, they connect to other words by docking.
To achieve a compact visual representation a new word should always dock at the port that

is nearest to the center of the cloud. These rules lead to algorithm 2.3, which is graphically explained in figure 2.4.

As with the previously mentioned spiral approach, the cloud's final size is not known before all words have been placed and therefore a coordinate translation is needed before the rendering phase (sample result in figure 2.5).

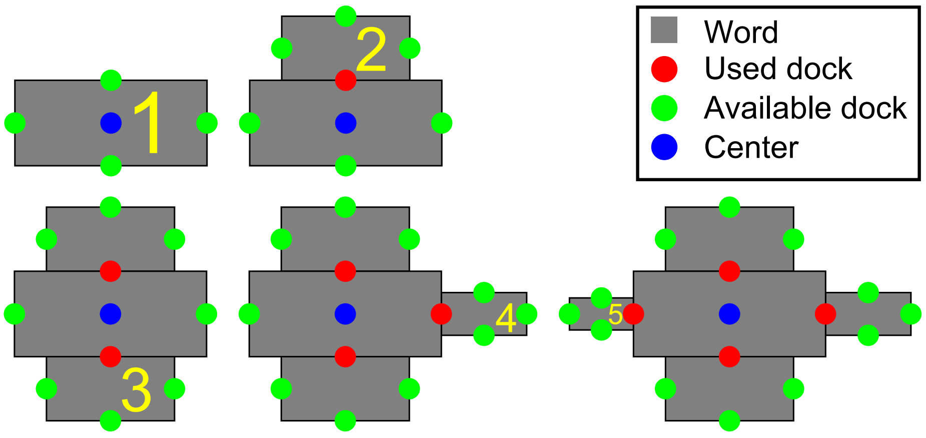


Figure 2.4: Explanation of the space station algorithm.

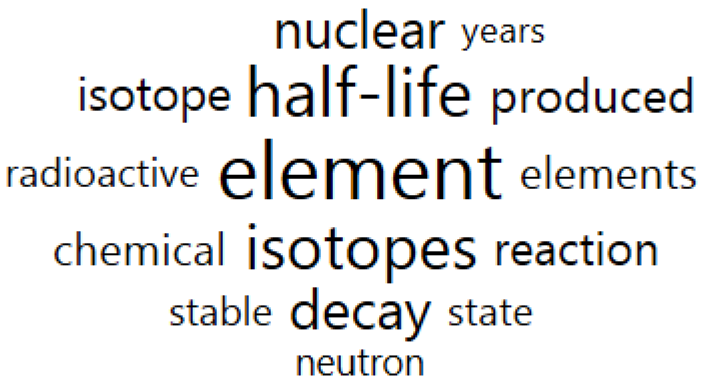


Figure 2.5: Sample result of the space station layout algorithm.

Algorithm 2.3 The space station algorithm – Part I

```
procedure AddWord(rectangle word)
    if this.Docks = $\emptyset$ then                    // The docks are stored in a sorted set (by distance)
        $word.X \leftarrow -word.HalfWidth$
        $word.Y \leftarrow -word.HalfHeight$
        this.Modules.Add(word)
        AddDocks(word, dock)                                    // See Part II on page 19
    else
        dockEnumerator ← this.Docks.GetEnumerator()
        repeat
            dock gets dockEnumerator.GetNext()
            word.Position ← GetPosition(word, dock)
        until Overlaps(word) = false
        this.Modules.Add(word)
        this.Docks.Remove(dock)
        AddDocks(word, dock)                                    // See Part II on page 19
    end if
end procedure

function GetPosition(rectangle word, dock dock)
    newPosition ← dock.Position
    switch dock.Direction do
        case top
            newPosition.X ← newPosition.X−word.HalfWidth
            newPosition.Y ← newPosition.Y−word.Height
        end case
        case right
            newPosition.Y ← newPosition.Y−word.HalfHeight
        end case
        case bottom
            newPosition.X ← newPosition.X−word.HalfWidth
        end case
        case left
            newPosition.X ← newPosition.X−word.Width
            newPosition.Y ← newPosition.Y−word.HalfHeight
        end case
    end switch
    return newPosition
end function

function Overlaps(rectangle word)
    for all module ∈ this.Modules do
        if module.Overlaps(word) then
            return true
        end if
    end for
    return false
end function
```

Algorithm 2.4 The space station algorithm – Part II

```
48: procedure AddDocks(rectangle word, dock dock)
49:     if dock.Direction ≠ bottom then
50:         newDock.Position ← word.TopCenter
51:         newDock.Direction ← top
52:         newDock.Distance ← $\left|\overrightarrow{\text{newDock.Position - this.Center}}\right|$
53:         this.Docks.Insert(newDock)
54:     end if
55:     if dock.Direction ≠ left then
56:         newDock.Position ← word.RightCenter
57:         newDock.Direction ← right
58:         newDock.Distance ← $\left|\overrightarrow{\text{newDock.Position - this.Center}}\right|$
59:         this.Docks.Insert(newDock)
60:     end if
61:     if dock.Direction ≠ top then
62:         newDock.Position ← word.BottomCenter
63:         newDock.Direction ← bottom
64:         newDock.Distance ← $\left|\overrightarrow{\text{newDock.Position - this.Center}}\right|$
65:         this.Docks.Insert(newDock)
66:     end if
67:     if dock.Direction ≠ right then
68:         newDock.Position ← word.LeftCenter
69:         newDock.Direction ← left
70:         newDock.Distance ← $\left|\overrightarrow{\text{newDock.Position - this.Center}}\right|$
71:         this.Docks.Insert(newDock)
72:     end if
73: end procedure
```

2.1.2.4 Vertical stack

All previously described placement algorithms have produced word clouds that form a dense cluster. There representations of words are well suited for standard clouds, but there are issues when working with multiple ones side by side. For example, the user can not always say in which order the words were placed into the visualization. Therefore it is difficult to compare the individual word's importance in one cloud with it's relevance in another.

A vertical stack algorithm trades off the compact representation of a cloud for better comparability. As already mentioned above (section 2.1.2.1 Spiral), the words are ordered by relevance. Placing each word beneath the previous one therefore creates an ordered list. As visible in figure 2.6, the user can choose a word and quickly compare its position in other clouds.

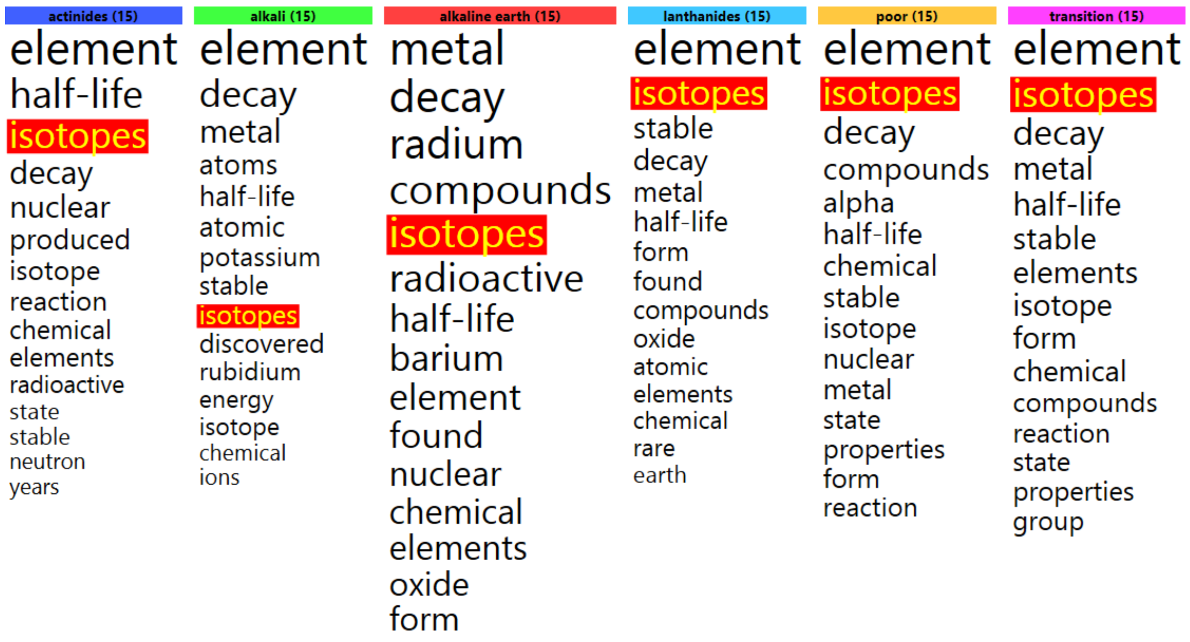


Figure 2.6: Word clouds as layed out by the vertical stack algorithm and with synchronized selection.

2.2 Merged word clouds

All algorithms presented earlier, always produce standard separate word clouds. These may be helpful for several tasks, but when one cloud at a time is not enough, multiple ones have to be used. One use case where one cloud would not be enough, is when working with textual data that falls into different categories: using only one cloud would not provide the user with the information about the source of a word. The creation of separate clouds – one for each category – can reintroduce this lost information.

However, separate clouds also lack some basic, but useful, functionality:

Set comparison
: Word clouds as mentioned until now are essentially sets of unique character tokens. Very prominent operations on sets are their comparisons. Euler and Venn diagrams provide visual representations that allow for fast comparability. But word clouds are always next to each other without overlapping, thus they get harder to compare as each single word from one cloud has to be searched in the others.

Relevance in categories
: A word can be very important in one category and less relevant in another. To compare those importances the beholder first has to find the word in each cloud. This can be very time consuming if there are many clouds or if they contain large amounts of words.

Castella and Sutton approached the problem of bad comparability by creating *word storms* with similarities between the clouds [CS13]. Their algorithm tries to maintain the words' size, position, color and orientation in all separate word clouds. This works well for a limited number of words and clouds. However, as the amounts increase it requires more and more comparative work from the users.

Collins and Carpendale created *VisLink* [CC07]. It helps with comparisons between many types of visualizations, as long as they represent the same data. To achieve this, the visualizations are drawn on a semi-transparent surface that can be moved in three dimensional space. The user can select rendered elements and VisLink then adds lines between those elements in one visualization and their representation in the other.

Collins et al. also developed *parallel tag clouds* [CVW09]. This representation allows for fast comparisons while the individual clouds are laid out similarly to the vertical stack algorithm (see section 2.1.2.4). However, the words still appear multiple times.

To solve both problems (*Set comparison* and *Relevance in categories*), the idea of a single cloud and separate ones for each category can be merged. Therefore providing only one cloud that contains the information about the belonging of words to the categories. Such a merged cloud should not contain multiple copies of the same words because it would force the user to search for all appearances. The merging should therefore be performed like the union operation in set theory to offer additional value over separate representations as mentioned in chapter 1.

2.2.1 Previous work on merging two clouds

The need for comparability between word clouds has risen especially in the context of sociology. In order to satisfy this requirement, there have already been multiple attempts for creating merged word clouds that contain the information about the source of the words. The origin can then in turn be used to deduct other information.

Ian Fellows created the *comparison cloud* in the programming language of *R* [Fel12]. *R* was developed for statistical calculations and is well suited for the task because of the similar calculations necessary for the analysis of statistical and textual data. The comparison cloud contains the differences between texts and merges them into one cloud. To separate the words from different sources, the available space is divided equally, resulting in a radial layout (see figure 2.7a) [Fel12].
However this visualization only depicts differences, thus it won't contain the same word in more than one cloud. This also means, that from a visual stand point each word belongs to exactly one cloud. When two arbitrary separate representations both contain the same word, this information is therefore lost in the process of combination.

When Professor Winter Mason used his *Comparative Word Cloud* [Mas12] on the transcripts of two speaches from Sarah Louise Palin and Barack Hussein Obama II, the result was figure 2.7b. He addressed the issues of merging with an implementation that also uses *R* and builds two clouds based on the appearance frequency of words from two source texts. These clouds can potentially have common words. When merging them, instead of using a random algorithm for

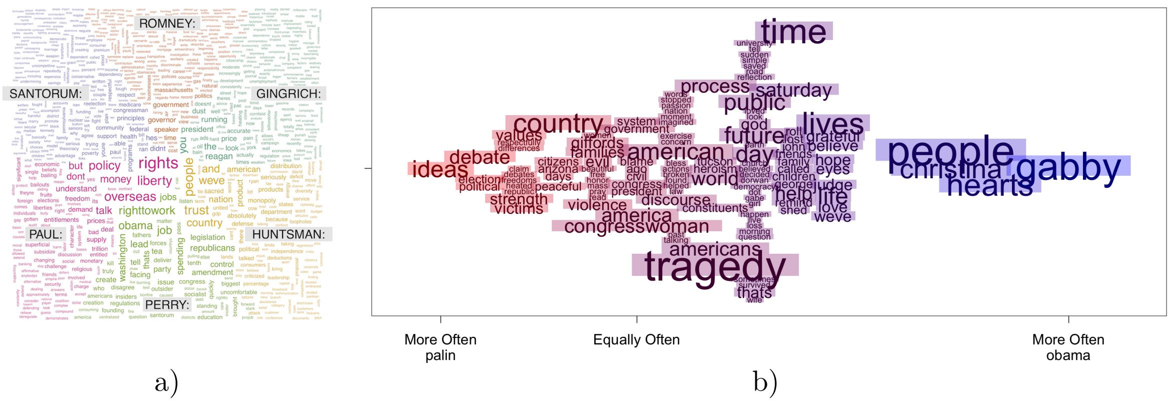


Figure 2.7: Examples of merged word clouds from **(a)** Ian Fellows' Comparison Cloud and **(b)** Winter Mason's Comparative Word Cloud.

an initial word position, he creates a horizontal scale. Each of the two documents represents an end of this scale. The more often a word appears in a document, the farther it moves to the respective end. Collisions are then avoided by vertical movement.

Mason's work therefore is very suitable to accomplish the goal of merging clouds, while also showing how much a word belongs to a category.

2.2.2 RadCloud

Even though the Comparative Word Cloud [Mas12] works very well, it is restricted to two clouds. A general distribution of information along an axis or scale has previously been described by Hoffman et al. as *dimensional anchors* [HGP99]. These anchors themselves, again, are a generalization of Hoffman et al. previous work on *RadViz*, a visualization for multivariate data [HGM$^+$97].

RadViz was created by merging two ideas [HGM$^+$97]:

- Ankerst et al. developed the *Circle segments* visualization, where the space in a circle was equally divided to contain single pixels [AKK96].
- Olsen et al. were looking for a way to visualize documents in the context of a retrieval system. The result was *VIBE* [OKS$^+$93].

VIBE's layout system attached a metaphorical approximation of physical springs between mobile documents and stationary points of interest (POI). The spring constants were then modeled by a measure of belonging or relevance of the documents in the context of certain interests [OKS$^+$93]. To distinguish this type of springs from others that will be introduced later in this document, they will be labeled as being "positional".

For RadViz, the positioning of points of interest was restricted to a circle. As in the *Circle Segments*, the points were equally distributed, to get a segmentation. Furthermore, Hoffman et al. transfered the calculation of spring constants from belonging to a POI to belonging to any value. This enables RadViz to visualize multidimensional data, where every rendered point has springs to all stationary dimension anchors [HGM+97].

Bertini et al. created *SpringView* [BDS05] by using this method side by side with a parallel coordinates visualization for a more detailed view. Their study used dimensions like horsepower, mileage and country of origin to plot the data from a set of cars. Using *SpringView*, they noticed that many powerful cars got fewer miles per gallon and came mostly from the USA.

2.2.2.1 RadViz for merged word clouds

The RadViz visualization technique can be used to create merged word clouds: the RadCloud is born. Instead of rendering a single icon at a data points' position p_w, a word is drawn at p_w. Where Olsen et al. used points of interest and Hoffman et al. dimensions, RadCloud uses categories. The spring constants are gathered by taking the word's relevance in each category.

2.2.2.2 Implementation

The way in which Olsen at al. calculated the position of a document can be described as follows after these definitions:

$$\begin{aligned} P_i &:= position(POI_i) \\ w_i^j &:= weight(document_i, POI_j) \\ D_i^j &:= position(document_i) \text{ after consideration of } POI_0 \ldots POI_j \\ D_i^0 &:= P_0 \\ V_i^1 &:= \overrightarrow{P_1} - \overrightarrow{P_0} \\ V_i^{j+1} &:= \overrightarrow{P_{j+1}} - \overrightarrow{D_i^j} \end{aligned} \tag{2.4}$$

First two points of interest POI_0 and POI_1 are selected. Then the vector $\overrightarrow{V_1}$ between the POIs is calculated. The document then starts at P_0 and moves along V_1 as much as the relative weight in POI_1. Other POIs are added into the equation one by one, adapting the document's position until the last POI has been used [OKS+93]:

$$D_i^{j+1} = \overrightarrow{D_i^j} + V_i^{j+1} \times \frac{w_i^{j+1}}{\sum_{k=0}^{j+1} w_i^k} \tag{2.5}$$

RadCloud approaches the calculation of a word's position differently to optimize the performance. First when the data is loaded the words have to get a relative weight ${w'}_i^c$ in each category:

$$\begin{aligned}
\mathcal{C} &:= \{c_1, \dots, c_u\} = \{\text{category}_1, \dots, \text{category}_u\} = \text{all categories} \\
words^c &:= \text{ all words that belong to } category_c \\
words &:= \bigcup words^c \\
w_i^c &:= \begin{cases} 0 & \text{if } w_i \notin words^c \\ weight(word_i, category_c) \in \mathbb{R} & \text{if } w_i \in words^c \end{cases} \\
{w'}_i^c &:= \frac{w_i^c}{\sum_{k=1}^{|words|} w_k^c} \quad \text{(normalized weight in category)}
\end{aligned} \tag{2.6}$$

The circle in which all words will be placed is centered around the coordinate system's origin. The categories are equally distributed along this circle. The individual positions are then obtained by rotating the point at the circle's top. The vectors V_c from the origin to the categories are then very simply created as:

$$V_c := \overrightarrow{position(category_c)} \tag{2.7}$$

The category positions and spring vectors are cached and reused as long as the render area, the category order or visibility do not change. The intended placement for each word is then obtained using:

$$\begin{aligned}
{w''}_i^c &:= \frac{{w'}_i^c}{\sum_{k=1}^{|\mathcal{C}|} {w'}_i^k} \\
position(word_i) &:= \sum_{c=1}^{|\mathcal{C}|} \left(V_c \times {w''}_i^c\right)
\end{aligned} \tag{2.8}$$

Figure 2.8 illustrates the composition of the intended position by multiple separate vectors to the categories.

2.2.2.3 Overlap avoidance

RadViz was meant for the representation of data as single points or small shapes. Words however, take up a relatively large area. This leads to many overlaps and therefore poor readability. An overlap avoidance system is necessary to arrange the words as near to their intended position as possible, while maintaining a readable representation without overlaps.

2.2.2.3.1 Preservation of semantics

To achieve this, the previous positional springs that moved a word to its intended position are disconnected from it and attached to a joint instead. This joint is at the same position as intended for the word, but it is fixed in the layout and will not move. A new avoidance spring is then added, connecting the joint to the center of the word. The new spring has a constant

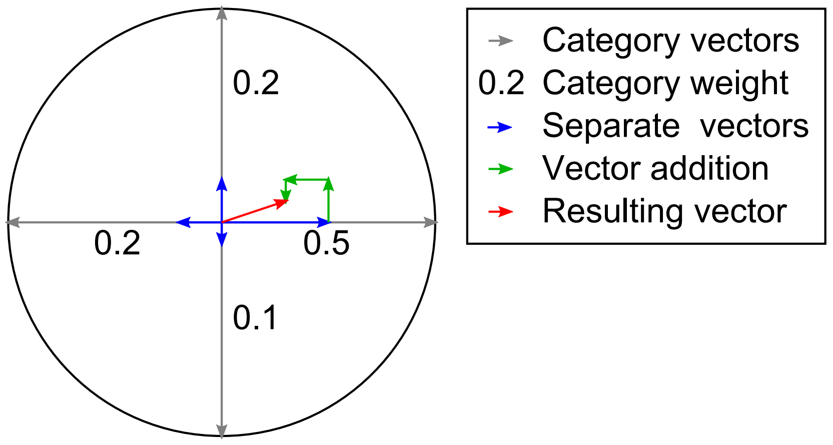


Figure 2.8: The composition of the intended position of a word in a RadCloud with four categories.

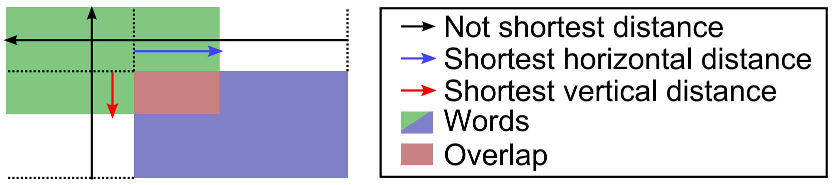


Figure 2.9: The shortest movement necessary to separate two words.

that corresponds to the word's overall importance instead of category-related relevance. There are multiple ways to calculate such a global weight w_i. Here are two examples from which RadCloud uses the latter one:

$$\begin{aligned} weights_i &:= \{w_i^1, \ldots, w_i^{n_c}\} \\ w_i &:= \sum weights_i \end{aligned} \tag{2.9}$$

$$w_i := max(weights_i) \tag{2.10}$$

When two words overlap, they should move away from each other by the minimum possible distance so that they only touch. Figure 2.9 shows how this minimum distance is calculated. However, the distance is not the only criteria by which a new location should be assessed. As the direction in which a word moves relative to the center is semantically relevant, it should not change too much. Therefore a value is calculated, that represents the new position's entropy and depends on both the distance and angle in which the word is moved.

$$
\begin{aligned}
v &= \text{possible avoidance movement}\\
d &= |\vec{v}| \in \mathbb{R}\\
\alpha &= anglebetween(\vec{v}, \overrightarrow{intendedPosition}) \in \mathbb{R} \wedge [0,180]\\
a + b &= 1\\
entropy &= a\frac{d}{cloudRadius} + b\frac{\alpha - 90}{90}
\end{aligned}
\tag{2.11}
$$

When a movement is applied, it does not dislocate a single word by the entire distance, but distributes it to *both* involved words. The individual movements of each of them depend on the avoidance spring: the harder the spring, the shorter the distance. This way, the sum of the tension of both springs is minimal, which means the forces are also locally minimal while the words are in equilibrium and still readable.

2.2.2.3.2 Clustering for performance

Assuming every word would always be handled individually with its avoidance spring, there would arise performance problems. These would be due to the further insertion of words. The newly added word A could overlap with the existing word B. The algorithm would move both words to new positions. At this point both, A and B could overlap with other words, and so on. A cascade of overlaps would be the result, and this would lead to a significant loss of performance.

A possible solution to the relocation cascade is the clustering of words. After a normal avoidance movement the words will still touch each other. These touching words can be merged into a single rigid cluster. If this cluster is involved in an overlap, it moves as a whole and preserves the relative positions of the contained words. If a cascade of relocations is triggered, the participating elements are clusters. Since clusters contain at least one word, they are fewer in number and the total amount even decrease with each avoidance movement: the performance problem is solved.

However, the usage of clusters also means, that the overlap avoidance gets more complex. In RadCloud the clusters are represented by their shape, which is a polygon that can have holes. This means, that it is not only a list of rectangles, but also a list of lines along the perimeter that define the covered area. As algorithm 2.5 explains, the test for overlaps is first approximated by checking the bounding boxes and later algorithm 2.6 is used to check for the inclusion of a point in the cluster. The jordan curve algorithm is used as a base here, but since this polygon only uses horizontal or vertical lines, it can be adapted to create algorithm 2.6. The functionality depends on the lines for the clusters being created in predefined directions. This allows to determine which side of the line is inside and which one is outside the polygon. RadCloud always creates the lines clockwise around the rectangles. This means that top lines go right, right lines go down, bottom lines go left and left lines go up.

Algorithm 2.5 Checks for overlaps between a cluster and a word

```
function OVERLAP(cluster cluster, rectangle word)
    if cluster.BoundingBox overlaps with word then
        return false
    end if
    if cluster.Lines intersect with word then
        return true
    end if
    if cluster contains word.Center then
        return true
    end if
                        // Each cluster stores the center of its first word as a contained point
    if word contains cluster.ContainedPoint then
        return true
    end if
    return false
end function
```

Algorithm 2.6 Checks whether a point is contained in a cluster

```
function CONTAINS(cluster cluster, point point)
    lines ← {line: line ∈ cluster.HorizontalLines ∧ line.Y < point.Y }
    SortFromTopToBottom(lines)
    isInside ← false
    wasForward ← nil
    for all line ∈ lines do
        if line.Left = point.X ∨ line.Right = point.X then              // At the edges
            if wasForward = nil then
                wasForward ← line.IsForward          // left to right or top to bottom
                isInside ← ¬ isInside
            else if wasForward ≠ line.IsForward then
                wasForward ← nil
                isInside ← ¬ isInside
            else
                wasForward ← nil
            end if
        else if line.Left < point.X ∧ line.Right > point.X then         // In the middle
            wasForward ← nil
            isInside ← ¬ isInside
        end if
    end for
    return isInside
end function
```

If a new word does not overlap, a new cluster is created for this word. If it overlaps with one or more existing clusters, it is merged with the one that will place the new word with the least entropy. To find this new position, every cluster calculates where the new word could be placed in a merging operation. The combined list of all positions is then sorted by entropy. Using a backtracking technique, these possible solutions are tested, by placing the new word and resolving subsequent merges between clusters as well as reinsertions (see 2.2.2.3.4). If a proposed position would cause unresolvable problems, it is discarded and the next best solution is tried.

During the process of merging, the involved elements' springs to the intended positions are removed. Instead, a new spring to the actual position is added. Which means, that it is not elongated, but will exert forces when resolving a new overlap. Its rigidity is equal to the sum of both previous springs:

$$\begin{aligned} &\mathit{rigidity}(\text{new}) := \mathit{rigidity}(\text{cluster's spring}) + \mathit{rigidity}(\text{word's spring}) \\ &\mathit{rigidity}(\text{new}) := \mathit{rigidity}(1^{\text{st}}\text{ cluster's spring}) + \mathit{rigidity}(2^{\text{nd}}\text{ cluster's spring}) \end{aligned} \tag{2.12}$$

However small this movement is, it can still cause new overlaps. Thus the clusters have to be checked for overlaps between each other (see algorithm 2.7). These are then resolved by merging the clusters that need the least movement and then repeating the check with the merged result. This process is repeated until there are no more overlaps.

Algorithm 2.7 Check for overlaps between two clusters

```
1:  function OVERLAP(cluster $a$, cluster $b$)
2:      if bounds do not overlap then
3:          return false
4:      end if
5:      if ($a$ contains $b.ContainedPoint$) $\vee$ ($b$ contains $a.ContainedPoint$) then
6:          return true
7:      end if
8:      if $a.VerticalLines$ intersect with $b.HorizontalLines$ then
9:          return true
10:     else
11:         return false
12:     end if
13: end function
```

2.2.2.3.3 Merging

To this point, collision avoidance is still relatively abstract, because the merging has not been explained yet. When a word is to be merged into a cluster, possible movements along the outer lines, that don't lead outside the circle are then determined and stored in a sorted list (see algorithm 2.8): the one with the lowest entropy is at the beginning. This list is then traversed sequentially to simulate the initial movement and all movements that result from newly created overlaps. The first position with a possibility to place the word without overlaps or overdraws outside the circle is then selected as final avoidance movement).

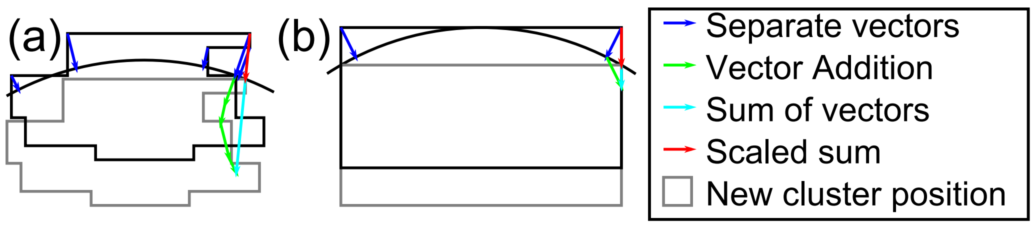


Figure 2.10: How the vectors for reinsertion are determined.
(a) Vectors from each corner to the center are summed up and scaled.
(b) The sum is also necessary in a simpler case, to prevent movements that would lead outside and would require subsequent corrections.

If the algorithm does not find a possible insertion position it stops inserting words. The selected size for the circle was too small. Therefore, a minimal size is calculated for the circle radius. This calculation sums up the area of all words and multiplies it by a constant factor $f \geq 1$. Manual testing achieved satisfactory results with $f = 1.1$, but this value can be adjusted by the user. The minimal radius is then calculated by using the well known equation for the circle area:

$$\begin{aligned} a &= \pi r^2 \\ \sqrt{\frac{a}{\pi}} &= r \end{aligned} \tag{2.13}$$

2.2.2.3.4 Reinsertion
Words or clusters that go outside the circle can still be pushed back in as in figure 2.10. To achieve this the outside of the circle is modeled as being a soft body, while the clusters stay rigid and protrude into the surrounding shape. To approximate, how the flexible outside pushes against the rigid inside, all overdrawn corners get a vector to the center. These vectors are then cut at the circle's perimeter and their values are summed up, to get the direction of the force pushing the cluster inside. To get the amount of force, the vector with the inward movement is positioned at the point that goes most outside and then scaled to end on the circle perimeter. This method is not guaranteed to get the cluster inside the circle as it could protrude on the other side. Therefore the procedure is repeated until the cluster is completely inside or one movement goes back in the opposite direction as the previous movement. To check for this case, the angle between the vector of the previous and the current movement is calculated. If its absolute value exceeds a threshold, the movements are recognized as being opposed. The upper limit is set to 170 degrees and is user-adjustable. The same procedure is performed when a word is to be overlapped at a position that goes outside the circle, but does not overlap with clusters.

Algorithm 2.8 Gets all movements for the merging of a word into a cluster

```
function GetAllMovements(cluster $cluster$, rectangle $word$)
    movements                          // Sorted set of information about possible movements
    newWord                            // Used as temporary storage to calculate the vectors

    for all line ∈ $cluster.HorizontalLines$ do
        overlapsDistantly ← (line.Right > $word.Left$) ∧ (line.Left < $word.Right$)
        if overlapsDistantly then
            newWord ← word
            newWord.MoveVerticallyTo(line)
            movements.Add($\overrightarrow{\text{newWord.Position}} - \overrightarrow{word.Position}$)
        end if
        if (line.Left > $word.Left$) ∨ (overlapsDistantly) then
            newWord ← word
            newWord.MoveToLeft(line)
            movements.Add($\overrightarrow{\text{newWord.Position}} - \overrightarrow{word.Position}$)
        end if
        if (line.Right < $word.Left$) ∨ (overlapsDistantly) then
            newWord ← word
            newWord.MoveToRight(line)
            movements.Add($\overrightarrow{\text{newWord.Position}} - \overrightarrow{word.Position}$)
        end if
    end for

    Repeat analogously for vertical lines

    for all line ∈ $cluster.Lines$ do
        newWord ← word
        newWord.MoveToNearestEndOf(line)
        if newWord is not outside circle ∧$cluster$ does not overlap with newWord then
            movementInfo ← new MovementInfo()
            movementInfo.Movement ← $\overrightarrow{\text{newWord.Position}} - \overrightarrow{word.Position}$
            movementInfoeInfo.Entropy ← entropy according to equation (2.11)
            movements.Add(movementInfo)
        end if
    end for

    return movements
end function
```

When clusters are merged, it is often necessary to search for diagonal movement in oder to resolve an overlap. Because of the limited directions the lines of the cluster can go, a simple horizontal or vertical movement is less complex to calculate and therefore faster.
Algorithm 2.9 gets the minimal needed movement. For every line it is calculated how much it

protrudes into the other cluster. The maximum protrusion of top and bottom lines is then determined. The same procedure is repeated for right and left lines. The minimal overall protrusion is then selected as minimum necessary movement.

Algorithm 2.9 Selects the movement for the merging of two clusters

```
1:  function GetMovement(cluster $a$, cluster $b$)
2:      movements ← $\emptyset$
3:      movements.Add(GetUp($a$, $b$))
4:      movements.Add(GetDown($a$, $b$))
5:      movements.Add(GetLeft($a$, $b$))
6:      movements.Add(GetRight($a$, $b$))
7:      movements ← shortest(movements)
8:      if |movements| = 1 then
9:          return movements.First
10:     end if
11:     angles ← $\{\alpha_i \colon \alpha_i = AngleBetween(\text{movements}_i, \overrightarrow{b.Bounds.Center}\}$
12:     movements ← $\{m_i \colon m_i \in \text{movements} \wedge \alpha_i = \min(angles)\}$
13:     return movements.First
14: end function
15:
16: function GetUp(cluster $a$, cluster $b$)            // Down, left and right work analogously
17:     overlap ← $+\infty$
18:     for all aLine ∈ $a.HorizontalLines$ do
19:         if ¬aLine.IsForward ∧ aLine.Y > b.Bounds then
20:             for all bLine ∈ $b.SortedHorizontalLines$ do            // From top to bottom
21:                 if (bLine.IsForward ≠ aLine.IsForward) ∧
                       (bLine.Left < aLine.Right) ∧ (bLine.Right > aLine.Left) then
22:                     overlap ← min(overlap, bLine.Y - aLine.Y)
23:                     break          // The other lines are lower, therefore the distance is less
24:                 end if
25:             end for
26:         end if
27:     end for
28:     return $\begin{pmatrix} 0 \\ \text{overlap} \end{pmatrix}$
29: end function
```

2.2.2.3.5 Cleanup

The described method of overlap avoidance works, but also has drawbacks. When a word A is placed but overlaps, it is merged into a cluster with other words. The cluster can then move as part of later avoidance movements and leave A's originally intended position clear again. However, since A already is in the cluster, it will move away with it and increase its distance from the intended location.

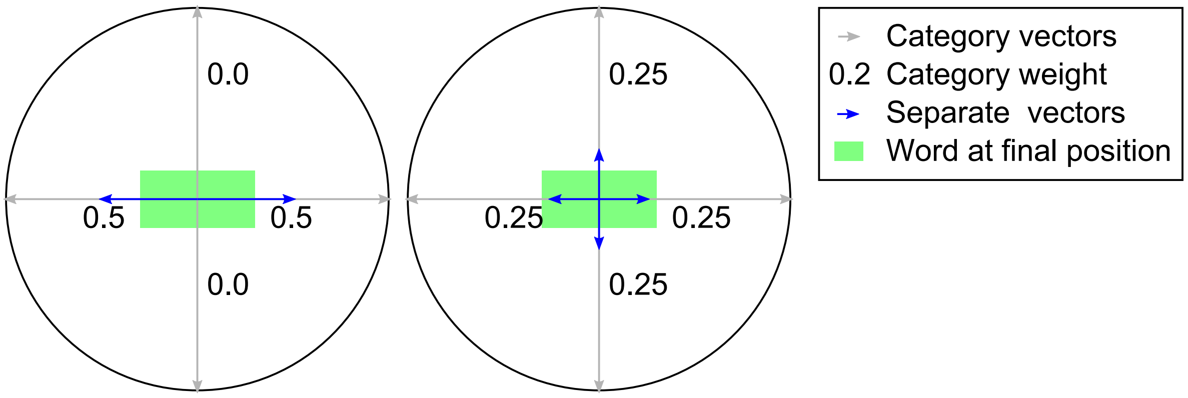


Figure 2.11: Example of the ambiguity of the positions in RadCloud (and also RadViz): two words at the same place but with different weights.

For this reason, the RacCloud employs a second stage in which the clusters are dissolved and every word gets checked individually to see whether its intended position has been cleared. To do this, it is not sufficient to loop once over all words, because there could be a cascade of cleared spots. Thus the loop has to be repeated until no more changes occur.

2.2.2.4 Possible improvements

While the RadCloud produces usable visualizations, it has room for improvement. Figure 2.11 explains how the words' positions suffer from an ambiguity when more than three categories are displayed. The difference between multiple possible interpretations of a word's placement could be decreased by rearranging the categories. For this, a new relation $\mathcal{R}_c$ would have to be defined on the set of categories $\mathcal{C}$ (see equation (2.6) on page 24). Categories with a high weight in their relations would be placed next to each other:

$$\begin{aligned} \mathcal{R}_c &:= \{(c_i, c_j)\colon c_i \in \mathcal{C} \land c_j \in \mathcal{C}\} \\ weight((c_j, c_k)) &:= \sum w_i^{c_j} + \sum w_i^{c_k} \qquad \text{see (2.6)for the definition of } w_i^c \end{aligned} \tag{2.14}$$

This and the two other ways of optimizing the RadCloud have been thought of, but ultimately dismissed.

2.2.2.4.1 White space reduction

Non-meaningful white space in a visualization is an indicator for room for improvement. The space in RadCloud has its purpose of showing the absence of words with certain weight distributions. However, the existing words could be rendered with a bigger font size to improve the readability. Since the final distribution of white space can not be determined before all words have been placed, a change in font size causes a new execution of the layout algorithm. Furthermore, there is no known way of determining the amount by which the size can be changed without causing an overflow of words outside the circle. Therefore, the optimal font size would have to be found by making incremental changes, updating the layout and checking the bounds. This would increase the runtime of the layout algorithm, thus reduce the interactivity, thus decrease the usability of the tool.

2.2.2.4.2 Entropy reduction

Moving the categories into different positions can change the amount and size of overlaps as well as the overall degree of ambiguity.

The more severe (i.e. big) overlaps occur, the more the words move away from their originally intended positions. Using the avoidance springs, the system's entropy can be calculated. A different arrangement of the categories could result in less collisions of important word, therefore reducing the entropy. However, as with the white space optimization, this would need an execution of the layout algorithm for every unique combination of category placements.

Because of the representation in a circle, the unique combinations are limited compared to the possible permutations of characters. For example the set $\{a, b, c, d\}$ with $4! = 24$ permutations can be arranged as *abcd* or *bcda*. While both permutations are different, in the circle they would just be a rotation of the same arrangement. The same goes for a reversed order as with *abcd* and *dcba*: it would only be an axisymmetric transformation. According to Sloane this amount of unique permutations would be 1 for up to 2 categories and $\frac{1}{2}(n-1)!$ [Slo73]. For the current example this would result in 3 permutations but it still depends on the amount of categories and increases rapidly.

2.2.2.4.3 Multithreading

There are parts in almost all algorithms, that can profit from multithreading on today's multicore processors. These parts are often so small, that creating an entirely new thread would cause too much overhead. This would ultimately result in exactly the opposite effect and therefore affect the performance negatively. By using thread pools the small parts could still be calculated in parallel without having to create new threads. However, the added complexity while debugging would slow down development of MuWoC. In conjunction with the limited time frame for the thesis, this optimization was implemented in some parts of the algorithms but disabled during development.

3 Visual analytics system

3.1 Introduction

Visualizations only display data, they do not retrieve it. A visual analysis tool for textual data was created, based on the word clouds from chapter 2. This tool would extract words files and show them in either separate clouds or merged in the RadCloud. Furthermore it lists named entities that were previously recognized and also recognizes them itself. To show the original context of the single words or entities, it displays surroundings and entire documents. At the end of development the result was *MuWoC*.

To obtain visible word clouds, MuWoC uses four main phases: the recognition of named entities, the extraction of words, the determination of a word's relevance in a category and the final visualization.

3.2 Data sources

The support for multiple source formats is part of the effort to make MuWoC a versatile and useful visual analytics tool for textual data. However, since texts are the main intended form of input, there has to be a way to distinguish documents from different categories. These categories are the base for the creation of multiple word clouds. Without the information about the belonging to categories, there would only be one cloud. To create separate ones, a support vector machine[VC74] (SVM), could be trained to classify the documents. This would take a training set, and manual labor for an initial classification. Since there already are other tools - for instance WEKA [HFH$^+$09] and RapidMiner [MWK$^+$06] - to classify texts, MuWoC does not implement such a feature.

For small, independent data sets a manual categorization is faster because of the lack of subsequent steps like the training and application of software tools. For large data sets, the total amount of work could be reduced, but that is not MuWoC's development goal. Therefore, the source data is required to already be categorized and separated into different directories with the category names. The user then selects a root directory that contains the folders with the source files. All files that are direct children of the root are ignored. The same goes for subdirectories of the categories and files with unknown extensions.

3.2.1 File formats

MuWoC uses the .NET Frameworks composition feature to implement a plug-in system that allows for multiple input file formats.

3.2.1.1 TXT

Style information beyond the text itself is subject to the author's environment and personal preferences. It could be interesting to create profiles from the style information and analyse texts for their sources. However, that is not MuWoC's goal. Therefore, only the text itself and its semantics are used as input.

In computer operating systems plain text files are usually stored with the extension *TXT*. Their character content can be stored in different encodings. The 8 bit Universal Character Set Transformation Format (UTF-8) is an often used encoding that supports a wide variety of characters and is compatible to the first 128 indices (0-127) of the American Standard Code for Information Interchange (ASCII). Therefore MuWoC uses this format for all in- and output. While more encodings could be supported, it was not deemed less important than other features as the user can always use third party tools to transform the files from and to different encodings.

The Windows operating system sets restrictions on the length of file names and their paths. When the .NET Framework's file in- and output methods are used, these restrictions still apply. However, the operating system also contains native libraries that allow access to files in locations that exceed these limitations. Peter Palotas' and Alexandr Normuradov's open source library *AlphaFS* [PN13], allows .NET-Programs to make use of the native functionality in an easy way, without requiring special knowledge of the underlying architecture. MuWoC uses AlphaFS for all file in- and output, to enable users to store their files without the 260 character limitation on their paths.

3.2.1.2 NER and TXP

As for instance Wordle[Fei13] shows, the analysis of plain text suffices to get a term frequency value and produce word clouds. For a deeper understanding of the analyzed data it may be necessary to perform more complex tasks to get more information. A powerful means to gather further information is the named entity recognition.

A plug-in was implemented that can read custom files created during MuWoC's integrated entity recognition (see section 3.2.2). As visible in the sample in figure 3.2 the file is structured as character separated values (CSV) and uses semi-colons to separate columns. The ends of sentences are not marked in a dedicated column that would have to appear in each row, but by leaving blank lines. To distinguish the files from other input sources, the extension was selected to be "NER".

```
"PERSON";"Adloff"
"O";"."

"O";"He"
```

Listing 3.1: Sample output in the NER format.

```
# FILE: ./Actinium.txt
# FIELDS: token sentence      pos     entity
Adloff.  <eos>  SPN    B-PER
He       -      SPN    O
```

Listing 3.2: Sample output in the TXP format.

More adjustable and powerful named entity recognition requires third party software. This enables the use of specially trained models. For instance the Center for the Evaluation of Language and Communication Technologies' tool *Evalita* ([LT12]) can be used to preprocess text files and provide additional information on Italian texts. Evalita's output is stored in files with a TXP extension. Internally the files also uses CSV, but with horizontal tabulators, as shown in listing 3.2. In the header, there are a relative path to the source file and the names of the used columns:

token A single token from the source file.

sentence Marks the ends of sentences with the value "<eos>".

pos (Part Of Speech) Is not used in MuWoC.

entity The type of entity to which the token belongs.

As other commonly used tools, Evalita supports four types of entities: persons (PER), organizations (ORG), locations (LOC) and geo-political entities (GPE). The first token of an entity is prefixed with "B-", subsequent ones with "I-". If a token does not belong any entity, it is marked as being of type "O". MuWoC already contains a plug-in to read Evalita's preprocessed files.

3.2.1.3 CSV

The previous file formats all represented source texts, with or without information about entities. To make MuWoC not only an analysis, but also a visualization tool, more formats have to be supported. A simple CSV file, like in listing 3.3 can contain all information about the words to display. For this each row contains the word as first value and then its weight in each category. To store the names of the categories for later display, a header is necessary. The first value would denominate the name of the word column, which is not necessary, therefore, it is kept empty. The subsequent values can contain any string. The header is masked as a

```
#;actinides;alkali;alkaline earth;lanthanides;poor;transition
actinium;0.0340919601439586;0.000680500025109322;0.000546382596031954;0;5.78873747545298E-05;0
```

Listing 3.3: Sample output in the simple CSV format.

comment so incompatible programs will just ignore it. If a word has a weight of zero, it is assumed to not belong to the category. Negative weights are not allowed and are multiplied by -1.

3.2.2 Named entity recognition

Before plain text is used as a data source, the user can preproces it with a named entity recognizer. This introduces new information for later use.

The *Stanford Named Entity Recognizer* [The13] from the Stanford University's *Natural Language Processing Group* is a freely available tool for this task. It is natively implemented as Java software with a command line interface, but there are community provided bindings for other programming languages, too. These adaptations also allow the use of the software as a library.

As MuWoC is written in `C#` and uses the .NET Framework, it could not use the Java implementation directly. However, thanks to the *IKVM.NET* project [Fri13], there are a Java virtual machine for .NET and a reimplementation of the base class libraries. Java's compiled byte code can even be recompiled to .NET's intermediate language, so it works outside of the virtual machine and with the .NET Framework. Sergey Tihon has already performed these tasks and published the results [Tih13] as a *NuGet*[Out13] package.

The Stanford Named Entity Recognizer's website provides downloads for the Java implementation as well as three trained models for English texts:

All - 3 classes: Location, Person, Organization

CoNLL - 4 classes: Location, Person, Organization, Misc

MUC - 7 classes: Time, Location, Organization, Person, Money, Percent, Date

Furthermore they also provide a hyperlink to Prof. Dr. Sebastian Padó's website[1] on which he distributes two models for named entity recognition on German texts. These classifiers were created in cooperation with Manaal Faruqui [FP10] and trained on the CoNLL 2003 Shared Task German training set. Semantic generalization data was provided by use two corpora:

HGC - University of Stuttgart's "Huge German Corpus" [Uni13]

deWac - .de top-level domain "web as corpus" [BBFZ09]

[1]http://www.nlpado.de/~sebastian/software/ner_german.shtml

After the Stanford NER has processed the input files using one of the provided models, the results are stored in files with the format described in section 3.2.1.2. Thus, there be technical computer problems during a later phase or the user accidentally makes wrong input, the work is not lost.

3.2.3 Data extraction

MuWoC can not use the plain text or preprocessed files directly to show word clouds, but has to perform further analyses on them. The main result is a set of words with their respective weight in every category. Keeping this information in memory can strain the available resources of the executing computer, especially if the source data is large.

3.2.3.1 Dimensionality reduction

A dimensionality reduction can decrease the amount of necessary memory and shorten the processing time. However, this reduction also means a loss of information. MuWoC tries to keep the more relevant part by only removing what is deemed as unnecessary. To achieve this task, every word goes through two stages and four filters before the main metric calculation.

3.2.3.1.1 Word recognition

The first stage is the recognition of words. For this the source text is read without any filters. Depending on the input format, single lines, or entire files are cached in memory and then evaluated to obtain the words. The evaluation can be the parsing of the CSV columns or the application of a regular expression on the entire string:

```
\b\w+\b
```

Let the resulting set of words be called W.

Additionally to W, MuWoC also creates an index on the ends of sentences. While the NER and TXP formats contain this data by default, plain text files do not. Therefore they have to be recognized separately. The implementation makes use of another regular expression to detect ends of sentences:

```
(((?<=\w{2,})\.)|\.{2,}|(?:(?:!|\?|\:)+))(?=\s|$)
```

Since it is customary to leave a space after punctuation, the expression requires it to recognize ends of sentences. Another assumption is made with the period. When it is used in the context of abbreviations, there are less characters before it than with the words that were written out completely. Therefore if the period appears after less than two word characters, it is ignored. Should it however appear more than once, it will be seen as an end of sentence. The English and German languages also contain abbreviations with two characters, such as “Mr.” and “Dr.”. Since the regular expression is contained in the TXT file plug-in and not part of MuWoC’s localization plug-ins, it has to maintain some cultural independence and therefore does not take such special cases into account. The gained information on ends of sentences can later enable more options in the interactive part of the program.

3.2.3.1.2 Filtering

The second dimensionality reduction stage contains four filters that create a subset W' of W. Their goals is to remove probably irrelevant words, so that their weight won't be calculated.

- **Predefined stop words**
  As section 3.3.1.4 explains, MuWoC provides multiple lists of stop words.
- **Minimum word length**
  A word that is shorter than the minimum allowed length is not plausible. Many search algorithms for internet forums also use this technique to ignore search patterns that would yield too many results.
- **Maximum word length**
  Words that longer than the maximum allowed length are unlikely. For every language the words do not normally exceed a certain length. The ones that do, are very rare and probably won't be important.
- **Maximum number of equal consecutive characters**
  A word that contains more than the maximum allowed number of equal consecutive characters is not plausible. Many languages have words with double letters. Some, like the German language, do even allow triple letters. If a word has even more equal consecutive characters, it is probably an error in the input. Such faults can for instance occur when optical character recognition (OCR) is used to digitize documents.

The predefined stop words and default values for the other three criteria are provided as plug-ins. They support the localization independently from the source format, while keeping the user in control.

3.2.4 Relevance metrics

Only after a word has passed all filters, is it added to W' and will be available for the calculations that provide a metric to determine its relevance.

3.2.4.1 Term frequency (TF)

Wordle uses the overall term frequency to measure the importance of words [Fei10]. This assumes, that the most relevant words are also the most frequently used ones. While this approach works for single documents, there are caveats for the input used for MuWoC. Since there are multiple categories, there are also multiple documents. Assuming there are two documents A and B, and that A is much longer than B. If A contains a relatively unimportant word, it can appear more often than an important one in B. MuWoC still allows the user to select the TF as decisive metric for the word relevance, despite its shortcomings.

3.2.4.2 Term frequency - inverse document frequency (TFiDF)

To work around problems associated with the TF, the metric can be multiplied with the inverse document frequency. This takes into account in how many documents the word appears and changes its weight accordingly. The more documents contain it, the less important it is. Sebastiani even employed various other methods to weight words and select the most relevant ones in an effort of term space reduction (TSR) [Seb02]. However, implementing them all would not fit MuWoC's tight schedule. Instead, the TFiDF was selected but not as a means of TSR. MuWoC can use it as sole metric or as input for another step in the calculation of the importances.

To support the determination of the inverse term frequency, it is not enough for the extraction program to store a word's number of occurrences. It is also necessary to store in which file it appeared. If additionally to the source file name, the index in the document is also stored, later the neighborhoods of words can be determined and displayed to the user. Therefore MuWoC uses a dictionary to hold this information: the document's relative file path in respect to the selected root folder serves as key. A sorted list of indices of occurrences serves as value of the dictionary's entries.

3.2.4.3 Support vector machine

Instead of applying more metrics surveyed by Sebastiani, a new approach was implemented in which the TF and TFiDF serve as input for a support vector machine. Normally an SVM is used with textual data to determine the document's belonging to categories [Seb02]. But since the words are the features, the SVM could determine the most relevant ones in a category as part of its support vector. During normal operation with an SVM, it would first be trained and later the results would be used to categorize documents. However, to get the mentioned support vector the last step is not needed.

As mentioned in section 3.2, the input documents for MuWoC are already categorized by their file location. However, there still is one more problem: a support vector machine only works with exactly two categories. Fortunately, Sebastiani already described how to solve this issue. For n categories, the SVM is trained n times by labeling all documents from the targeted category c_i as belonging and all documents from other categories c_j as not belonging [Seb02].

MuWoC does not implement an SVM itself, but uses Mathew Alastair Johnson's SVM.NET [Joh09]. Johnson ported the original implementation of Chih-Chung Chang's and Chih-Jen Lin's [CL11] *libsvm 2.89* [CL09] to the .NET platform.

An SVM expects vectors that are labeled as belonging to one or the other category. MuWoC creates a vector for each document. The dimensions are set by getting the entire list of all extracted words, essentially creating a bag of words representation [Har81] over all documents. This leads to all vectors having the same dimensionality. The TF or TFiDF of a word in a document is then the value of the respective dimension. If a document is part of the category to be examined, it is labeled with a value of +1, all others are labeled −1.

$$\begin{aligned}
D &:= \text{set of all documents} \\
d &:= \text{document in } D \\
W' &:= \text{set of all words from all documents that passed the filters} \\
w &:= \text{word in } W' \\
m &:= |W'| \\
\mathit{tfidf}(x,y) &:= \text{the TFiDF of word } x \text{ in document } y \\
v_i &:= \Big[\ \mathit{tfidf}(w_1,d_i),\ \ \dots\ \ ,\mathit{tfidf}(w_m,d_i)\ \Big]^T
\end{aligned} \tag{3.1}$$

The importance of each word in each category is then calculated as the sum of values that the SVM yielded for the vectors:

$$\begin{aligned}
C &:= \text{set of all categories} \\
c &:= \text{category in } C \\
\mathit{category}(x) &:= \text{the category to which document } x \text{ belongs} \\
\mathit{importance}(w_j,c_k) &:= \sum (\forall v_i[j] : \mathit{category}(d_i) = c_k)
\end{aligned} \tag{3.2}$$

After the training, the quality of the SVM's model is assessed by performing a five-fold cross validation on the training data itself. The result is then displayed, so the user can select other settings in case of an unsatisfactory accuracy. This is an important feature, because the more accurate the SVM classifies the documents, the better it discovered the words that distinguish them. Since Hirao et al. [HIMM02] have shown that an SVM is effective in finding the most important sentences, it should also be capable of discovering the most important words.

3.2.5 Result format

Depending on the type of performed analysis (see section 3.2.3), the runtime can be relatively high. The results are therefore organized in projects and have to be stored in order not to force the user to recalculate them every time some files are opened or the program exits. Furthermore, when the output files are saved, the number of included words is reduced again. As not only the words themselves but also extra information is written to the file, it can grow bigger than the original source texts. Displaying so many words in clouds could lead to very poor performance or lack of computer resources. Therefore, the user can set a maximum number of words that will be exported for each category.

When the project is about to be saved, the extra information is removed. If the user would saves a file with a small amount of words and afterwards wants to create a larger version the full information still has to be available. Therefore MuWoC does not remove data, but creates reduced copies that are then written to the output destination. This also allows to use a variation of SVM settings and recalculate the word values, without having to perform the extraction every time.

The word reduction also allows to discard data on files, from which no information was retrieved. This reduces the file size and later also the memory footprint in the viewer. To find the superfluous data, a hash set with all names of files that contained the extracted words is created. Since the file lists are also stored as hash sets that belong to categories, the actual filtering is only a set operation.

For a simple extraction of words that are only to produce word clouds, the result can be written to a CSV in the same format as for the input described in section 3.2.1.3.

3.2.5.1 Extended CSV

In the case the user wants to keep the extraction settings for later reference, the format has to be extended. The header now still starts with the names of the categories, but it does not end there. The extra meta information is stored right after the regular header:

- The type of plug-in that was used to read the source files.
- The type of plug-in that provided localization info.
- The minimum and maximum allowed word length.
- Whether numbers were allowed as input.
- How many words were selected to be saved per category.
- The used metric.
- The settings for the SVM.
- The average accuracy of the trained SVM.
- The accuracy of the trained SVM on each category.

As listing 3.4 show, all this information is again placed behind comments. Incompatible programs will just ignore it, while the user can still open the file in any text editor to get the information. MuWoC itself can use this data to adjust settings in the user interface so that only the neighborhood and source file functionality is missing.

```
#;actinides;alkali;alkaline earth;lanthanides;poor;transition
#ExtractorType;"MWC.DocumentAccess.NER.NerExtractor, MWC.DocumentAccess.NER, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"
#LocalizationType;"MWC.DocumentAccess.English.English, MWC.DocumentAccess.English, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"
#MinimumWordLength;3
#MaximumWordLength;50
#AllowNumbers;False
#MaximumWordsPerCategory;100
#Metric;SVTFiDF
#KernelType;Linear
#GridSearch;False
#SvmParamEpsilon;0.001
#SvmParamC;1
#SvmParamGamma;0
#AverageAccuracy;0.88008034026465
#CategoryAccuracy;0.91304347826087;0.891304347826087;0.826086956521739;0.880434782608696;0.858695652173913;0.880434782608696
actinium;0.0340919601439586;0.000680500025109322;0.000546382596031954;0;5.78873747545298E-05;0
```

Listing 3.4: Sample output in the extended CSV format.

3.2.5.2 MWCE

If all the gathered information is to be saved, a CSV format is not well suited for the task. MuWoC uses an XML file format and applies the extension “MWCE” to store all data in a single file that does not violate standards and is humanly readable. This data does not only include the previous format’s information, but also extends it.

The XML representation suits the object oriented paradigm better, and allows the writing and reading of the files to be integrated with the project’s serialization. The .NET Framework provides a default method for serialization and deserialization of objects. However, this method is very generalized and therefore produces XML tags without attributes. Instead, every object and property is written into its own tag. This representation produces relatively large output files, that can be bigger than the original source texts, even though the contents has been reduced as explained above. An additionally downside to the default object serialization is the lack of support for dictionaries. The projects use them to retrieve information about words, entities and ends of sentences more efficiently than with collections that use numbers as indices. Without this structure, the objects have to be organized in lists and have an extra property to hold the key by which they are referenced.

To solve the problems of output size and dictionary support MuWoC hooks into the XML serialization pipeline. It uses attributes instead of creating new tags for every simple property. When an object only contains one child collection, the encapsulating tags can be omitted. Lists of integers do not need a tag for every value, but can be serialized into a single string that is then stored as an attribute. When dictionaries are to be written, every entry can have its own tag that contains the key as an attribute and the value as a child element. Entries of simple types can be written into single tags that contain both the key and the value as attributes.

As the sample in listing 3.5 shows, the result is better readable for humans while its size is effectively reduced.

```xml
<words>
  <word>
    <value>element</value>
    <occurrences>
      <document>
        <file>\actinides\Actinium.txt</file>
        <indices>
          <index>5</index>
          <index>10</index>
        </indices>
      </document>
      <document>
        <file>\actinides\Americium.txt</file>
        <indices>
          <index>7</index>
        </indices>
      </document>
    </occurrences>
  </word>
</words>
```

```xml
<words>
  <word value="element">
    <document file="\actinides\Actinium.txt" indices="5 10"/>
    <document file="\actinides\Americium.txt" indices="7"/>
  </word>
</words>
```

Listing 3.5: .NET XML serialization. **Left** - default output. **Right** - custom implementation.

Using this XML format, every location of words, entities and ends of sentences can be stored, allowing for a navigable data in the UI (as described in section 3.3). It also allows storing which file belonged to which category and this in turn leads to the ability to link entities to categories. Without the extra information this would not be supported, because the entities are not assigned weights during the previous extraction.

3.3 Graphical user interface (GUI)

MuWoC has two major functions: extracting information from text sources and displaying it in word clouds. The graphical user interface reflects this fact by showing the start screen (figure 3.1). The user decides here whether he wants to analyze more texts or display existing projects. After the windows of either choice have been closed, the program returns to this start screen.

The start screen buttons contain icons from the Crystal Project [dlb07]. With the exception of the file viewer window, all other icons in MuWoC come from this set by Everaldo Coelho [Coe13].

3.3.1 Analysis wizard

If the user selects to proceed with an analysis, the program goes into a wizard like mode. In this mode, the settings are saved between sessions reduce the amount of necessary user input.

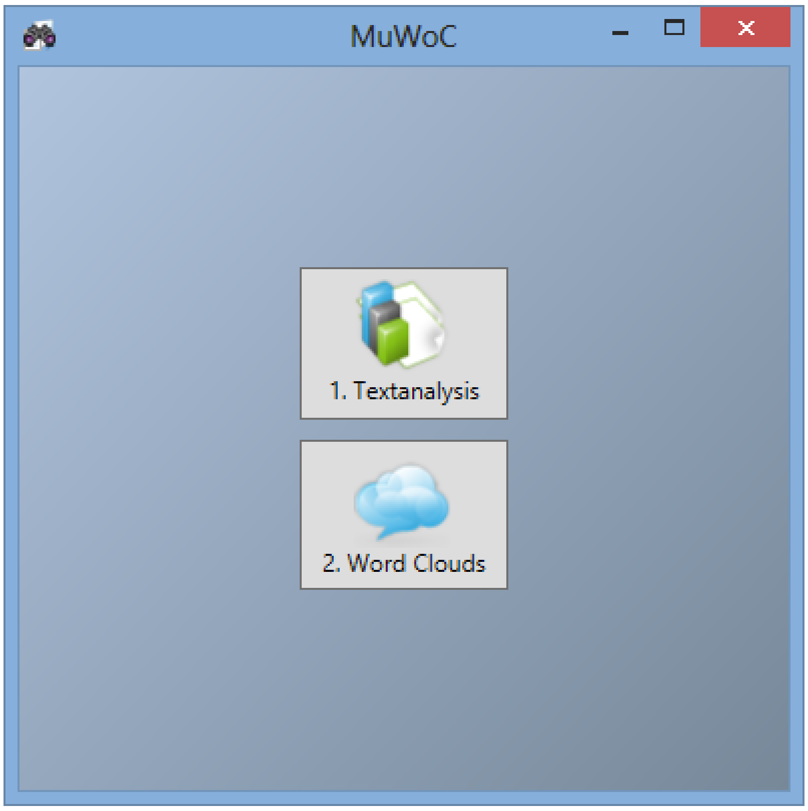
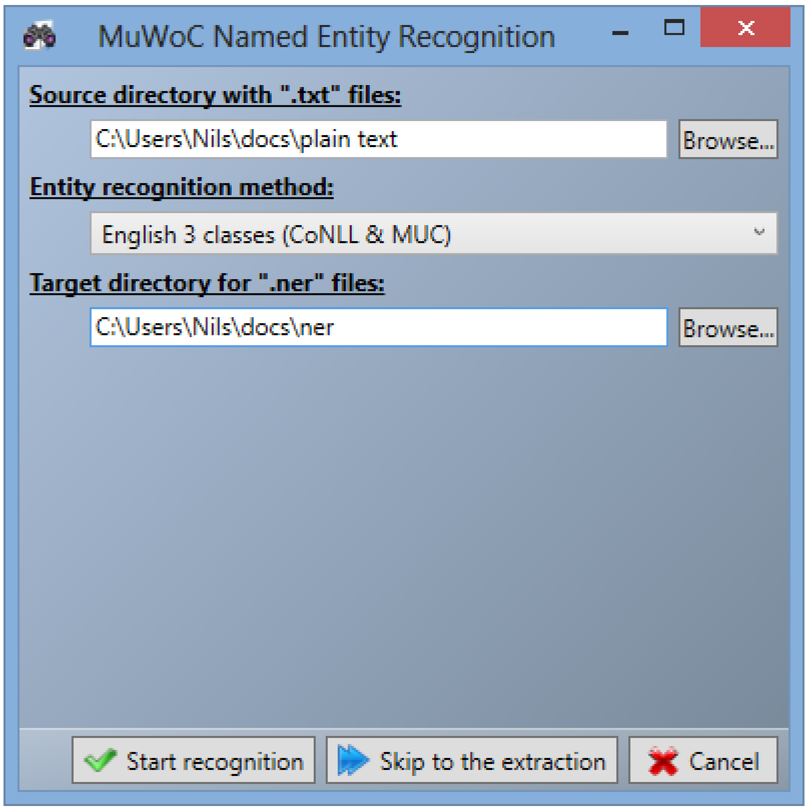


Figure 3.1: MuWoC's initial screen.

3.3.1.1 Named entity recognition

First, a window with settings for named entity recognition (see figure 3.2) is shown. The user can skip to the extraction window (see section 3.3.1.4) immediately, if he does not want to use the features that entities provide or if the input already includes this information. Should the user chose to perform the recognition, he has to select a method and a trained model to be used. To keep the user interface simple, these two choices are not separated into multiple interface controls, as the models normally don't work with more than one method. Therefore only one list is displayed. Every entry contains the name of the method and the type of model to use. By default MuWoC is equipped with plug-ins that use the Stanford Named Entity Recognizer [The13] with the models listed in section 3.2.2.

Further settings are the source and target directory for the files to be processed. The folder structure for the input has to be the same as for the extraction (described in section 3.2). The output should be an empty folder as the directories and files from the source are copied. To reflect the new format, the file names' extensions are changed to ".ner" by concatenating the previous name with the new ending. To avoid unwanted loss of data, existing files are not overwritten but produce an error message.



Source directory with ".txt" files: C:\Users\Nils\docs\plain text

Entity recognition method: English 3 classes (CoNLL & MUC)

Target directory for ".ner" files: C:\Users\Nils\docs\ner

Figure 3.2: MuWoC's NER settings window.

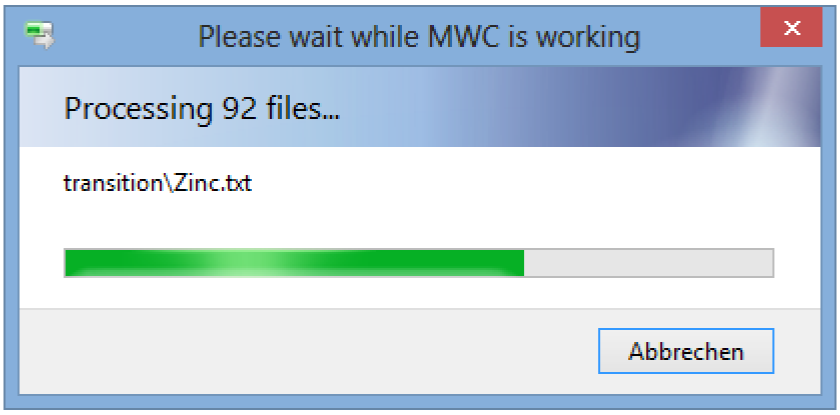
3.3.1.2 Progress window

The NER window is hidden while the recognition is being performed. A progress window like the example in figure 3.3 is displayed instead. It contains information about the current activity and the overall progress percentage. This window is provided by the Windows operating system, but has no direct access class in the .NET Framework. Sven Groot's[2] *Ookii.Dialogs* [Gro13] provides such a class. MuWoC wraps around this dialog, to make the display of the progress transparent to the running tasks. This way it is only necessary to call a method on the object containing the settings for the current task, to route the activity messages to the progress window.

3.3.1.3 Error window

Errors are handled similarly to the activity events. A method on the settings object notifies the progress window, which in turn accumulates the messages. Later they can be displayed using the error window. As the entity recognition can be a lengthy process, this window is important

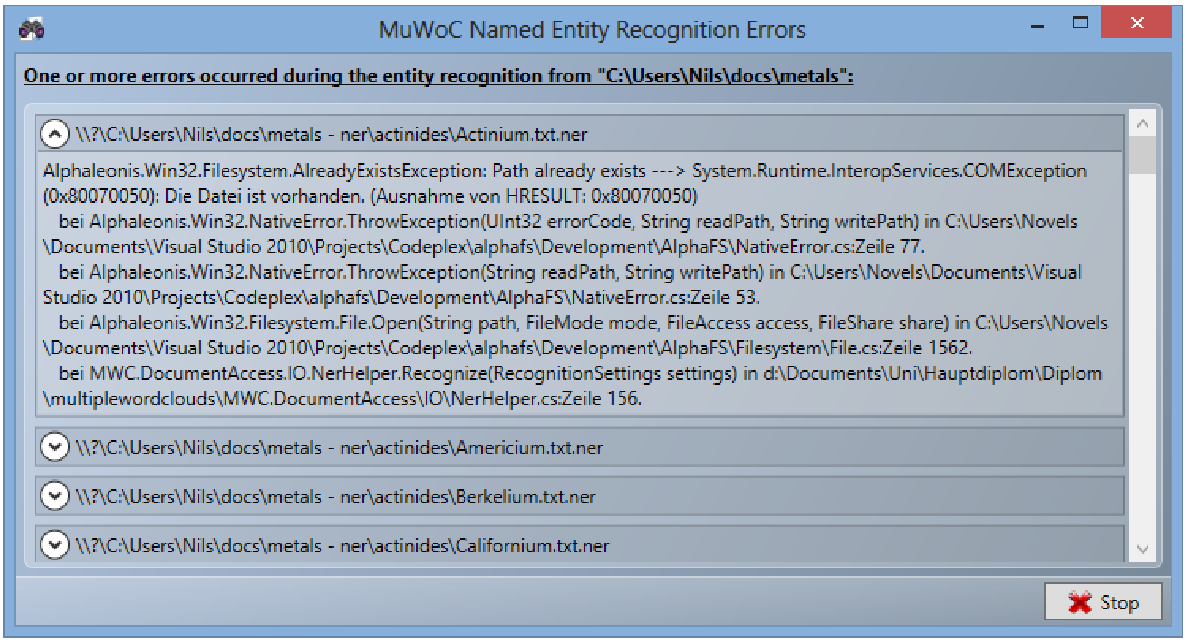
[2] `http://www.ookii.org/`


Please wait while MWC is working

Processing 92 files...

transition\Zinc.txt

Figure 3.3: MuWoC's progress dialog for lengthy tasks.

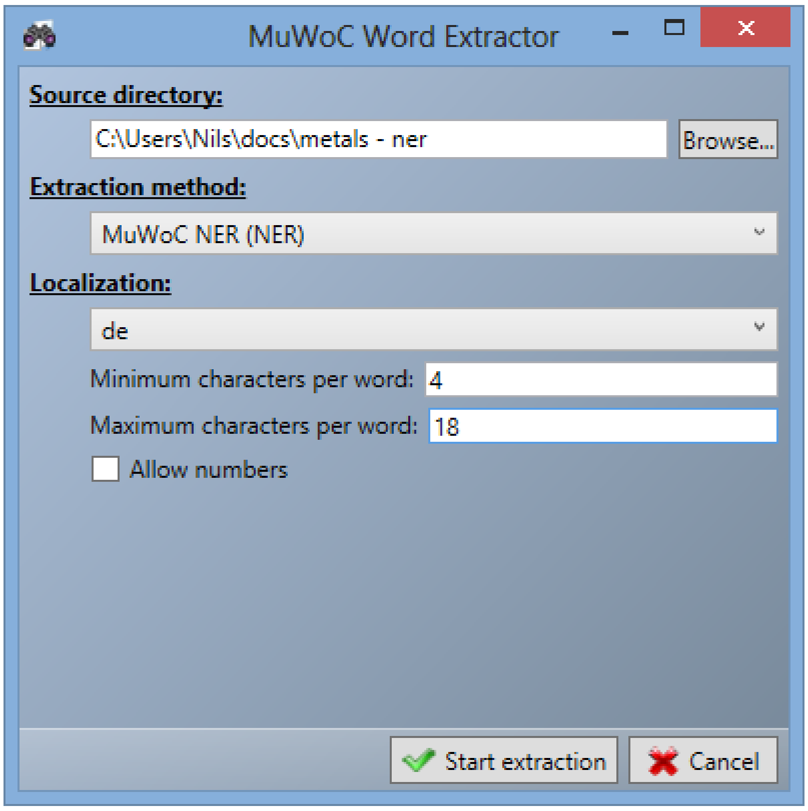

MuWoC Named Entity Recognition Errors

One or more errors occurred during the entity recognition from "C:\Users\Nils\docs\metals":

```
\\?\C:\Users\Nils\docs\metals - ner\actinides\Actinium.txt.ner
Alphaleonis.Win32.Filesystem.AlreadyExistsException: Path already exists ---> System.Runtime.InteropServices.COMException
(0x80070050): Die Datei ist vorhanden. (Ausnahme von HRESULT: 0x80070050)
   bei Alphaleonis.Win32.NativeError.ThrowException(UInt32 errorCode, String readPath, String writePath) in C:\Users\Novels
\Documents\Visual Studio 2010\Projects\Codeplex\alphafs\Development\AlphaFS\NativeError.cs:Zeile 77.
   bei Alphaleonis.Win32.NativeError.ThrowException(String readPath, String writePath) in C:\Users\Novels\Documents\Visual
Studio 2010\Projects\Codeplex\alphafs\Development\AlphaFS\NativeError.cs:Zeile 53.
   bei Alphaleonis.Win32.Filesystem.File.Open(String path, FileMode mode, FileAccess access, FileShare share) in C:\Users\Novels
\Documents\Visual Studio 2010\Projects\Codeplex\alphafs\Development\AlphaFS\Filesystem\File.cs:Zeile 1562.
   bei MWC.DocumentAccess.IO.NerHelper.Recognize(RecognitionSettings settings) in d:\Documents\Uni\Hauptdiplom\Diplom
\multiplewordclouds\MWC.DocumentAccess\IO\NerHelper.cs:Zeile 156.
\\?\C:\Users\Nils\docs\metals - ner\actinides\Americium.txt.ner
\\?\C:\Users\Nils\docs\metals - ner\actinides\Berkelium.txt.ner
\\?\C:\Users\Nils\docs\metals - ner\actinides\Californium.txt.ner
```

Figure 3.4: MuWoC's error dialog for the accumulation of messages during lengthy tasks.

to allow for an unsupervised program execution. If there would be no error messages, the user would have to manually check the output folder for faults. If the messages would be displayed in the operating system's default dialog windows, the process would be halted until the user dismissed the error. Non-blocking windows could be used to show the errors without hindering the progress. However, if for instance the user has no permission to write to the target directory, there is the potential for the creation of many message windows. The user would then have to close them all manually. Therefore, the error information is accumulated during the runtime of the recognition process and only displayed in a single window at the end (see figure 3.4).


MuWoC Word Extractor

Source directory: C:\Users\Nils\docs\metals - ner

Extraction method: MuWoC NER (NER)

Localization: de

Minimum characters per word: 4

Maximum characters per word: 18

☐ Allow numbers

Figure 3.5: MuWoC's extraction settings window.

3.3.1.4 Word extraction

The next window is for the extraction of the words and entities and can be seen in figure 3.5. The user should again provide an input folder with a structure as described in section 3.2. If the entity recognition was performed previously, its output is automatically selected as input, otherwise the user has to manually provide a folder path. To do this, there are two options. The path can either be entered as a string in an input box or selected via a folder selection browser. The .NET framework's default folder dialog only shows a tree visualization of the file system and does not provide a rich experience as with the file dialogs. However, the operating system is equipped with a corresponding window, that is not accessible by default. The *Ookii.Dialogs* library enables the use of the more user friendly dialog, while maintaining an application programming interface that is similar to the framework's default dialogs. The box for the string path remains editable independently from the input method.

Next an extraction method from the installed plug-ins has to be chosen. The TXT plug-in, provides only one option, while the others implement multiple varieties. The TXP and NER plug-ins share a common code base and allow two for the mixed use of two options:

Entities only

If this option is used, only words that are a part of entities are read. All others are ignored.

Entities as words

If this option is used, the tokens that are marked as entities are added as if they were normal words. Normally words do not contain spaces but using this option they can. For instance “United Kingdom” would normally be added as two words. If it is recognized as an entity, it would be added as single one.

To give the user the choice about which options should be enabled, these plug-ins appear four times, each with a different combination of the above.

To provide better support for different source text languages, MuWoC also uses plug-ins that help implement the four filters from section 3.2.3. The user can select the plug-in that best suits the input to achieve better results in the word extraction. By default, MuWoC comes with four plug-ins:

en

Sets the maximum equal consecutive character count to 2. Uses an English stop word list [Sav13a] published by Jacques Savoy on the university of Neuchâtel’s (Université de Neuchâtel)website [Sav13c] but also adds the “gotta” and “who’ll”.

de

Sets the maximum equal consecutive character count to 3. Uses an unmodified version of German stop word list [Sav13b] from the university of Neuchâtel.

it-IT

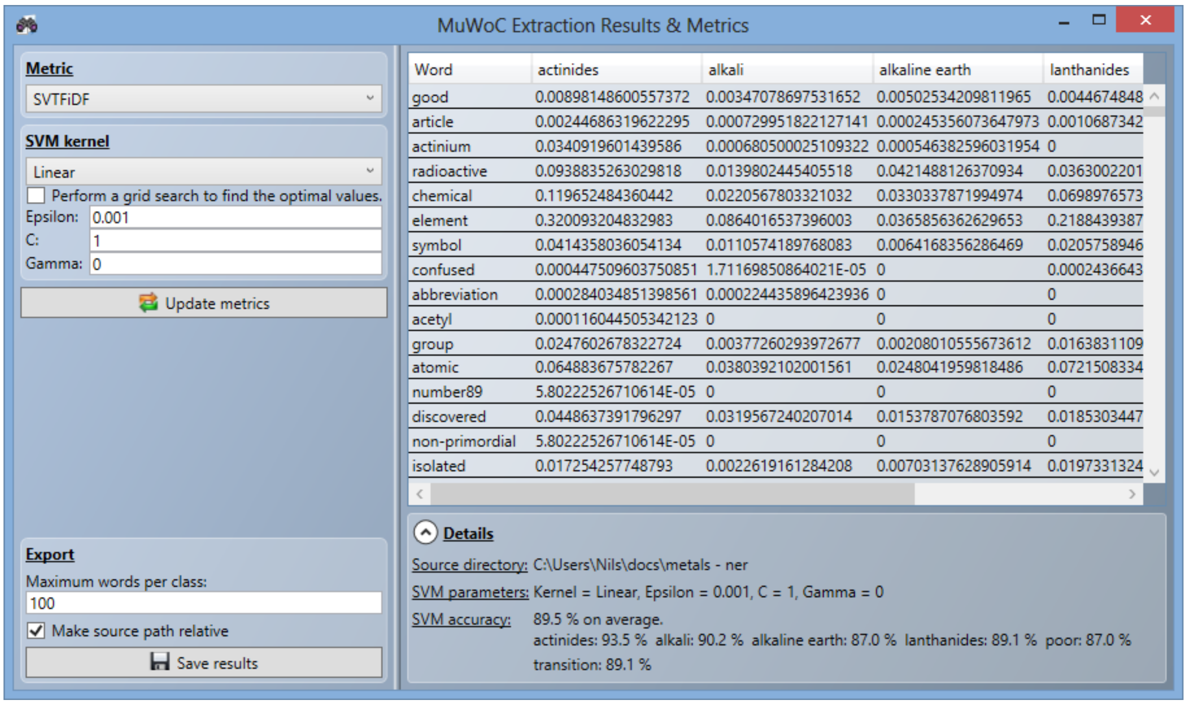
Sets the maximum equal consecutive character count to 2. Uses an adapted combination of the Italian stop word lists from the [Sav13d] university of Neuchâtel and the Google Code project “stop-words” [Bal11].

it-IT Legal

Extends the it-IT plug-in by adding stop words from the context of legal documents. This was added as an example of how the lists can be mixed and modified to allow more specific filters.

The stop word lists for all included plug-ins are stored in text files where every line contains a word. This way the user can adapt them as he sees fit. MuWoC has to be restarted for the changes to take effect.

A check box controls whether the words can represent numbers, while text fields serve as input for the minimum and maximum allowed word length as integers. All provided strings are validated before the user can start the extraction.



Metric: SVTFiDF

SVM kernel: Linear

☐ Perform a grid search to find the optimal values.

Epsilon: 0.001

C: 1

Gamma: 0

Maximum words per class: 100

☑ Make source path relative

Word	actinides	alkali	alkaline earth	lanthanides
good	0.00898148600557372	0.00347078697531652	0.00502534209811965	0.0044674848
article	0.00244686319622295	0.000729951822127141	0.000245356073647973	0.0010687342
actinium	0.0340919601439586	0.000680500025109322	0.000546382596031954	0
radioactive	0.0938835263029818	0.0139802445405518	0.0421488126370934	0.0363002201
chemical	0.119652484360442	0.0220567803321032	0.0330337871994974	0.0698976573
element	0.320093204832983	0.0864016537396003	0.0365856362629653	0.2188439387
symbol	0.0414358036054134	0.0110574189768083	0.0064168356286469	0.0205758946
confused	0.000447509603750851	1.71169850864021E-05	0	0.0002436643
abbreviation	0.000284034851398561	0.000224435896423936	0	0
acetyl	0.000116044505342123	0	0	0
group	0.0247602678322724	0.00377260293972677	0.00208010555673612	0.0163831109
atomic	0.064883675782267	0.0380392102001561	0.0248041959818486	0.0721508334
number89	5.80222526710614E-05	0	0	0
discovered	0.0448637391796297	0.0319567240207014	0.0153787076803592	0.0185303447
non-primordial	5.80222526710614E-05	0	0	0
isolated	0.017254257748793	0.0022619161284208	0.00703137628905914	0.0197331324

Source directory: C:\Users\Nils\docs\metals - ner

SVM parameters: Kernel = Linear, Epsilon = 0.001, C = 1, Gamma = 0

SVM accuracy: 89.5 % on average.
actinides: 93.5 % alkali: 90.2 % alkaline earth: 87.0 % lanthanides: 89.1 % poor: 87.0 %
transition: 89.1 %

Figure 3.6: MuWoC's metrics window.

3.3.1.5 Metric window

After all words have been extracted, the error window from section 3.3.1.3 is shown again. This step is skipped if there were no errors and the metric window is shown directly. Here the words are displayed in a grid (see figure 3.6). At the same time, the user can select the type of metric he wants to apply to give the words their weights. As already mentioned in section 3.2.4, the available options are term frequency, term frequency * inverse document frequency, the SVM's support vector on TF and on TFiDF.

If the user selects the SVM, there are more settings to adjust in order to get satisfactory results. MuWoC provides default values for these settings, that perform reasonably well in all use cases mentioned in chapter 4. SVM.NET's [Joh09] option to perform a grid search is also present in the UI. It tests various settings around a start configuration and automatically selects the values that should give good results.

The user starts the calculation of the metric by clicking a button. While the data is processed, the program displays a wait window from the *Ookii.Dialogs* library to signal that it is busy. The results are then loaded into the same grid as the words to make them immediately available for inspection. If the metric was calculated by an SVM, the used settings and the resulting accuracy of each class as well as the average are shown in an information area. This serves to help evaluate the quality of the results and as reference for new settings. Should the user not want this information, but rather check the word metric values manually, the information area can be collapsed to provide more screen space for the word grid.

Projects can be exported as soon as the first metric has been calculated. The user only has to set a maximum count of words that should be saved from each category. This setting is important to keep the export file sizes within reasonable limits and to restrict the amount of information that should later be displayed in the word clouds. By clicking the save button, the user opens the operating system's save file dialog and selects the format in which the project should be written. There are three options available that were explained in greater detail in section 3.2.5.

The main export format contains the path to the source files. In a multi-user environment, it can be helpful to store the path in relative form, so that the project can be saved alongside the sources. It is then irrelevant where the individual user stores the files, as the contained path still indicates where MuWoC can find the original input.

If the user desires to compare results between different types of metrics, it easily doable by updating the weights with another calculation method and re-exporting the results.

3.3.2 Cloud window

MuWoC can be registered with the operative system to be used as default handler for MWCE files. If a project is opened directly using this method, the start screen is omitted and the option of performing more analyses is not available. Instead the cloud window (see figure 3.7) with the opened project is shown. If the window is closed in this mode the program does not revert to start screen but shuts down instead.

MuWoC does not resemble a wizard when it is in visualization mode. This time around everything is centered on the cloud window as the root of all interactivity. Here the user can choose to open projects and display their clouds.

3.3.2.1 Separate view

By default, the cloud window starts in the separate view. Each category produces a its own word cloud and all clouds are displayed in a common layout panel.

3.3.2.1.1 Custom cloud layout panel

This panel uses a custom implementation for the layout of the clouds. It starts similarly to the framework's *WrapPanel* in that it lines up its child controls horizontally. As soon as the available width is exceeded, a new line beneath the previous one is started. This behavior can also be observed in the layout of words in texts. What makes the custom implementation different, is that it also tries to display its children as big as possible.
When Microsoft's WPF technology is used, every control has to calculate a *DesiredSize*. This also happens with the displayed word clouds. If there is more screen space available, than is necessary to display all clouds at their desired size, they can be scaled up in the render phase, so that they fill the available space. This has no impact on the layout itself as it is just a graphics transformation. The custom panel therefore checks the aspect ratio of the available

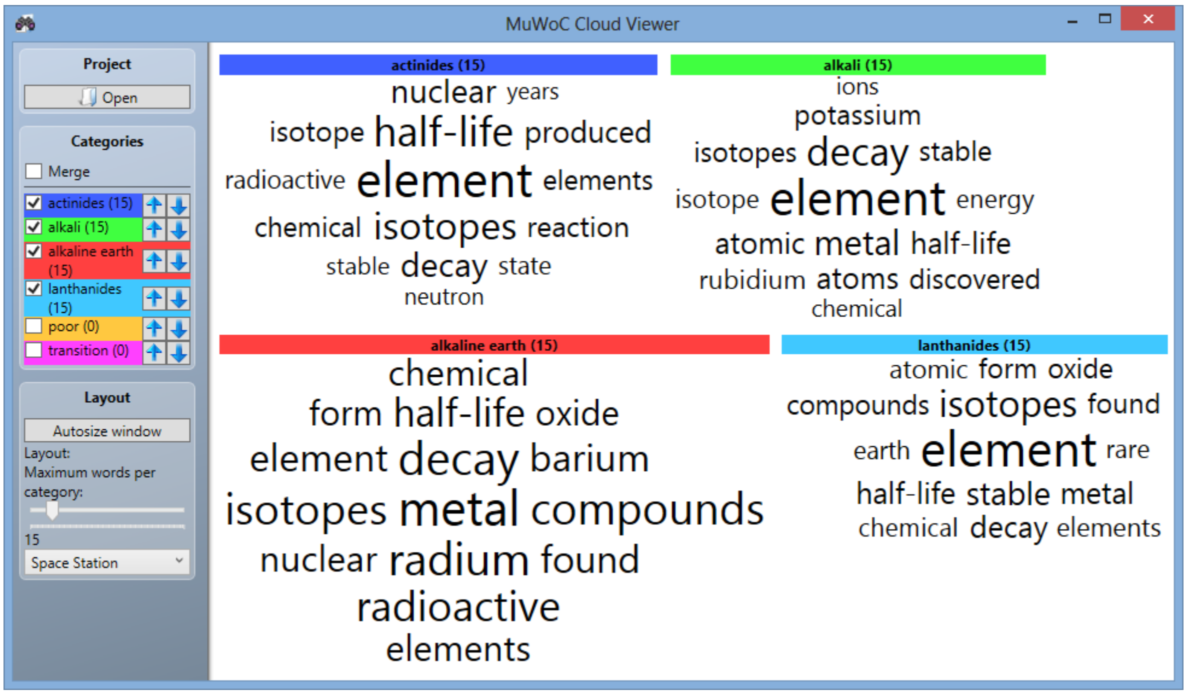

MuWoC Cloud Viewer
Project
Open
Categories
Merge
actinides (15)
alkali (15)
alkaline earth (15)
lanthanides (15)
poor (0)
transition (0)
Layout
Autosize window
Layout:
Maximum words per category:
15
Space Station
actinides (15)
nuclear years isotope half-life produced radioactive element elements chemical isotopes reaction stable decay state neutron
alkali (15)
ions potassium isotopes decay stable isotope element energy atomic metal half-life rubidium atoms discovered chemical
alkaline earth (15)
chemical form half-life oxide element decay barium isotopes metal compounds nuclear radium found radioactive elements
lanthanides (15)
atomic form oxide compounds isotopes found earth element rare half-life stable metal chemical decay elements

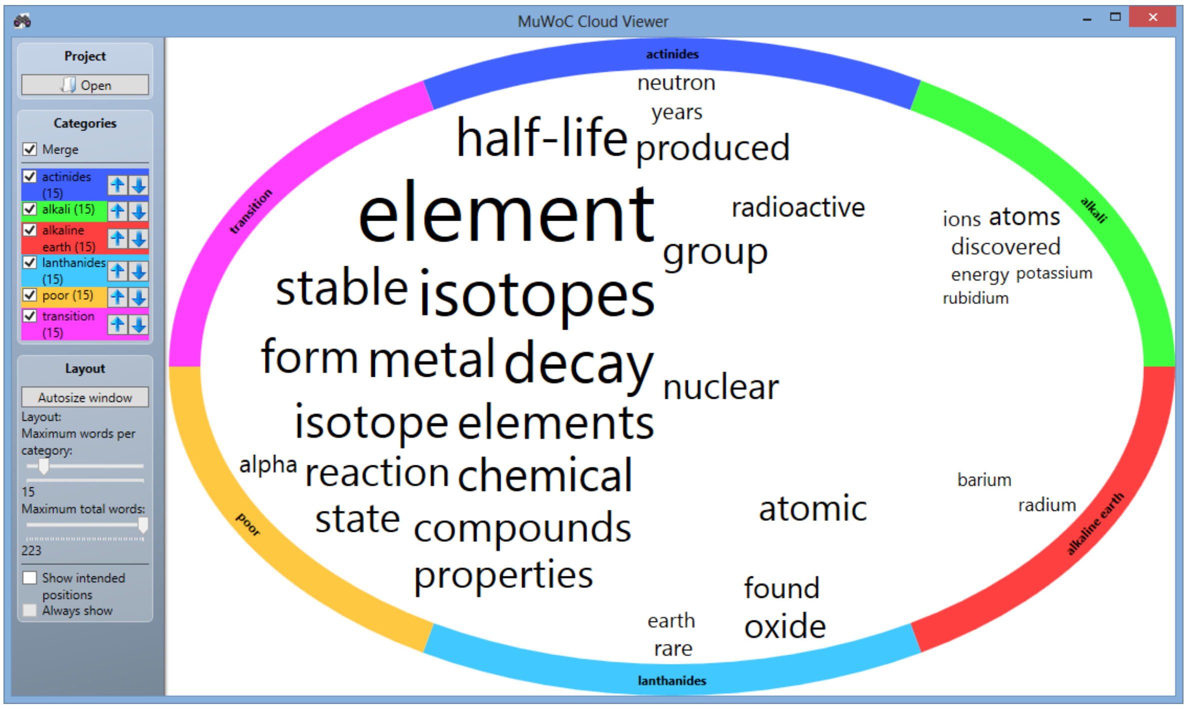
Figure 3.7: MuWoC's separate view. Two categories are hidden.

space and tries to place the children in such a manner, that the ratio is matched as closely as possible. The similar ratios then allow for greater zoom in the render phase. When the scaling of components is concerned, the use of WPF is a great upside as it is vector based and therefore does not introduce image artifacts when not drawing at the same size used during the layout.

3.3.2.1.2 Extra information and interactivity

As single word clouds are concerned, MuWoC display more information than just the visualizations themselves. Every cloud has its own title that contains the category's name and the number of displayed words. When the user hovers with the mouse pointer over a word, its relative importance in the cloud is shown as a percentage in a tooltip.

The placement algorithms used in the separate clouds work independently from the integration in the UI. This allows MuWoC to use existing, framework-provided user controls with their interactive features as hosts for the clouds. The *ListBox* was chosen to serve as such a container because it allows the selection of elements, but contrary to the *ComboBox* also their deselection. Further more, it supports the interaction with the mouse and keyboard and can also be synchronized with other controls that provide users with the ability to chose individual elements from a list. This enables MuWoC to use framework methods to implement brushing and linking across all visualizations: if the user selects a word in one cloud, it is also selected in all others. This provides the ability to quickly check for the occurrences of words in difference categories.


MuWoC Cloud Viewer

Project
Open

Categories
☑ Merge
☑ actinides (15)
☑ alkali (15)
☑ alkaline earth (15)
☑ lanthanides (15)
☑ poor (15)
☑ transition (15)

Layout
Autosize window
Layout:
Maximum words per category:
15
Maximum total words:
223
☐ Show intended positions
☐ Always show

Figure 3.8: MuWoC's merged view without coloration, category bar and intended position display.

3.3.2.2 Merged view

If the user wants to merge the separate word clouds into a single one, it is only necessary to check the merge option in the side bar and the program changes its mode with a result like the one in figure 3.8.

3.3.2.2.1 Layout

MuWoC then switches to the RadCloud (see section 2.2.2) as a unified representation. Since this visualization does not have a default, but only a minimum size, it can grow with the surrounding window. This allows for more words to be at their intended position because greater space also means less overlaps. At the same time, a visualization that adapts its size to the available area is not necessarily limited to a single aspect ratio. The RadCloud can be any ellipse, as long as its dimensions don't fall short of the estimated minimum width or height. That also means, that when there are only two categories, stretching the ellipse in one direction makes the visualization acquire some resemblance with Mason's *Comparative Word Cloud* [Mas12].

If the RadCloud is not provided with enough space to place a word it is not shown. Additionally all subsequent ones are omitted as well in order to avoid usage scenarios in which the user has to check whether specific words have been displayed. A red overlay in the upper left corner informs about the inability to place all elements and also contains the number of words

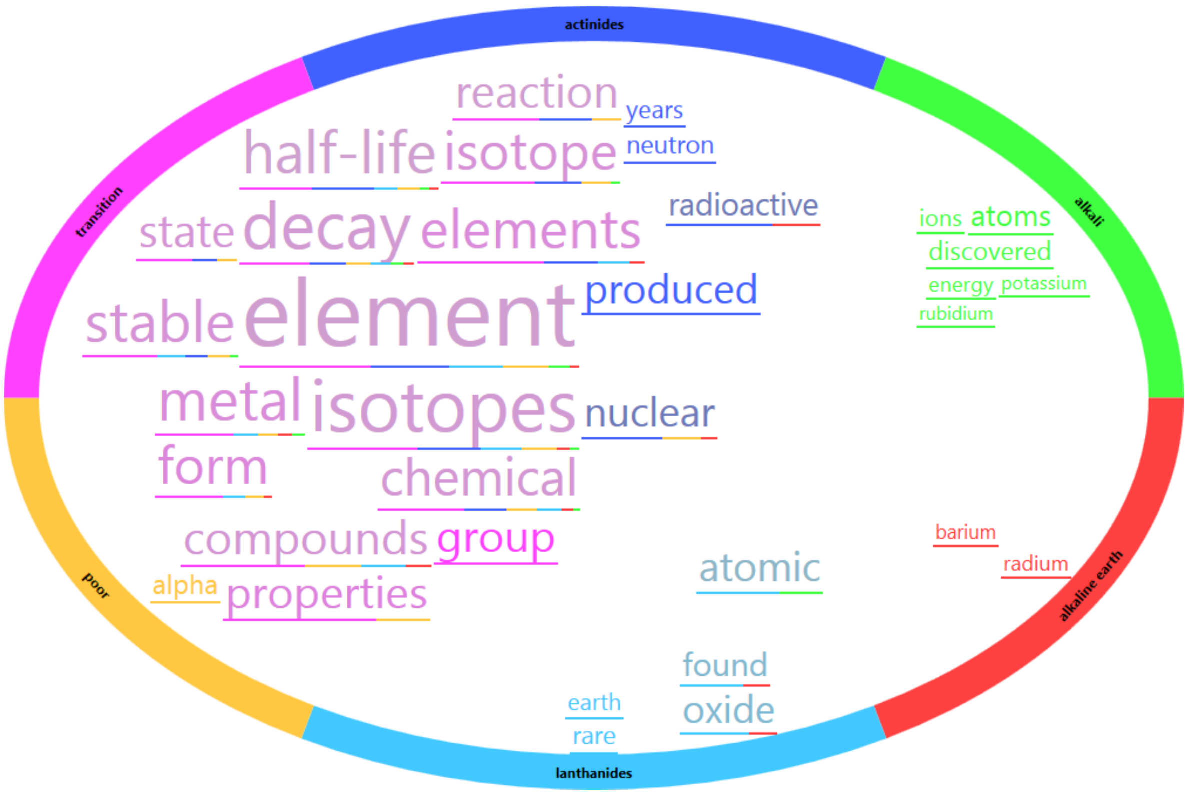


Figure 3.9: The merged view with word coloration, category bar and intended position display enabled.

that are not visible but should be. This information can be useful in combination with the interactivity features described in section 3.3.2.3.2.

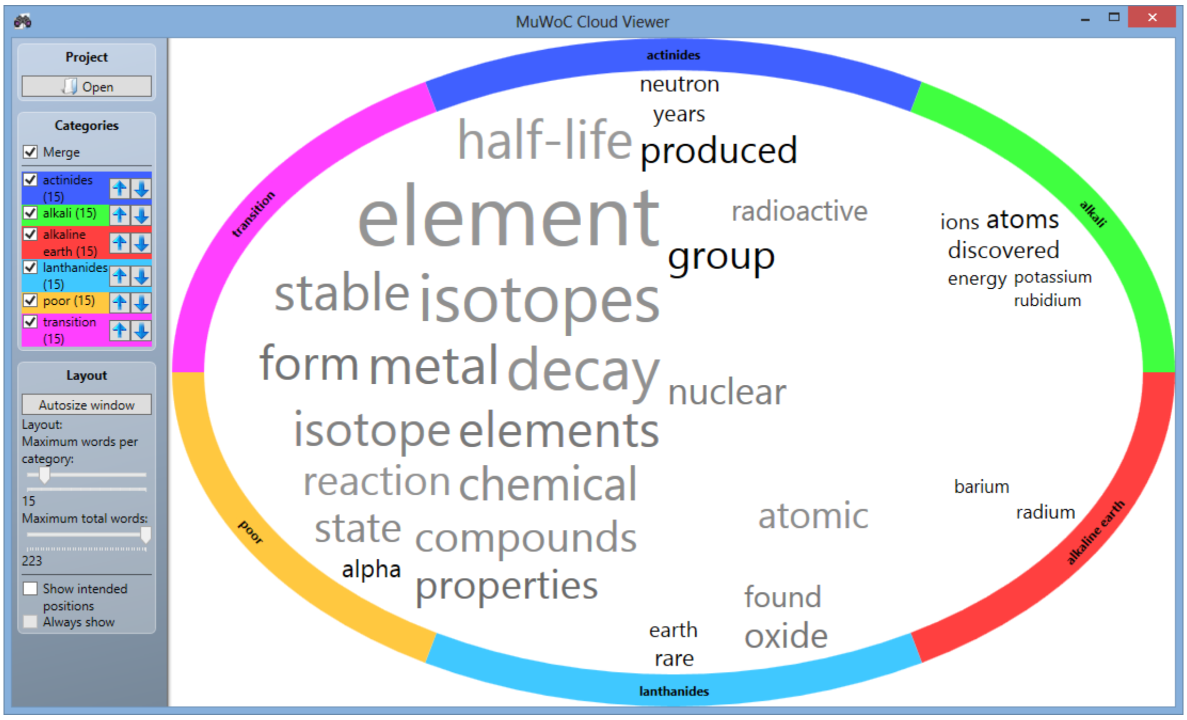
3.3.2.2.2 Category bar

As the words' positions are dependent on how much they belong to each category, there should be a way to show this comparative information to the user in a less ambiguous manner. MuWoC provides the possibility to underline each word with a horizontally stacked bar chart with predetermined width as shown in figure 3.9. As it displays the relevance in each category, it is called *category bar*.

The colors used in the bar represent the different categories. The categories in turn get their color assigned by the program, which stores them as a list in the user settings. This list is repeated, should there be more categories than colors. By default, MuWoC already contains a list with seven colors, which can be changed manually in the supplied settings file.

3.3.2.2.3 Word colors

Figure 3.9 shows that alongside the category bar, there is also a method of using the colors on the words themselves. To obtain a foreground brush for the text display, the weight distribution



MuWoC Cloud Viewer

Project

Open

Categories

☑ Merge

☑ actinides (15)

☑ alkali (15)

☑ alkaline earth (15)

☑ lanthanides (15)

☑ poor (15)

☑ transition (15)

Layout

Autosize window

Layout:

Maximum words per category: 15

Maximum total words: 223

☐ Show intended positions

☐ Always show

Figure 3.10: The merged view using bichromatic word coloration with black and gray.

across the categories has to be analyzed. If a word has equal relevance in each cloud equally, it does not belong specifically to only one. This results in an intended position near the middle. MuWoC maps this belonging and not-belonging to colors. When a word is only in one category, it inherits its color. As the word also belongs to other categories, the color gets less and less saturated, but does not change its hue. This way the color still shows to which category a word belongs most, but also signals that is present in others, too. The more a word belongs to multiple categories, the closer it is to the center and the less saturated it becomes.

MuWoC can also use this type of color mapping in a bichrome mode as shown in figure 3.10. For this, it does not operate with the saturation, but with a linear gradient. The normal display color and the one used for the bichrome mode are the end points. A word that belongs to only one category gets assigned the regular display color, one that belongs to multiple categories gets a color along the gradient that is nearer to the bichrome target color.

In order to calculate a metric of belonging to one or more categories, MuWoC uses the relative weights introduced in the RadCloud.

$$\begin{aligned} C &:= \text{the set of all categories} \\ w_{c_i} &:= \text{a word's weight in category } c_i \\ W &:= \{w_{c_i} : c_i \in C\} \\ \sum(W) &:= 1 \end{aligned} \tag{3.3}$$

Assuming a word belongs to $n \leq |C|$ categories, there are certain boundaries in between which $\max(W)$ has to lie. If it is only in one category then $\max(W) = 1$ is the upper and lower limit at the same time. If it belongs to more than one categories then $\max(W) = \frac{1}{n}$ is the lower limit and $\max(W) = 1$ can be approximated but never achieved. Therefore the measure of belonging is 1 for only one and calculated as follows for multiple categories:

$$\begin{aligned} min &:= \frac{1}{n} \\ max &:= \max(W) \\ belonging &:= \frac{max - min}{1 - min} \end{aligned} \tag{3.4}$$

3.3.2.2.4 Extra information and interactivity

The interactive features of the merged view are similar to the ones also available in the separate representation. The RadCloud algorithm is also used for the layout of the elements contained in a ListBox, so that it is fully integrated with the linking of the separate clouds. As the user selects a word in the merged view, the selection is propagated to the individual clouds for the categories. The user can take advantage of this feature to quickly switch between visualization methods and check where the words appear and relevant they are with respect to single categories or the whole dataset.

In the information available as tooltips on the words is more extensive than in the separate representation. It includes the merged metric value as well the metrics for each category.

3.3.2.3 Runtime settings

The cloud window does not use a typical top menu for its runtime user adjustable settings. Many consumer displays have adopted an aspect ratio of 16:9. Since many visualizations approximate a rectangular or circular shape, it is better to use space from the sides, than from the top or bottom for menus. Therefore MuWoC employs a side bar for typical application menu scenarios (see for instance figure 3.10).

It contains a button that serves to select and open projects at the top most position. The file selection uses the *Ookii.Dialogs* [Gro13] library's file dialog to provide a comparable user experience across multiple operating system versions.

Beneath that is an area that contains the controls for possible interactions with the categories. The user can hide individual word clouds by unchecking the corresponding boxes in this area. Additionally the categories and their respective clouds can be reordered by use of the up and down buttons that are displayed on each category check box. The visualizations can be merged by selecting the check box on top of the category list.

The last control area is dedicated to the layout. It contains a button to let the window revert to its default size. This size and other available controls change depending on the mode the window is in.

3.3.2.3.1 Separate mode
In the separate mode, the window is shrunk until the category clouds are shown in their current layout but without being zoomed or leaving unused space.

Another layout option is adjusted by a slider than limits the maximum amount of displayed words per visualization. Its value ranges from 1 to the number the user selected when the project was saved (see section 3.2.5). This information is not available when a simple CSV is loaded. MuWoC therefore checks the categories when it opens the project to determine the amount of words in the biggest categories and sets it as the upper limit for the slider.

A combo box is used to select the algorithm for the layout of the separate word clouds. The available options are implementations of methods described in section 2.1.2 as well as experimental variations of the Space Station algorithm:

Spiral (see section 2.1.2.1)

Vertical Stack (see section 2.1.2.4)

Space Station (see section 2.1.2.3)

Double Space Station
Uses two docks on each side. They are located at the same spot in the middle but the adjacent words are placed there using their corners instead of the centers.

Brick Wall
Uses two docks on the top and bottom like the Double Space Station. The left and right sides are the same as with the regular Space Station.

Corner Dock
Uses the same docks as the regular Space Station but also adds new ones. For each of the four sides, there are docks at the beginning and the end. This allows the words to not only be centered in the middle but to also be aligned at the right, left, top or bottom of the respective sides.

3.3.2.3.2 Merged mode
In the merged mode, the window's default size is dependent on the estimated minimal size of the RadCloud (see section 2.2.2.3.3). The possibility to limit the maximum number of visible words per category is still present and has the same functionality as in the separate view. However, since the RadCloud only merges the information that would be available in the standard clouds, the word's weight in a category is assumed to be 0.0 if it is not visible in it. This also means, that the extra information in the tooltips is updated and does not take the categories into account to which the word would normally belong if it would still be visible.

As the RadCloud visualization uses a considerably more complex algorithm, it does not perform as well as the standard clouds. To enable the program to operate with a smooth user experience even when there are many words, the maximum number of rendered elements can be restricted without affecting the word visibility in the separate view. MuWoC provides a slider for this

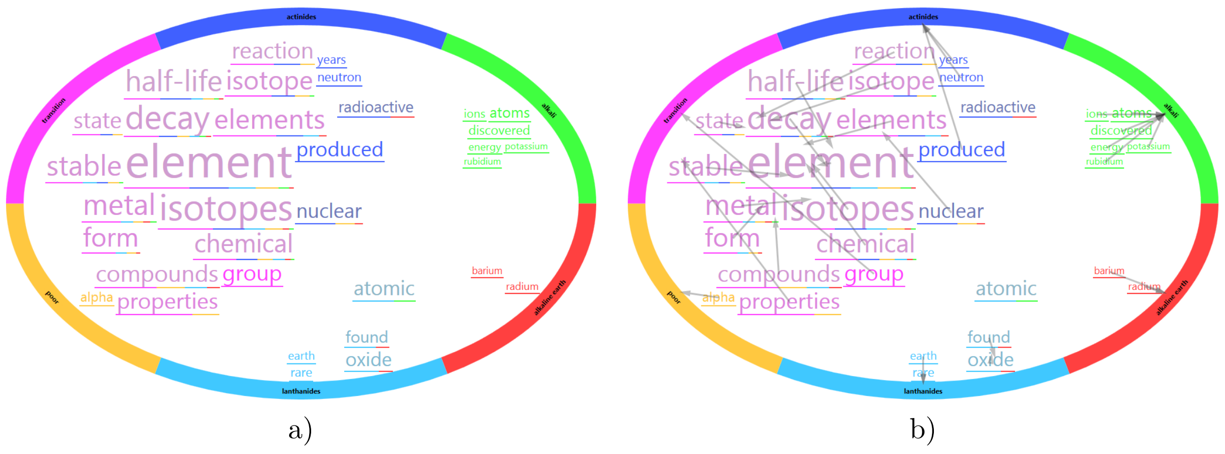


Figure 3.11: The RadCloud with (b) and without (a) intended position indicators.

option that is restricted by 1 on the lower end and by the total amount of words on the upper side.

The main difficulty with the RadCloud is the avoidance of overlaps and the applied solution to this problem shifts words from there intended positions. To provide the user with the unmodified, original position, the side bar offers a check box that enables the display of an overlayed arrow to the selected word's intended location. A second option adds the same indicator to all words but renders them semi-transparently to preserve readability (see figure 3.11). This feature also indicates how well the words were placed and therefore helps adjusting the RadCloud's settings.

As the individual categories can still be hidden, the UI also allows displaying the RadCloud with a single or even no category at all. However, this visualization was specifically designed for the merging of *multiple* word clouds. Therefore, it will not render any content if less than two categories are visible. It is possible to block the interface controls to avoid this situation but that also forces the user to adopt a specific behavior when the last two visible categories are to be interchanged with others. To solve the conflict between implausible settings and user restrictions, MuWoC displays a blue information message in the center of the RadCloud when less than two clouds are displayed in the merged view.

3.3.2.4 Entity explorer

The cloud window provides overviews on the categories, their similarities and differences but the level of detail is not very high. The word clouds do not show any entities or connections between the displayed items. Thus MuWoC offers a hierarchical way of data exploration. The entity explorer is the first level, as it allows selecting individual entities. The linkage by neighborhood serves as a step during the entity and word exploration. The second level only displays the neighborhoods from the source files, while the third level shows them entirely.

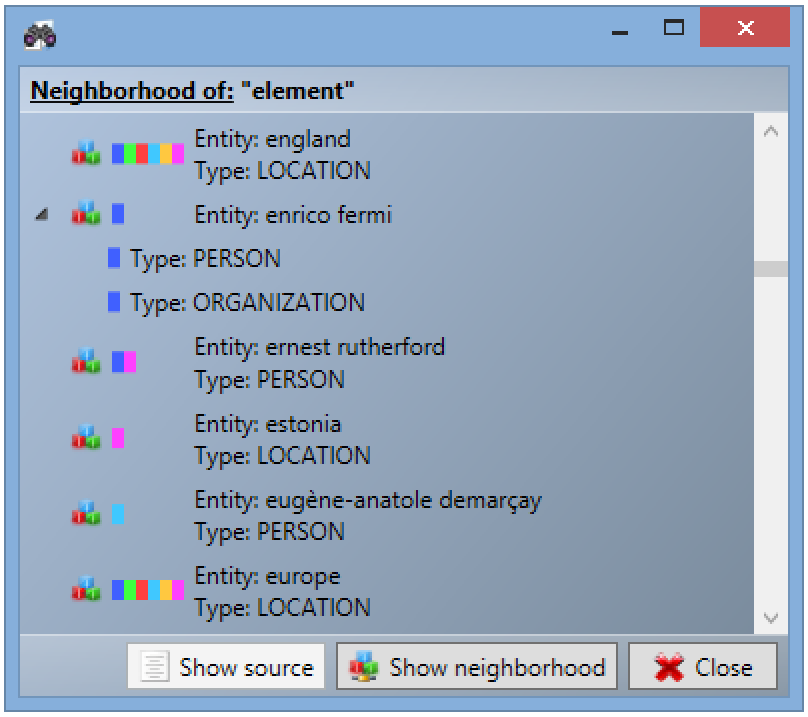


Figure 3.12: MuWoC's entity explorer.

For instance if a single forename is present in a word cloud, it is not clear which person was being referenced in the original source text. To get this information, the program can use the entities that were detected before extracting the single text tokens. When the user double clicks on a word in a cloud, MuWoC searches all entities for the inclusion of this exact string and displays them in the entity explorer. As shown in figure 3.12, the explorer is a relatively simple window. The found entities are displayed in a list that is ordered by length: the shorter the entity name, the closer it matches the contained word. If there are multiple types of entities with the same name, the entry can be expanded to reveal all of them.

Each list item also has an indicator that shows in which category it appears. It is comprised of rectangular icons that stacked horizontally and colored in accordance with the categories they represent. To not force the user to memorize the colors that were assigned to the individuals categories, the icons also provide tooltips with the names of what they represent.

3.3.2.4.1 Linkage by neighborhoods

From this first level of exploration, the user can double click on an item in the list. The action will open a new explorer that shows all entities that appeared near the click target's positions in the source documents. This time around the list is ordered alphabetically, as the name's length does not represent how close the entities are to each other. The user can define a neighborhood that is used to determine whether two appearances are near to each other. This definition can be set more abstractly as being a sentence or more concretely by specifying the maximum radius in terms of a word count.

Both options require the entities and words to be stored in the project with the exact index of occurrence. The character index of the original source file is not a fitting way of measuring the point of appearance because words do not have a constant length. It would therefore be necessary to add the length of the current word to its index and then search for the next one at this target location. But this method is not reliable because the amount of whitespace characters between the words can vary and because not all words are stored in the project files. Text sources must therefore be indexed by words and not characters.
To enable the neighborhood definition by sentence, it is necessary to include the indices of the ends of sentences in the project information. MuWoC can then check whether two entities appeared in the same range by using algorithms 3.1 or 3.2.

Algorithm 3.1 Checks whether two words appear in the same fixed width neighborhood.

```
function AreFixedNeighbors(array $first$, array $second$, $radius$)
    f ← 0
    s ← 0
    repeat
        diff ← $first$[f] − $second$[s]
        if diff = 0 then
            return true
        end if
        if diff < 0 then
            if −diff ≤ $radius$ then
                return true
            end if
            f++
        end if
        if diff > 0 then
            if diff ≤ $radius$ then
                return true
            end if
            s++
        end if
    until f ≥ $first$.Length ∨ s ≥ $second$.Length
    return false
end function
```

When the user double clicks on an element of an explorer on second level or higher, he opens the next window in the hierarchy of data exploration: a source viewer.

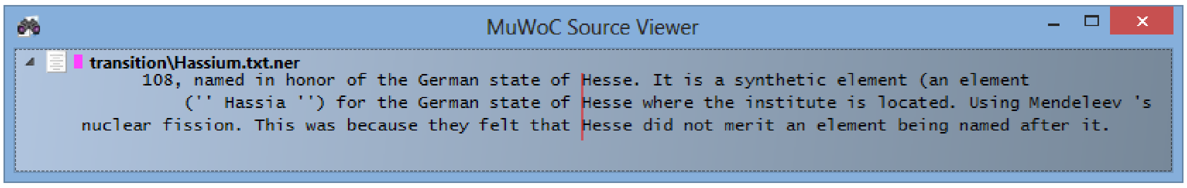
3.3.2.5 Source viewer

The source viewer displays the neighborhoods of all appearances of a word or entity (see figure 3.13). As it implements a concordance view [Wik13a], the main goal is to stack them vertically while aligning the target words. MuWoC places a vertical red line at the beginning

Algorithm 3.2 Checks whether two words appear in the same sentence.

```
function AreSentenceNeighbors(array $sentences$, array $first$, array $second$)
                                                        // Check the parameters
    if $first$.Length $= 0 \lor$ $second$.Length $= 0$ then
        return false
    end if
    if $sentences$.Length $< 1$ then
        return $first$.Length $> 0 \land$ $second$.Length $> 0$
    end if
                                        // Get the boundaries for the first sentence
    lower $\leftarrow -1$
    if $first[0] <$ lower then
        lower $\leftarrow first[0] - 1$
    end if
    if $second[0] <$ lower then
        lower $\leftarrow second[0] - 1$
    end if
    upper $\leftarrow sentences[0]$
                            // Check whether the first sentence contains both words
    f $\leftarrow 0$
    s $\leftarrow 0$
    firstContained $\leftarrow$ false
    while f $<$ $first$.Length do
        if $first[f] >$ lower then
            firstContained $\leftarrow first[f] \leq$ upper
            break
        end if
        f++
    end while
    secondContained $\leftarrow$ false
    while s $<$ $second$.Length do
        if $second[s] >$ lower then
            secondContained $\leftarrow second[s] \leq$ upper
            break
        end if
        s++
    end while
    if firstContained $\land$ secondContained then
        return true
    end if
```

```
                                                        // Repeat for every sentence
    for i = 1, *sentences*.Length − 1 do
        lower ← *sentences*[i − 1]
        upper ← *sentences*[i]
        firstContained ← false
        while f < *first*.Length do
            if *first*[f] > lower then
                firstContained ← *first*[f] ≤ upper
                break
            end if
            f++
        end while
        secondContained ← false
        while s < *second*.Length do
            if *second*[s] > lower then
                secondContained ← *second*[s] ≤ upper
                break
            end if
            s++
        end while
        if firstContained ∧ secondContained then
            return true
        end if
    end for
    return false
end function
```



```
transition\Hassium.txt.ner
        108, named in honor of the German state of Hesse. It is a synthetic element (an element
             ('' Hassia '') for the German state of Hesse where the institute is located. Using Mendeleev 's
  nuclear fission. This was because they felt that Hesse did not merit an element being named after it.
```

Figure 3.13: The source viewer with extracts from a single file.

of the target to highlight its position. To ensure good comparability between the lines of text, the UI font is replaced by a monospaced family.

Since the project might not contain all words that should be displayed, it has to access the original source files of the extraction process. MuWoC uses plug-ins to support a variety of file formats and therefore cannot read them directly on its own. However, the projects also contain the complete type name of the access and localization plug-ins that were used at the time of the extraction. This information can be passed to the .NET Framework to try and create new instances of the used classes. If the operation is successful, the plug-ins are used to read the files anew. This time they are not responsible for returning the words and entities, but to read the neighborhoods of the indices of the target word or entity.

As source formats like TXP or NER lack the information about whitespace in the unprocessed files, there is no definitive way for MuWoC to reconstruct an exact replica of the source text. It is however possible to use the cultural information from the localization plug-in to only add spaces where they usually occur. While consecutive words are normally separated, single character tokens - for instance punctuation - can be directly adjacent to the preceding word. The reconstruction methods from the file format plug-ins send single character tokens to the localization plug-ins to check whether they usually have spaces before and after them. The included English, German and Italian implementations use two hash sets each to quickly find whether the characters are listed as being used directly before or after words.

Since this task involves input operations on non-volatile media, it is potentially long-running. There is no implemented upper bound on the number of source files a word or entity can have and if the source location needs spin up time or is hosted remotely, the read operations can take more than a single second. Thus MuWoC performs these retrievals asynchronously to prevent UI lockups. The source viewer also provides a button at the bottom that can be used to cancel the neighborhood lookups.
If there is any problem with the process, the file that caused it is skipped and an error icon appears next to its name in the UI. The user can then hover over the icon to get a tooltip with information about the error that occurred.

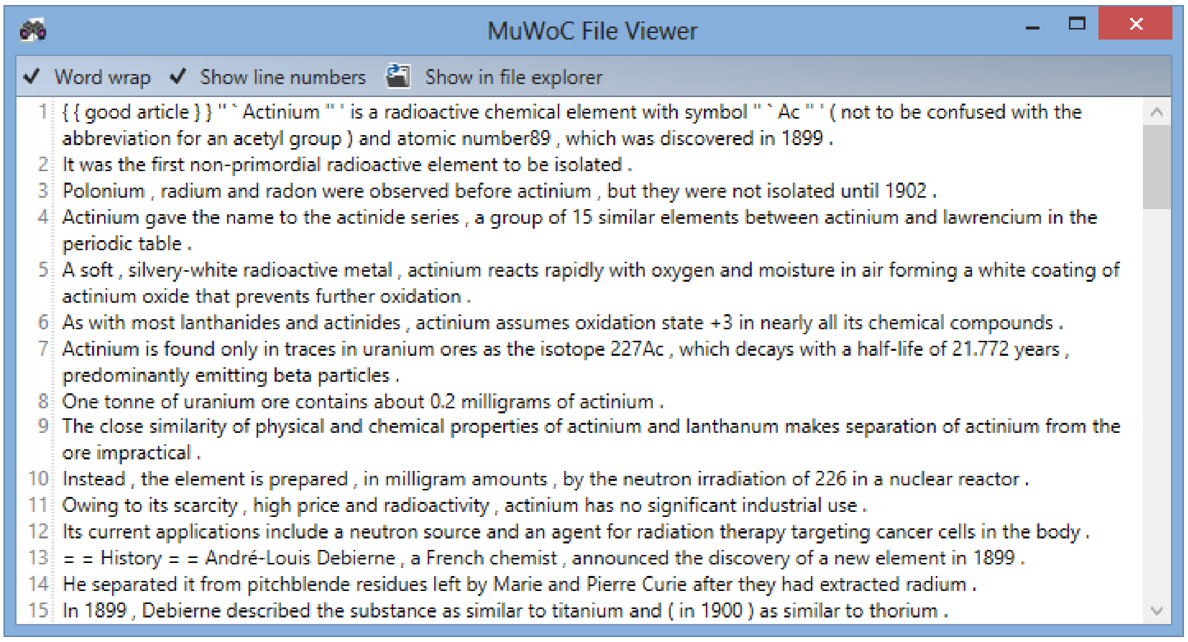
There can be thousands or more appearances of a word or entity. If the UI would display them all, it could become unresponsive or exceed the program's memory limits. To keep the interface clean and prevent automatic overflows, the loaded neighborhoods are only displayed when the user expands the information on the specific files individually.

3.3.2.6 File viewer

A double click on a line from the source viewer goes on to the third and final level of the data exploration hierarchy described in section 3.3.2.4. It displays the file viewer showcased in figure 3.14. This viewer makes use of the plug-ins, too. The difference is, that it doesn't only load neighborhoods, but entire files.

In the case of the TXT plug-in the most simple implementation is to use the IO system's default methods to return the complete content from text files.
The TXP and NER formats however do not contain information on line breaks and paragraphs. To avoid displaying the entire file's text in a single line, a break is inserted after each sentence.

Microsoft's default WPF text controls are suited for shorter texts but their performance quickly declines with the length on their content. Daniel Grunwald developed *AvalonEdit* [Gru13a] using WPF. The main goal was to replace the previous code editor for the integrated development environment *SharpDevelop*. To accomplish that it had to be extensible and easy to use while coping with large files. Grunwald's example of a large file is 7 Megabyte in size and contains 74100 lines of code [Gru13b].



```
{ { good article } } " ` Actinium " ' is a radioactive chemical element with symbol " ` Ac " ' ( not to be confused with the abbreviation for an acetyl group ) and atomic number89 , which was discovered in 1899 .
It was the first non-primordial radioactive element to be isolated .
Polonium , radium and radon were observed before actinium , but they were not isolated until 1902 .
Actinium gave the name to the actinide series , a group of 15 similar elements between actinium and lawrencium in the periodic table .
A soft , silvery-white radioactive metal , actinium reacts rapidly with oxygen and moisture in air forming a white coating of actinium oxide that prevents further oxidation .
As with most lanthanides and actinides , actinium assumes oxidation state +3 in nearly all its chemical compounds .
Actinium is found only in traces in uranium ores as the isotope 227Ac , which decays with a half-life of 21.772 years , predominantly emitting beta particles .
One tonne of uranium ore contains about 0.2 milligrams of actinium .
The close similarity of physical and chemical properties of actinium and lanthanum makes separation of actinium from the ore impractical .
Instead , the element is prepared , in milligram amounts , by the neutron irradiation of 226 in a nuclear reactor .
Owing to its scarcity , high price and radioactivity , actinium has no significant industrial use .
Its current applications include a neutron source and an agent for radiation therapy targeting cancer cells in the body .
= = History = = André-Louis Debierne , a French chemist , announced the discovery of a new element in 1899 .
He separated it from pitchblende residues left by Marie and Pierre Curie after they had extracted radium .
In 1899 , Debierne described the substance as similar to titanium and ( in 1900 ) as similar to thorium .
```

Figure 3.14: The source viewer with extracts from a single file.

MuWoC uses AvalonEdit in a read-only mode to display the text content from the source files. It was primarily chosen because of its performance advantages but also offers other useful features. Two of those are available from the window's menu. They are line numbers and the option to wrap long lines. As the user might also want to open the currently displayed file in another program, there is one more menu option that opens Microsoft Window's filesystem explorer and preselects the source.

The menu buttons in the file viewer use icons to ease their recognition. Their images were created by Microsoft and distributed in the *Modern Image Library*, which is part of the *Visual Studio 2012* installation.

3.3.2.7 Exploration menu

The exploration levels described in the sections 3.3.2.4 to 3.3.2.6 represent the intended use of MuWoC. To enable the user to navigate the available information more freely, the doubly clicked elements also feature a context menu that supports skipping steps (see figure 3.15). The menu also allows to search for words instead of entities in the neighborhoods. The entity explorer therefore has information messages that explain what action produced the displayed list and provides every entry with an icon that facilitates the distinction between works and entities.

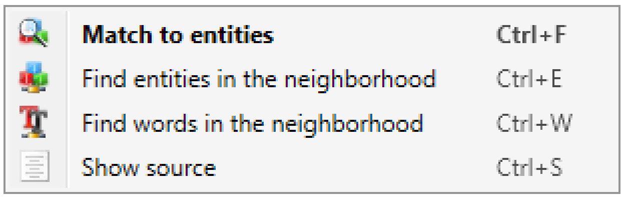

Match to entities Ctrl+F
Find entities in the neighborhood Ctrl+E
Find words in the neighborhood Ctrl+W
Show source Ctrl+S

Figure 3.15: The context menu on a word in the separate view.

3.4 Possible improvements

There are several possible improvements to MuWoC that have not been implemented yet

3.4.1 Stemming

Stemming could be used to improve the quality of the word extraction results, because it allows the user to see all forms of a word together [MRS08, ch. 15.3.2, p. 338]. However, this would also abstract away differences between documents. Since the SVM already provides a method of finding out which are the most relevant ones, it probably would only give moderately better results. The memory representation of the findings could be implemented analogously to the entities. They are stored with their data separately for each type but also aggregated in a container. The prefix tag cloud [BLW13] by Burch et al., could be used directly to visualize this information in the separate view. A variation thereof could also be suited for the merged mode.

3.4.2 Custom stop words

It would be helpful to provide a method for the user to add entries from the metric window's grid to the stop word list. This method would not require an application restart and re-extraction when there are unwanted words. The feature could be extended to include not only localization-specific stop word lists, but also custom user defined lists as input for the extraction settings.

3.4.3 Entity clouds

The word clouds could also be used with the entities. This would replace the TXP and NER plug-ins' mode to read only entities and store them as words. The cloud window could then provide a switch for the visualizations' data source at runtime.

3.4.4 Category colors

MuWoC ships with a list of predefined colors for the categories. Instead of repeating this list, it would be better to generate new random colors that are as different as possible from the existing ones. Martin Ankerl has published an implementation of such a generator on this blog [Ank09]. The algorithm could be adapted to produce only colors with good readability on MuWoC's background.

It would also be good if the user could assign a specific colors to individual categories. This information would have to be persisted in the project files, which would also mean that the cloud viewer would not only open but also save them.

To provide a good category name visibility on any selected color, MuWoC would have to change the text brushes dynamically. A comment on Ankerl's website provides the address of another site which contains an algorithm that can help selecting white or black as text colors depending on the background [Gil07]. While the site itself does not provide information about its author, it appeats to be written by "Jon von Gillern".

3.4.5 Drag and drop for categories

The drag and drop functionality for the categories could improve the user experience as it is related to the way people work with non-virtual objects. This feature could be implemented primarily on the side bar's list. An extension to the standard and the merged word clouds would be even better.

3.4.6 File viewer

Currently the file viewer only reads the source and shows it starting from the top. A better first position for the file content display would be the neighborhood that was used to open the window. This would enable the user to start reading larger surroundings immediately without having to scan the text for the extract himself. Another fundamental and useful feature would be the ability to search for strings and highlight their appearances. *AvalonEdit* already implements methods of highlighting certain strings and using backgrounds because of its original design goals.

3.4.7 Word cloud layouts

The layout algorithm for the separate view in the cloud window could be improved to maximize the zoom factor. The RadCloud could be overlayed with more than just the intended position of the words. A conceivable option would show the springs that were used in the layout and connect the elements to the circle.

3.4.8 Image export

The export of the rendered clouds, would greatly improve MuWoC's value. It would allow the user to see the clouds after the program has been closed, without having to perform the layout anew. It would also allow preserving the exact image at the desired size but at a later time. Using only MuWoC would mean, that the window size would have to be manually adjusted to be exactly as it was the first time the user saw the rendered word clouds.

The separate view would allow the export as bitmap images in the *Portable Networks Graphics* (PNG) format. Also possible would be the use of vector-based image formats like *Scalable Vector Graphics* (SVG), *XML Paper Spefication* (XPS) as well as the *Portable Document Format* (PDF). Since the merged representation can grow infinitely, it would be restricted to the vector formats as the bitmaps could take up to much system memory.

4 Use Cases

The main goal in MuWoC's development was to create a visual text analysis tool that uses word clouds and can provide useful information. To evaluate its ability to help find commonalities and differences between multiple texts, it has been tested on a variety of source data sets.

4.1 Metals

The chemical elements that are categorized as metals belong to different groups. Alkali, alkaline earth, transition, actinides, lanthanides and poor metals are examples of such subcategories.

There is much scientific information available on chemical elements in general. It is therefore relatively easy to access it for this evaluation. Every groups contains multiple metals, which means that there are also multiple documents in each analysis category. The goal will be to find out whether a user can get information from the observation of the generated word clouds.

4.1.1 Format

The texts are taken from *Wikipedia, The Free Encyclopedia* [Fou13] using the public *MediaWiki* application programming interface. The regular website on Lithium is accessible through this URL:

`http://en.wikipedia.org/wiki/Lithium`

To get a plain text instead of an HTML file, the raw code written by the content contributors is easier to process. It can be retrieved by settings the URL parameter "action" to "raw". A web request to the following URL then yields a text file:

`http://en.wikipedia.org/wiki/Lithium?action=raw`

Annotations that produce hyperlinks, citations and tables are contained in this raw format. However, they are not helpful in producing a readable plain text version. Using a regular expression, these annotations can be removed almost completely.

To accumulate the text data for the evaluation, a *PowerShell* script for Windows was developed. Contrary to the regular command shell, the PowerShell allows leveraging the .NET Framework and can therefore even be used to substitute small programs. A sample of the scripts used to aggregate this data set is shown in listing 4.1.

```
# setup
$scriptpath = $MyInvocation.MyCommand.Path
$directory = Split-Path $scriptpath
$client = new-object System.Net.WebClient
$cleanpattern = "((?<=.)\{\{.+?\}\})|(\[\[([^\]]+\|)?)|(\]\])|(\{\|.*?[^\}]\}(?
      =[^\}]))|(\<ref\s*[^\>]*?\/\>)|(\<ref\s*[^\>]*?\>.*?\</ref\>)|(\<[^>]+?
      \>)|(==\s*Notes\s*==.*$)|(==\s*See also\s*==.*$)|(==\s*References\s*==.*$)"
$cleanoptions = [System.Text.RegularExpressions.RegexOptions] "SingleLine,
      CultureInvariant, Compiled"
$cleanregex = new-object System.Text.RegularExpressions.Regex($cleanpattern,
      $cleanoptions)
$entitypattern = "&\w+;"
$entityoptions = [System.Text.RegularExpressions.RegexOptions]"CultureInvariant,
      Compiled"
$entityregex = new-object System.Text.RegularExpressions.Regex($entitypattern,
      $entityoptions)

# alkali
$article = $client.DownloadString("http://en.wikipedia.org/wiki/Caesium?
      action=raw");
$article = $entityregex.Replace($article, "")
$article = $cleanregex.Replace($article, "")
Set-Content ($directory + "\alkali\Caesium.txt") $article
```

Listing 4.1: Excerpt from a script to get plain text versions of Wikipedia articles on alkali metals.

4.1.2 Results

The source texts were first used for a named entity recognition and then for the word extraction. The support vector metric on the TFiDF yielded an accuracy of 89.5 percent.

The merged view (see figure 4.1) with 40 words from each category shows words like *alpha*, *beta*, *radioactive* and *years*. This could indicate that many metals are radioactive and mainly have half-lives in the spans of years. They probably emit mostly alpha and beta radiation when decaying.

A word nearer to the actinide category is *neutron*. It indicates, that they also produce neutrons. This theory is supported by the fact, that Uranium is one of the metals in this group. It is well known that it is applied in various contexts of nuclear physics because of its emission of slow neutrons during the fission.
Another word near the actinide group is *target*. Further reading on Wikipedia reveals that elements with more Protons than Uranium were often produced by bombarding smaller atoms with alpha radiation.

The category of the alkali metals contains the word *light*. This suits them because of their relatively low density and firmness.

On the side of alkaline earth, words like *mineral*, *hydroxide* and *large* appear. This can probably be attributed to same reason, that also gave this group its name. Alkaline earth metals can be found in large quantities in the earth's crust. However, they usually don't appear in their pure form. Oxides are much more common. This also explains the word *process*: the pure elements have to be extracted first.

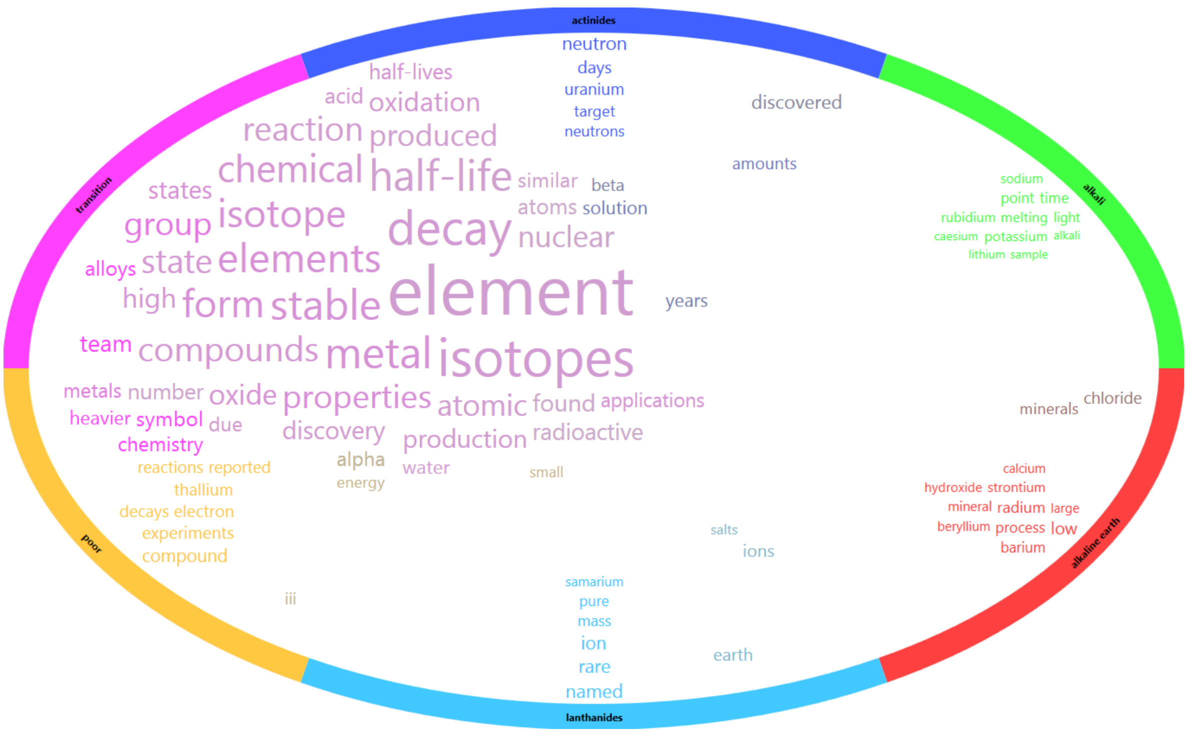


Figure 4.1: The merged view of the metal data with 40 words per category.

Rare and *earth* appear near the lanthanides. It is very probably attributed to their trivial name of "rare earths".

Team, *produced* provide hints that many transition metals do not occur naturally but were specifically created by scientists to analyze their properties. The transition metal Technetium was the first artificially produced element. This interpretation is supported by the finding from the actinides, which are a subcategory of the transition metals.

The group of poor metals did not contain special words that allowed fast access to its characteristics.

4.2 Reuters

Reuters' 21578 data set is commonly used as a benchmark for text classification tools. Since MuWoC can train a support vector machine for categorization, it should be possible to get relative good accuracy and therefore also helpful information.

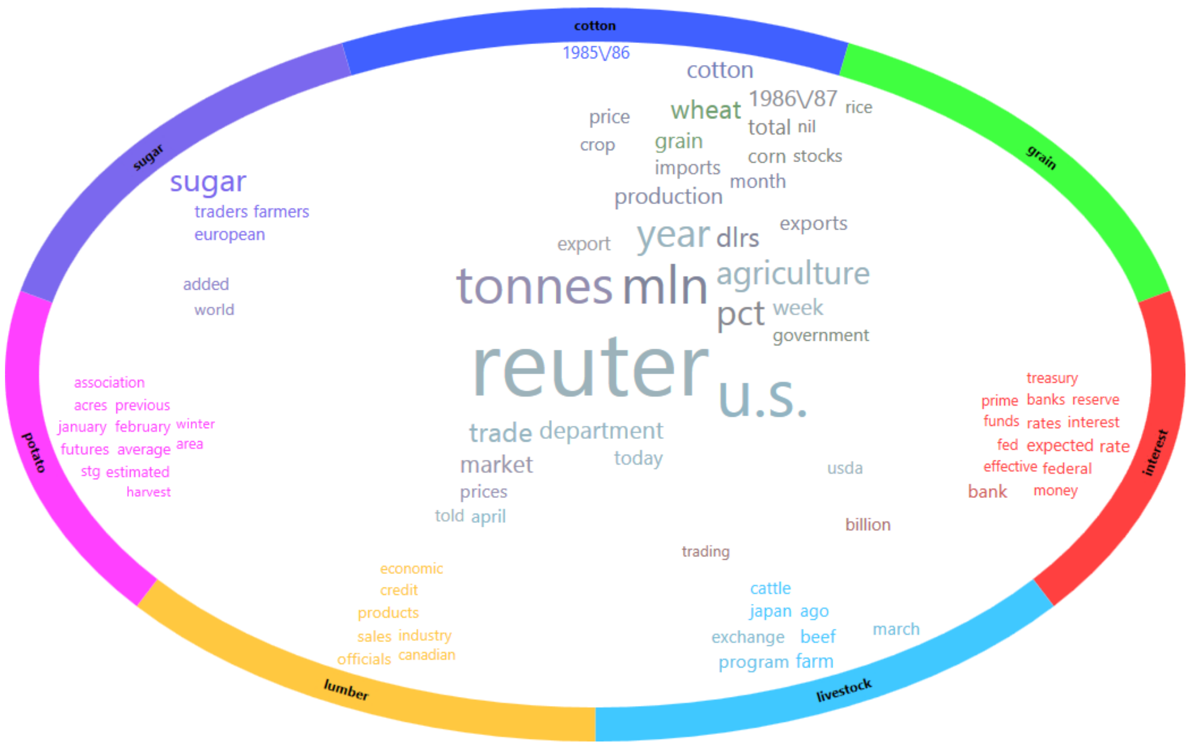


Figure 4.2: The merged view of the Reuters 21578 data with only 7 categories.

4.2.1 Format (also parser)

The entire dataset comes in SGML files. To get MuWoC's directory structure, it was necessary to implement a program to extract the single articles and sort them into the corresponding folders. Since the information about the correct category is already provided in the data set, it was only necessary to read it. Chris Lovett and Steve Bjorg's library *SgmlReader* [LB13] in version 1.8.11 was used to get the content of the downloaded files. An article can belong to multiple classifications at the same time. MuWoC respected this information by creating copies for each category.

The results for the classification by topic produced 13120 files in 119 directories. The separation by place even got 22341 files and 147 folders. Further categorizations were not processed to limit the amount of produced data.

4.2.2 Results

The named entity recognition on the places data set took only minutes in MuWoC. The extraction of words was done in some seconds but was followed by various minutes to load the user interface. The metric calculation failed in a first attempt with a 32 bit build, but caused no problems with the 64 bit version. The memory requirement during the SVM calculations remained constantly around 700 MiB for 8 hours. However, after this time only 19 of the 147 categories had been processed. The execution was therefore terminated by user input. The same procedure was repeated with only 7 topics containing 1370 files. The result had a good 92 percent accuracy and produced figure 4.2.

While *beef* was of importance in articles about livestock, *lumber* seems to be a big topic in Canada (*canadian*). The *potato harvest* per *acre* was *estimated* to be lower on *average* than in the *previous* year. The *european* market seems to have a great effect on the *sugar traders*. The *cotton* and *rice* industries appear to suffer from problems of low *prices* and need *government* subsidies to maintain *production*. Meanwhile the *bank rates* will not be lowered.

4.3 Jules Verne Selection

Jules Verne is an author of science fiction and adventure books. It would be interesting to find commonalities between the following selection of his works:

- Around The World In 80 Days [Ver73]
- 20,000 Leagues Under The Sea [Ver70]
- A Journey Into The Center Of The Earth [Ver77a]
- The Moon-Voyage [Ver77b]
- A Floating City and The Blockade Runners [Ver04]

4.3.1 Format

The files come from different sources: Archive.org[1], Project Gutenberg[2], feedbooks[3] and Wikisource[4]. The books have been downloaded in the EPUB format either directly from the respective websites or by using the *calibre* e-book management software. Calibre's integrated conversion then created the plain text files for the analysis.

4.3.2 Results

After tests with multiple different settings, the best achievable accuracy witht the SVM was 56 percent. This resulted in figure 4.3 but is not satisfactory at all. In the word clouds, there were no particular findings other than the word "made": it was used in almost every book. This could be an indication to the role of science fiction of Verne's texts. Since it means, that some content is not reality yet, it has to have been *made* by some character in the story.

[1] `http://archive.org/`
[2] `http://www.gutenberg.org/`
[3] `http://www.feedbooks.com/`
[4] `http://en.wikisource.org`

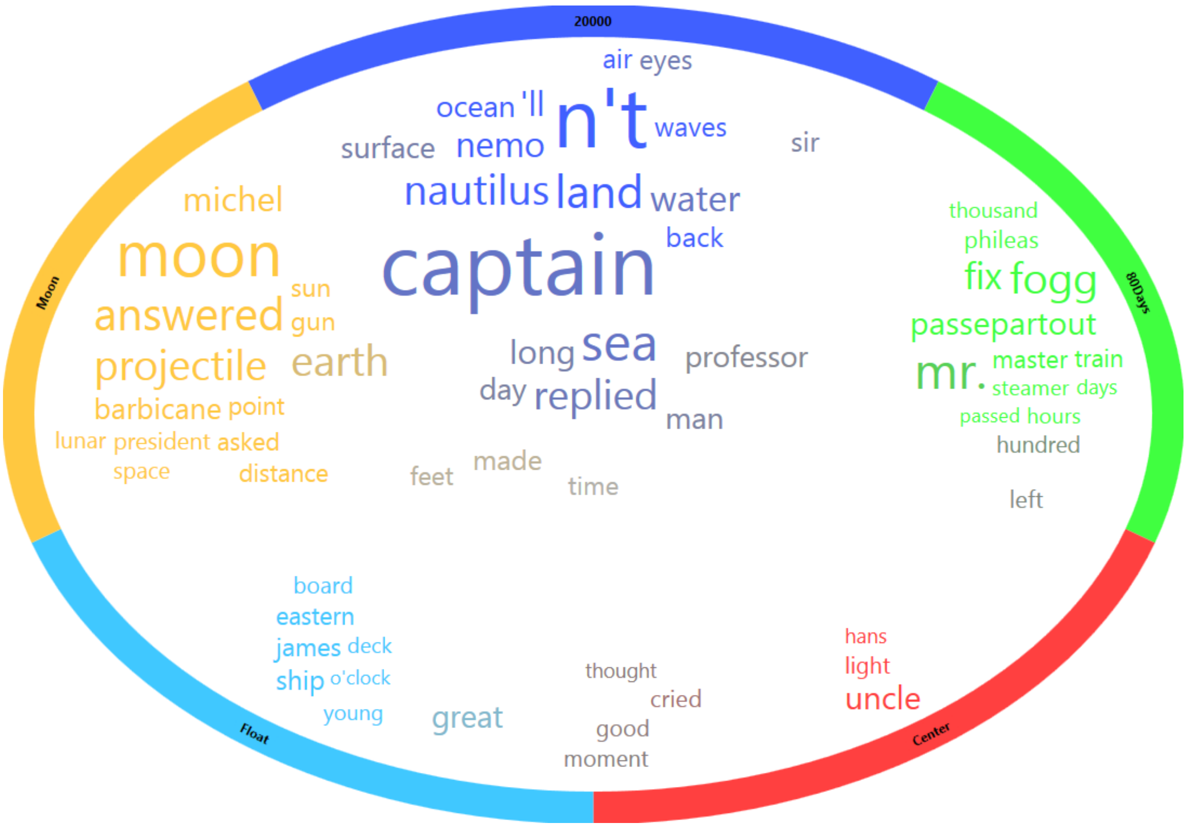


Figure 4.3: The merged view of the Jules Verne stories.

4.4 Goethe's Faust

The previous texts were all English. To have an example in other languages, Goethe's Faust was selected. It is a tragic play that inspired many other artists and became an important piece of German literature [Wik13b]. *Johann Wolfgang von Goethe* worked intermittently on several parts and versions between 1772 and 1831. The first version is called *Urfaust*. Later, Goethe returned to his work to create *Faust, a Fragment*. The last version was split into two part*Faust: The First Part of the Tragedy* and *Faust: The Second Part of the Tragedy*.

4.4.1 Format

The works were downloaded in plain text format from *Projekt Gutenberg*[2] and *bibliotheca Augustana*[5]:

- Urfaust[6]

[5]`http://www.hs-augsburg.de/~harsch/augustana.html`
[6]`http://www.hs-augsburg.de/~harsch/germanica/Chronologie/18Jh/Goethe/goe_uf00.html`

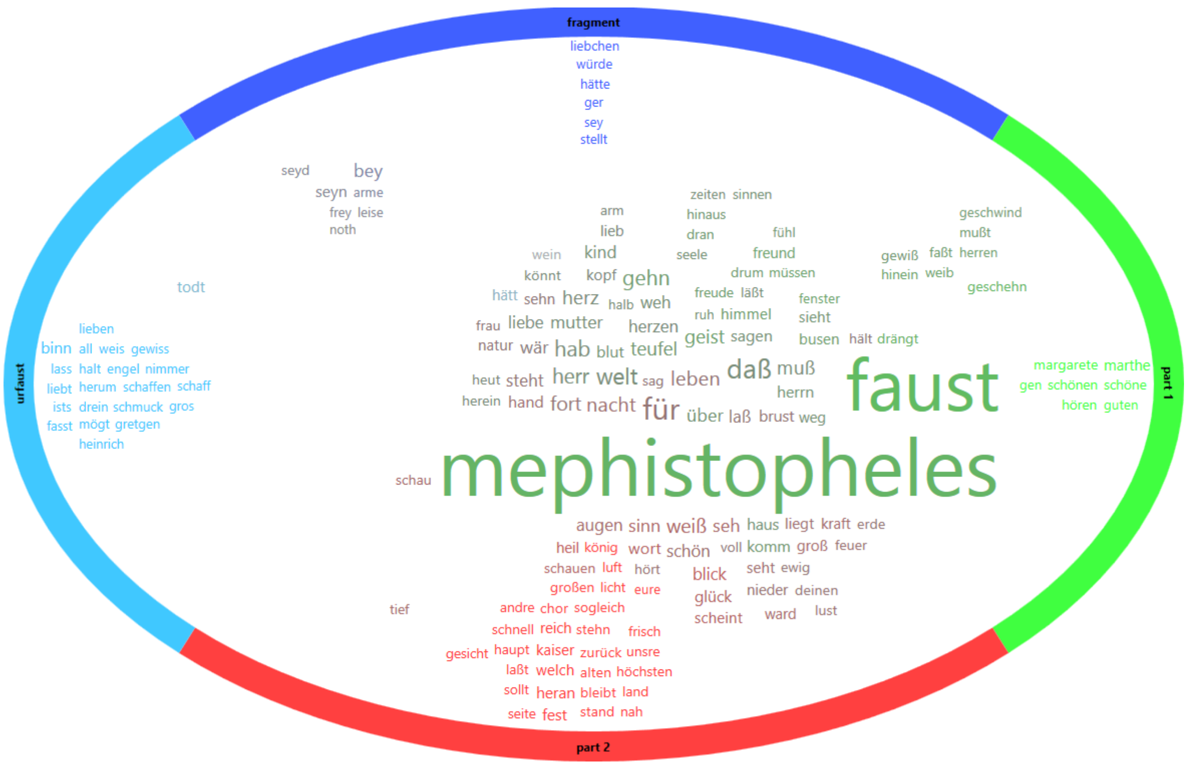


Figure 4.4: The merged view of the Goethe's Faust project.

- Faust, a Fragment. (Faust. Ein Fragment.)[7]
- Faust: The First Part of the Tragedy (Faust - Der Tragödie erster Teil)[8]
- Faust: The Second Part of the Tragedy (Faust - Der Tragödie zweiter Teil)[9]

As the bibliotheca Augustana does not provide direct download options, the website contents was copied and pasted into previously empty text files. Bibliotheca Augustana did not provide the entire works on a single site, but on one for each setting of the play. The texts were therefore stored in multiple files according to the divisions of the source.

4.4.2 Results

The use of the SVM only provided accuracies between 50 and 60 percent. To have an example of the use of other metrics and avoid the use of an unsatisfactory result, the TFiDF was used. The output of the 100 most relevant words per category yielded 168 distinct terms (see figure 4.4).

[7]`http://www.hs-augsburg.de/~harsch/germanica/Chronologie/18Jh/Goethe/goe_ff00.html`
[8]`http://www.gutenberg.org/ebooks/2229`
[9]`http://www.gutenberg.org/ebooks/2230`

The most important words are *Faust* and *Mephistopheles.* They are the names of the two main characters

The cloud shows well how *Urfaust* and *Faust, a Fragment* versions use older German orthography where many words are written with the letter *Y.* The newer first and second parts already use the letter *I* as in nowadays language.

Also very well visible is the omission of the letter *E* and the end of words. This is still customary in today's spoken language and was probably used this written form because it is a more accurate representation of what the characters say in the play. The omission of the terminal *E* also led to the inclusion of some stop words in the final output. This is due to the list only containing the entire form, not the shortened one.

4.5 CSV - Actors

A list of actors and their appearances in Wes Anderson's movies was created as a sample of MuWoC's ability to load arbitrary CSV sources.

4.5.1 Format

The file's columns correspond to the movies *Bottle Rocket*, *Rushmore*, *The Royal Tenenbaums*, *The Life Aquatic with Steve Zissou*, *The Darjeeling Limited*, *Fantastic Mr. Fox* and *Moonrise Kingdom.* Each actor is represented by a row: Noah Baumbach, Adrien Brody, Seymour Cassel, Brian Cox, Willem Dafoe, Michael Gambon, Jeff GoldBlum, Anjelica Huston, Harvey Keitel, Bill Murray, Edward Norton, Kumar Pallana, Jason Schwartzman, Andrew Wilson, Luke Wilson, Owen Wilson and Wallace Wolodarsky. The weights for each movie are only binary: 0 or 1. They correspond to whether or not the actors participated in the films.

4.5.2 Results

As figure 4.5 shows, three actors appeared in only one film. The others were part of multiple projects. Bill Murray and Owen Willson even acted in six of the seven selected movies. When words converge on the same spot, it means that those actors worked together on the exactly the same films. An example are Gambon, Baumbach and Dafoe on *Fantastic Mr. Fox* and *The Life Aquatic with Steve Zissou.*

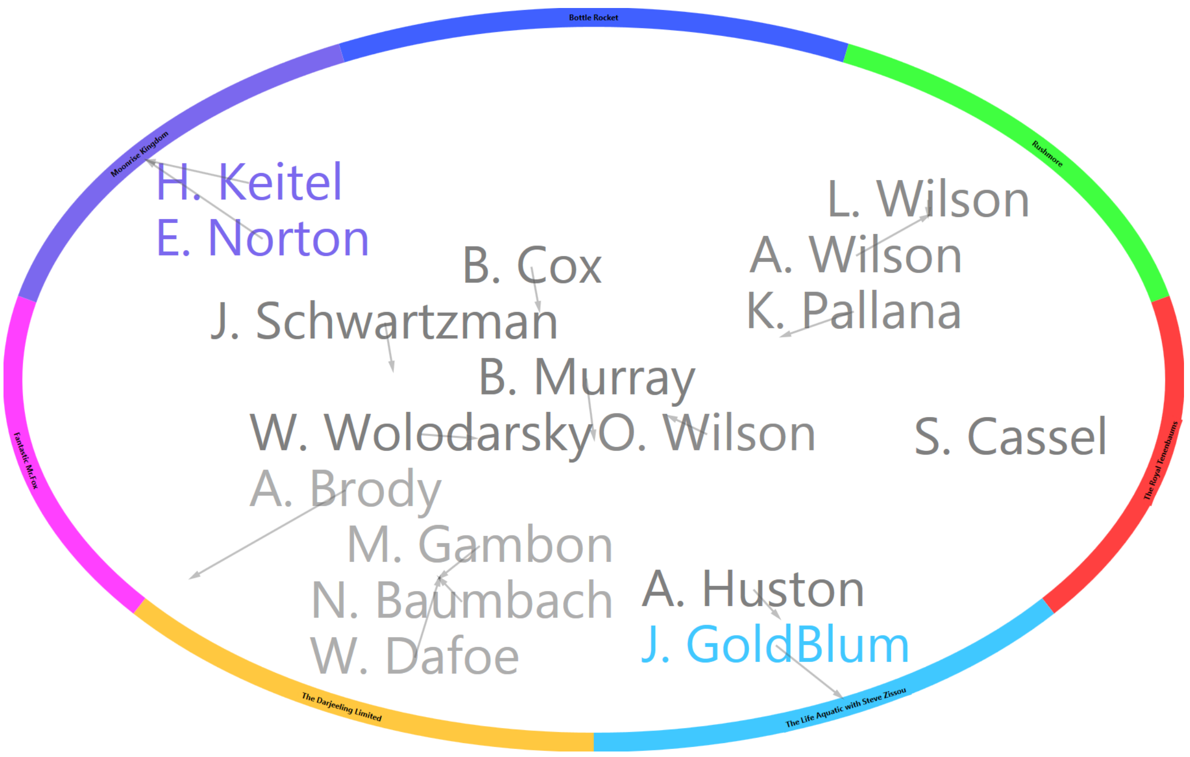


Figure 4.5: The merged view of the Jules Verne stories.

5 Conclusions

MuWoC is able to analyze texts and produce word clouds.

The RadCloud did help in finding similarities and differences between multiple categories. It was much more helpful to the user than separate traditional word clouds.

The combination of a support vector as relevance metric and the RadCloud as visualization can give a good overview of and insight into the source data. However, the quality is very dependent on the accuracy with which the support vector is able to categorize the documents.

Future versions should implement the possible improvements from the visualization and user interface sections, to make MuWoC a more useful tool in visual analytics. Further use case scenarios could compare how the TFiDF and the support vector perform in comparison. This could be especially interesting when the SVM does not produce satisfactory results.

Since MuWoC was developed to help humans get insight into textual data, the project's source code and binary output will be made publicly available on SourceForge[1] under the URL `http://muwoc.sourceforge.net/`. This will further improve the usefulness as the community can use and adapt it.

[1] `http://sourceforge.net/`

Bibliography

[AKK96] M. Ankerst, D. A. Keim, H.-P. Kriegel. Circle Segments: A Technique for Visually Exploring Large Multidimensional Data Sets. In *Visualization '96, Hot Topic Session, San Francisco, CA*. 1996. (Cited on page 22)

[Ank09] M. Ankerl. How to Generate Random Colors Programmatically. Website, 2009. URL `http://martin.ankerl.com/2009/12/09/how-to-create-random-colors-programmatically/`. (Cited on page 67)

[Bal11] A. Balucha. stop-words - Stop words. Website, 2011. URL `http://code.google.com/p/stop-words/`. (Cited on page 50)

[BBFZ09] M. Baroni, S. Bernardini, A. Ferraresi, E. Zanchetta. The WaCky wide web: a collection of very large linguistically processed web-crawled corpora. *Language resources and evaluation*, 43(3):209–226, 2009. (Cited on page 38)

[BDS05] E. Bertini, L. Dell'Aquila, G. Santucci. SpringView: Cooperation of radviz and parallel coordinates for view optimization and clutter reduction. In *Coordinated and Multiple Views in Exploratory Visualization, 2005.(CMV 2005). Proceedings. Third International Conference on*, pp. 22–29. IEEE, 2005. (Cited on page 23)

[BLW13] M. Burch, S. Lohmann, D. Weiskopf. Prefix Tag Clouds. In *Proceedings of the 17th International Conference on Information Visualisation (IV 2013)*, pp. 45–50. IEEE, Los Alamitos, CA, USA, 2013. (Cited on pages 13 and 66)

[CC07] C. Collins, S. Carpendale. VisLink: Revealing relationships amongst visualizations. *Visualization and Computer Graphics, IEEE Transactions on*, 13(6):1192–1199, 2007. (Cited on page 21)

[CL09] C.-C. Chang, C.-J. Lin. LIBSVM – A Library for Support Vector Machines. Computer software, 2009. URL `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`. (Cited on page 41)

[CL11] C.-C. Chang, C.-J. Lin. LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011. (Cited on page 41)

[CM86] W. S. Cleveland, R. McGill. An Experiment in Graphical Perception. *International Journal of Man-Machine Studies*, 25(5):491–501, 1986. (Cited on page 13)

[Coe13] E. Coelho. Everaldo Coelho. Website, 2013. URL `http://www.everaldo.com/`. (Cited on page 45)

[CS13] Q. Castella, C. Sutton. Word Storms: Multiples of Word Clouds for Visual Comparison of Documents. *arXiv preprint arXiv:1301.0503*, 2013. (Cited on page 21)

[CVW09] C. Collins, F. B. Viegas, M. Wattenberg. Parallel tag clouds to explore and analyze faceted text corpora. In *Visual Analytics Science and Technology, 2009. VAST 2009. IEEE Symposium on*, pp. 91–98. IEEE, 2009. (Cited on page 21)

[dlb07] dlb. Crystal Project is here : more than 900 icons ! Website, 2007. URL `http://www.crystalxp.net/news/en186-icons-crystal-project-download-everaldo.html`. (Cited on page 45)

[Fei10] J. Feinberg. Wordle. In N. I. Julie Steele, editor, *Beautiful Visualization*. O'Reilly Media, Inc., 2010. (Cited on pages 11, 12, 13 and 40)

[Fei13] J. Feinberg. Wordle. Computer software, 2013. URL `http://www.wordle.net/`. (Cited on pages 11 and 36)

[Fel12] I. Fellows. Words in Politics: Some extensions of the word cloud « Fells Stats. Website, 2012. URL `http://blog.fellstat.com/?p=101`. (Cited on page 21)

[Fou13] W. Foundation. Wikipedia, the free encyclopedia. Website, 2013. URL `http://en.wikipedia.org/`. (Cited on page 69)

[FP10] M. Faruqui, S. Padó. Training and Evaluating a German Named Entity Recognizer with Semantic Generalization. In *Proceedings of KONVENS 2010*. Saarbrücken, Germany, 2010. (Cited on page 38)

[Fri13] J. Frijters. IKVM.NET Home Page. Computer software, 2013. URL `http://www.ikvm.net/`. (Cited on page 38)

[Gil07] J. von Gillern. Black vs White Text. Website, 2007. URL `http://blog.nitriq.com/BlackVsWhiteText.aspx`. (Cited on page 67)

[Got09] T. Gottron. Document word clouds: Visualising web documents as tag clouds to aid users in relevance decisions. In *Research and Advanced Technology for Digital Libraries*, pp. 94–105. Springer, 2009. (Cited on page 9)

[Gro13] S. Groot. Ookii.Dialogs. Computer software, 2013. URL `http://www.ookii.org/Software/Dialogs/`. (Cited on pages 47 and 57)

[Gru13a] D. Grunwald. AvalonEdit 4.3.1.9430. Computer software, 2013. URL `http://www.nuget.org/packages/AvalonEdit`. (Cited on page 64)

[Gru13b] D. Grunwald. Using AvalonEdit (WPF Text Editor). Website, 2013. URL `http://www.codeproject.com/Articles/42490/Using-AvalonEdit-WPF-Text-Editor`. (Cited on page 64)

[Har81] Z. S. Harris. *Distributional structure*. Springer, 1981. (Cited on page 41)

[HFH$^+$09] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. H. Witten. The WEKA data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009. (Cited on page 35)

[HGM$^+$97] P. Hoffman, G. Grinstein, K. Marx, I. Grosse, E. Stanley. DNA visual and analytic data mining. In *Visualization'97., Proceedings*, pp. 437–441. IEEE, 1997. (Cited on pages 22 and 23)

[HGP99] P. Hoffman, G. Grinstein, D. Pinkney. Dimensional anchors: a graphic primitive for multidimensional multivariate information visualizations. In *Proceedings of the 1999 workshop on new paradigms in information visualization and manipulation in conjunction with the eighth ACM internation conference on Information and knowledge management*, pp. 9–16. ACM, 1999. (Cited on page 22)

[HIMM02] T. Hirao, H. Isozaki, E. Maeda, Y. Matsumoto. Extracting important sentences with support vector machines. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, pp. 1–7. Association for Computational Linguistics, 2002. (Cited on page 42)

[HM90] R. B. Haber, D. A. McNabb. Visualization idioms: A conceptual model for scientific visualization systems. *Visualization in scientific computing*, 74:93, 1990. (Cited on page 9)

[Hue98] E. B. Huey. Preliminary experiments in the physiology and psychology of reading. *The American Journal of Psychology*, 9(4):575–586, 1898. (Cited on page 12)

[Joh75] N. F. Johnson. On the function of letters in word identification: Some data and a preliminary model. *Journal of Verbal Learning and Verbal Behavior*, 14(1):17–29, 1975. (Cited on page 12)

[Joh09] M. A. Johnson. SVM.NET. Computer software, 2009. URL `http://www.matthewajohnson.org/software/svm.html`. (Cited on pages 41 and 51)

[LB13] C. Lovett, S. Bjorg. SgmlReader 1.8.11. Website, 2013. URL `http://www.nuget.org/packages/SgmlReader/`. (Cited on page 72)

[LL07] J. Ludewig, H. Lichter. *Software Engineering: Grundlagen, Menschen, Prozesse, Techniken.* dpunkt. verlag Heidelberg, 2007. (Cited on page 11)

[LT12] C. for the Evaluation of Language, C. Technologies. EVALITA | Evaluation of NLP and Speech Tools for Italian, 2012. URL `http://www.evalita.it/`. (Cited on page 37)

[Mas12] W. Mason. Comparative Word Cloud. Website, 2012. URL `http://winteram.com/blog/?p=208`. (Cited on pages 21, 22 and 54)

[MRS08] C. D. Manning, P. Raghavan, H. Schütze. *Introduction to information retrieval*, volume 1. Cambridge University Press Cambridge, 2008. (Cited on page 66)

[MWK+06] I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz, T. Euler. YALE: Rapid Prototyping for Complex Data Mining Tasks. In L. Ungar, M. Craven, D. Gunopulos, T. Eliassi-Rad, editors, *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 935–940. ACM, New York, NY, USA, 2006. doi:http://doi.acm.org/10.1145/1150402.1150531. URL `http://rapid-i.com/component/option,com_docman/task,doc_download/gid,25/Itemid,62/`. (Cited on page 35)

[OKS+93] K. A. Olsen, R. R. Korfhage, K. M. Sochats, M. B. Spring, J. G. Williams. Visualization of a document collection: The VIBE system. *Information Processing & Management*, 29(1):69–81, 1993. (Cited on pages 22 and 23)

[Out13] Outercurve Foundation. NuGet Gallery. Website, 2013. URL `http://www.nuget.org/`. (Cited on page 38)

[PN13] P. Palotas, A. Normuradov. AlphaFS. Website, 2013. URL `http://alphafs.codeplex.com/`. (Cited on page 36)

[Ray75] K. Rayner. The perceptual span and peripheral cues in reading. *Cognitive Psychology*, 7(1):65–81, 1975. doi:10.1016/0010-0285(75)90005-5. (Cited on pages 12 and 13)

[RB79] K. Rayner, J. H. Bertera. Reading without a fovea. *Science*, 206(4417):468–469, 1979. (Cited on page 13)

[RFP98] K. Rayner, M. H. Fischer, A. Pollatsek. Unspaced text interferes with both word identification and eye movement control. *Vision Research*, 38(8):1129–1144, 1998. (Cited on page 12)

[RHY05] B. Roberts, M. G. Harris, T. A. Yates. The roles of inducer size and distance in the Ebbinghaus illusion (Titchener circles). *PERCEPTION-LONDON-*, 34(7):847, 2005. (Cited on page 13)

[Sav13a] J. Savoy. englishST. Website, 2013. URL `http://members.unine.ch/jacques.savoy/clef/englishST.txt`. (Cited on page 50)

[Sav13b] J. Savoy. germanST. Website, 2013. URL `http://members.unine.ch/jacques.savoy/clef/germanST.txt`. (Cited on page 50)

[Sav13c] J. Savoy. IR Multilingual Resources at UniNE. Website, 2013. URL `http://members.unine.ch/jacques.savoy/clef/`. (Cited on page 50)

[Sav13d] J. Savoy. italianST. Website, 2013. URL `http://members.unine.ch/jacques.savoy/clef/italianST.txt`. (Cited on page 50)

[Seb02] F. Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002. (Cited on page 41)

[Slo73] N. J. A. Sloane. A001710/M2933. In *A Handbook of Integer Sequences*. Academic Press, 1973. (Cited on page 33)

[The13] The Stanford Natural Language Processing Group. Stanford Named Entity Recognizer (NER). Computer software, 2013. URL `http://nlp.stanford.edu/software/CRF-NER.shtml`. (Cited on pages 38 and 46)

[Tih13] S. Tihon. Stanford.NLP.NER 3.2.0. Computer software, 2013. URL `http://www.nuget.org/packages/Stanford.NLP.NER/`. (Cited on page 38)

[Uni13] Universität Stuttgart Institut für Maschinelle Sprachverarbeitung (University of Stuttgart Institute for Natural Language Processing). Huge German Corpus (HGC). Website, 2013. URL `http://www.ims.uni-stuttgart.de/forschung/ressourcen/korpora/hgc.en.html`. (Cited on page 38)

[VC74] V. N. Vapnik, A. J. Chervonenkis. Theory of pattern recognition. 1974. (Cited on page 35)

[Ver70] J. Verne. *20,000 Leagues Under the Sea*. Pierre-Jules Hetzel, 1870. URL `http://www.feedbooks.com/book/182`. Translation by F.P. Walter, 2001. (Cited on page 73)

[Ver73] J. Verne. *Around the World in Eighty Days*. Pierre-Jules Hetzel, 1873. URL `http://en.wikisource.org/w/index.php?title=Around_the_World_in_Eighty_Days&oldid=4561534`. Translation by Geo M. Towle. (Cited on page 73)

[Ver77a] J. Verne. *A Journey Into The Center Of The Earth*. Ward, Lock, &Co., Ltd., London, 1877. URL `http://www.feedbooks.com/book/3796`. (Cited on page 73)

[Ver77b] J. Verne. *The Moon-Voyage*. Ward, Lock, &Co., Ltd., London, 1877. URL `http://www.gutenberg.org/ebooks/12901`. (Cited on page 73)

[Ver04] J. Verne. *A floating city and the blockade runners*. C. Scribner's sons, 1904. URL `http://archive.org/details/floatingcitybloc00verniala`. (Cited on page 73)

[Wik13a] Wikipedia. Concordance (publishing) — Wikipedia, The Free Encyclopedia. Website, 2013. URL `http://en.wikipedia.org/w/index.php?title=Concordance_(publishing)&oldid=562109454`. (Cited on page 61)

[Wik13b] Wikipedia. Goethe's Faust — Wikipedia, The Free Encyclopedia. Website, 2013. URL `http://en.wikipedia.org/wiki/Goethe%27s_Faust?oldid=577711270`. (Cited on page 74)

All links were last followed on October 30, 2013.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature