Submitted to the Annals of the Department of Statistics and Operations Research

SCRAPING THE SYNTHETIC RISK AND REWARD INDICATOR FROM FUNDS' KEY INFORMATION DOCUMENTS

By Anonymous

University of Vienna

The main objective of this paper is to extract the SRRI (Synthetic Risk and Reward Indicator) from funds' KIDs (Key Information Documents). Two approaches are presented, the naive approach which does not rely on any statistical methods and the clustering approach, which utilizes hierarchical agglomerative clustering. The naive approach manages to correctly predict 80% of the test cases while the clustering approach performs worse at an accuracy of 70%. Both methods do not generate any incorrectly predicted cases, i.e., the result for the sample at hand is either a correct readout or an error. This is a crucial performance characteristic, as misclassified cases that do not result in an error are only observable if the files are opened and checked manually.

1. Introduction. According to BGBl. II Nr. 265/2011, effective 2011, every undertaking for collective investment in transferable securities (UCITS), has to publish a key information document (FMA, 2011). Said document is supposed to inform potential and current investors about various characteristics of the investment fund at hand. Besides a short description of the investments, as well as performance measured against a benchmark, this document also contains a synthetic risk- and reward indicator (SRRI). As the name suggests, the purpose of this indicator is to measure the risk and return associated with an investment in the respective fund. How the SRRI is derived, depends on data availability and will be described in greater detail in a separate section. Independent of the exact procedure of derivation, the underlying interpretation of risk is measured via the volatility of returns. This paper is structured as follows. First, a short and precise introduction into the calculation of the SRRI is provided. Then the data is briefly presented and the approach to measuring extraction performance is discussed. The following sections' focus shifts to extracting the SRRI, which is displayed as a graph, usually on the first page of every KID. In order to keep the structure as simple as possible, the proccess is split into two parent functions, which call a variety of helper functions. All functions are implemented in R, besides the base library, the use of further libraries is pointed out in the description of the individual functions (R Core Team, 2021).

MSC 2010 subject classifications: Primary 91C20, 62H30
Keywords and phrases: Data Mining, Hierarchical Clustering, LaTeX 2ε

2. SRRI. The SRRI aims to measure risk via the volatility of weekly returns from the last 5 years. Should weekly retuns be unobtainable, the calculation may be executed using montly returns. Accordingly, by ordinance the SRRI may be obtained as follows

$$\sigma_f = \sqrt{\frac{m}{T-1}\sum_{t=1}^{T}(r_{f,t} - \overline{r_f})^2},$$

where $r_{f,t}$ is the fund's return and $\overline{r_f}$ represents the mean of returns over T periods. Then scaling via m, the return frequency within a year, yields the standard deviation of yearly returns. For illustrative purposes, the calculation using weekly returns would result in $m = 52$, as there are 52 weeks in a year and thus $T = 5 * 52 = 260$ represents the number of weeks in 5 years. To obtain the SRRI which is measured on a scale from 1 to 7, the regulating authority provides a table (FMA, 2011).

TABLE 1
SRRI Classification

SRRI	σ_f
1	$0\% \leq \sigma_f < 0.5\%$
2	$0.5\% \leq \sigma_f < 2\%$
3	$2\% \leq \sigma_f < 5\%$
4	$5\% \leq \sigma_f < 10\%$
5	$10\% \leq \sigma_f < 15\%$
6	$15\% \leq \sigma_f < 25\%$
7	$25\% \geq \sigma_f$

3. Measuring Scraping Performance. This paper aims to automize the read-out of the SRRI given a KID, correspondingly, the performance evaluation is a bit more nuanced than in a usual prediction setting. We have to distinguish between a failed classification that results in an error, and a misclassification, i.e., the predicted SSRI is different to the actual SRRI. The latter carries the consequence, that the SRRI resulting from the extraction is different to the actual SRRI. In order for an automized extraction process to be a viable alternative to a human opening and reading the file, the amount of misclassified cases for the machine read-out may not exceed the expected error rate generated by a human read-out. To account for this, two different accuracy measures are introduced. Gross Accuracy measures the amount of misclassifications, before accounting for errors. Meaning that gross accuracy measures how many missclassified cases are created, that would be undetectable in production use. In order to also measure performance accounting for errors, Net Accuracy measures the extraction performance after accounting for errors. Formally, this means we will use a discrete metric for Gross Accuracy and Net Accuracy. Let m the number of files for which the readout did not result in an error and let n be the total number of files.

$$\text{Gross Accuracy} = \frac{1}{m}\sum_{i=1}^{m}\mathbb{1}_{\{pred_i\ =\ act_i\}},$$

$$\text{Net Accuracy} = \frac{1}{n}\sum_{i=1}^{n}\mathbb{1}_{\{pred_i\ =\ act_i\}},$$

$$\text{Error Rate} = \frac{n-m}{n}$$

4. Data. The KIDs are available at the respective fund managing firm's website. The sample contains 110 Key Information Documents, and can be found here. First ten Kapitalanlagegesellschaften (KAG), that manage funds registered in austria were chosen. Then the sample was generated via randomly choosing funds on the respective websites of the fund managing firm. The distribution of the SRRI as well as the amount of KIDs per KAG within the sample are displayed below. Further, to obtain a test set, a stratified sampling proccess was applied. Stratified sampling consists of dividing the population (KIDs) into subpopulations (group by KAG) called strata. The random sample is then generated by randomly sampling from each stratum. Accordingly, the training set used to optimize function input parameters includes 90 files, while the test set contains 20 files. The code to generate the test set may be found here. The rationale behind stratified sampling is that the sample is already quite small, random sampling across all files could thus poorly reflect the true document population. Moreover, for one of the functions applied, a color has to be passed as an argument, hence the stratified sampling allows to determine the color that is passed via the file location which drastically reduces KAG identification effort for the document at hand.

The color that is passed to the function represents the shading of the number that represents the SRRI in the document. For different KAGs the color differs across documents, hence having the documents organised by KAG enables the user to pass the color that a particular KAG uses to generate the SRRI graph. As mentioned in the performance diagnostics section, unfortunately, the color sheme does not only vary across different KAGs' documents but also within KAG documents. Meaning that the color used to shade the SRRI graph may look identical, however, the identifying Hex code is different. To resolve this issue, it is possible to pass a vector of colors to the function that performs the scraping via agglomerative clustering.

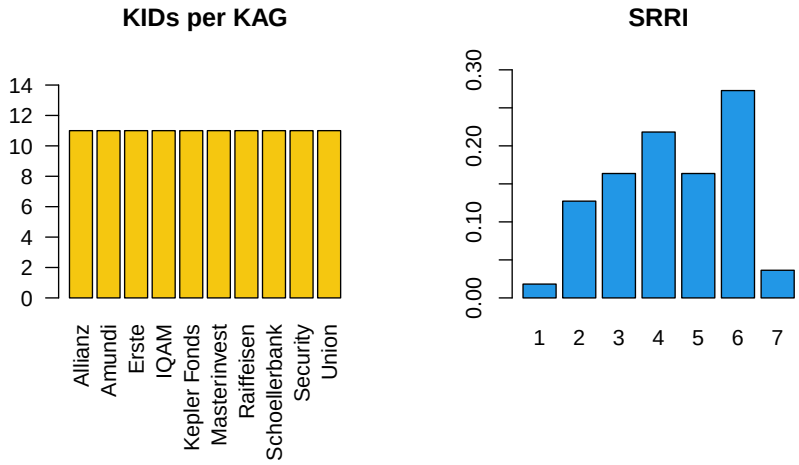

FIG 1. *Number of KIDs per KAG and distribution of the SRRI*

The two depicted plots provide a good summary of the sample at hand. We can immediately observe, that for all fund managing firms, 11 KIDs are available in the sample. Regarding the distribution of the SRRI, the graph depicted on the right enables us to notice that the sample seems to contain very little funds, that are associated with an SRRI of 1 or 7. The low number of funds classified into the first class, may be due to the way the standard deviation of returns is converted into the SRRI. The intervals which are used to determine the SRRI differ in length quite significantly. The first interval for which $0\% \leq \sigma_f < 0.5\%$ spans a range of half a percent, while the intervall for the classi fication into the 6th class $15\% \leq \sigma_f < 25\%$ spans 10%. The other end, namely the SRRI of 7, could be underrepresented because of a lack in demand for such high risk/return products. Of course, this is just mere speculation but interesting nonetheless. In regards to the extraction, to ease illustration of the coming chapters, an example document from Erste can be found here. Additionally, an example of the SRRI graph as it can be found in an Erste KID is provided below. In regards to the generalizability of the representation of the SRRI depiction, each document contains a scale as displayed in Figure 2. The difference between the documents across KAGs is mainly caused by three factors. The position of the graph, the color used to shade the SRRI, the width and height of the graph and seldom the alignment.

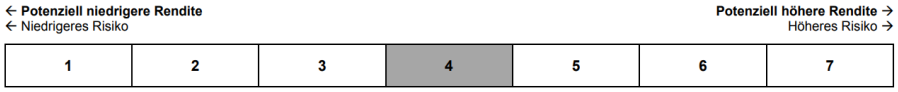

FIG 2. *Example of the SRRI graph from an Erste KID*

5. Extraction.

5.1. *Helper Functions.* Before splitting into the two approaches taken to obtain the SRRI, the helper functions that are utilized in both cases have to be explained. The naive approach relies on knowledge of the background color of the file at hand. In order to obtain this color, one can either assume that it is white, i.e., the color identification code being #ffffffff, or extract the most common color in the pdf. Both options are implemented, since obtaining the background color is a computationally intensive operation. The function called bg_col() first converts the input pdf file into a bitmap utilizing the R package pdftools (Ooms, 2021). A bitmap is an array of dimension $n \times m \times 4$, where n and m can be interpreted as the vertical and horizontal coordinates respectively, while 4 signifies that each color entry is split into four substrings, the so called Hex code of the color. Hex which stands for hexadecimal would usually imply a string of six characters. However, in the case of inclusion of an alpha value the string includes 8 characters. The alpha value is used to alter the transparency of the color. An example for illustrative purposes, should we be interested in the color of the pixel that lies the furthest to the left on the top of the page, we would subset with $n = 1$ and $m = 1$ to obtain the four substrings containing two characters each. Those 4 substrings then represent the color of the pixel. In order to obtain the background color, i.e., the most common color in the document, we thus first flatten the array to an $n \times m$ matrix by pasting together the substrings of the Hex code, to then find the color with the most entries in the matrix.
Besides the backgroundcolor, it is also important to localize the SRRI scale on the page, this task is designated to coord_id(), which calls two additional helper functions. As each document contains many numbers, to identify which of those make up the SRRI graph, scale_cand_cord() first identifies, which single digit numbers could possibly be part of the SRRI (Ooms, 2021). This is achieved by extracting all single digit numbers between 1 and 7 from the document, including the vertical and horizontal position on the page. Summarized in a matrix, this information is then passed to coord_id(), which first finds the absolute difference in vertical/horizontal position of all candidates utilizing abs_dis_vec(), including some tolerance level, to then check if there exist 7 single digit entries with the same vertical/horizontal position. If this is the case, the function has succesfully identified the position of the SRRI. Accordingly, the function then returns a matrix containing the vertical and horizontal position of each scale entry, as well as a string specifying whether the SRRI is displayed vertically or horizontally. Even though not a single occurance of a vertical SRRI depiction is included in the random sample of KIDs, this feature was implemented for the sake of completeness.

5.2. *Naive Approach.* To establish a benchmark for the extraction performance, the naive approach, as the name might suggest, does not make use of any statistical methods. The function SRRI_ext_rec() first localizes each of the 7 SRRI entries on the page via coord_id(). Then the rectangles as displayed in blue in the figure below are built using vertical (yellow) and horizontal (red) tolerances measured from the location of each SRRI entry. Those tolerances are an argument of the function SRRI_ext_rec(), however, default values are implemented that achieve a gross accuracy of 100% on all sample documents that are not scanned. Once the rectangles are built, the bitmap containing all color entries is sliced to generate subsets of the bitmap, that contain all pixels that make up the interior of the blue rectangles displayed below. Subsequently, the percentage of non-background colored pixels is calculated for each of the rectangles. Finally, the rectangle with the least amount of background colored pixels is chosen as the predicted SRRI. The reason for using the percentage of the pixels that are not identical to the background color, is that the pixels that make up the number itself are not background colored. This means, if we were to look fot the rectangle that only contains colors different to the background color, we would find that it does not exist. Accordingly, we may choose a robust way of identifying the rectangle that contains the least background colored pixels, by calculating the relative amount of pixels that are identified by a different Hex code than the background color. The robustness of this approach is further discussed in Section 7.

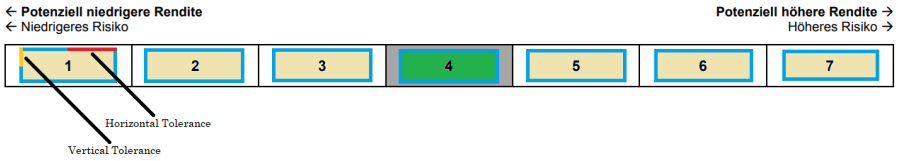

FIG 3. *Illustration of the Naive Approach*

5.3. *Clustering Approach.*

5.3.1. *Motivation.* The second approach is based on agglomerative hierarchical clustering. Before going into detail in regards to the technicalities of this clustering technique, firstly, the idea will be outlined using some graphical illustrations. Assume one knows the color that is used to shade the SRRI, in the case of Figure 3 that would be grey (Hex: #a6a6a6ff). Then an intuitive approach would be to generate a subset of all pixels in the color of the SRRI shade, to then produce some sort of representative value of the horizontal location.

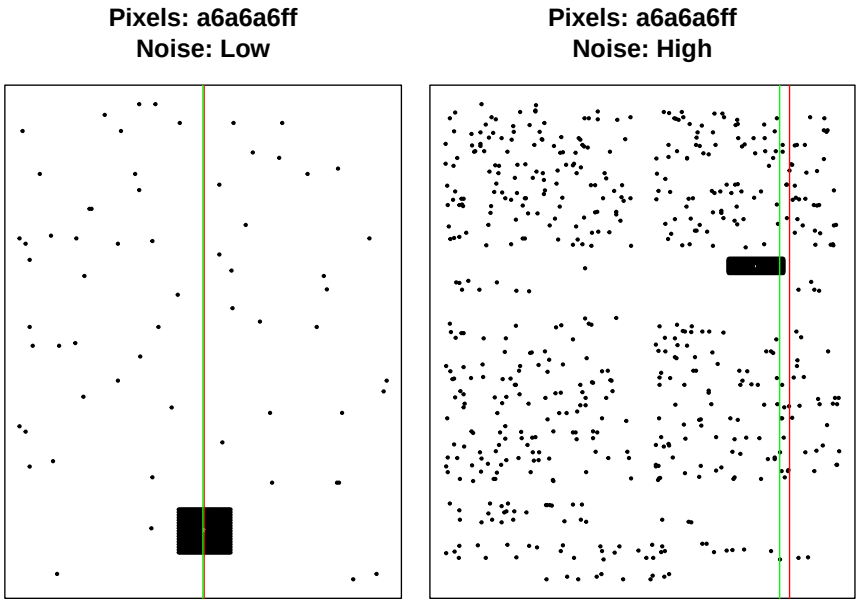

FIG 4. *Pixels in the color of the SRRI shade*

Above we observe the subset of pixels in the color of the SRRI shade for two different documents. Further, the mean of the horizontal coordinates of the pixels is represented by the green line while the median is represented by the red line. Unfortunately, this makes it painfully obvious that neither the median nor the mean of the horizontal coordinates will suffice as a representative of the cloud of pixels in the color of the SRRI shade. To be precise, the median is not a good representative, in a high noise enviroment as displayed on the right. Thus if we manage to reduce said noise to obtain a situation somewhat similar to the one on the left, we may be able to succesfully use the median as a good representative of the horizontal position of the SRRI pixel cloud. This noise reduction may be achieved via unsupervised classification into groups, based on agglomerative hierarchical clustering.

5.3.2. *Agglomerative Hierarchical Clustering.* The goal of noise reduction via clustering is achieved by grouping the pixels. The idea is to cluster the set of all pixels S into subsets $G_1, G_2, ..., G_n$, such that the union of all groups $\cup_{i=1}^{n} G_i$ forms a partition on the set of all pixels, i.e., $S = \cup_{i=1}^{n} G_i$ where $G_i \cap G_j = \emptyset$ for $i \neq j$ (Rokach, 2009). Since the grouping is based on the similarity across pixel coordinates, we need a measure for similarity. In this thesis the euclidean distance is used to determine the distance between pixels and thus their similarity. The rationale behind using the euclidean distance is, that it provides us with the shortest distance between pixels in the two dimensional metric space at hand.
The approach of agglomerative hierarchical clustering starts by initially assigning each pixel its own cluster. Then in each iteration based on the minimal distance, clusters are merged until the desired number of clusters is obtained. In accordance with Adams (2018) the algorithm may be denoted as follows:

Algorithm 1: *Agglomerative Hierarchical Clustering*

Data: N vectors of pixel coordinates $\{x_i\}_{i=1}^{N}$ and p the number of desired clusters.
$\mathbf{X} \leftarrow \emptyset$;
for $k \leftarrow 1...N$ **do**
  $\mathbf{X} \leftarrow \mathbf{X} \cup \{\{x_k\}\}$;
end
$\mathbf{T} \leftarrow \mathbf{X}$;
while $|\mathbf{X}| > p$ **do**
  $G_1^*, G_2^* \leftarrow \underset{G_i, G_j \in \mathbf{X}}{\operatorname{argmin}} \; Dist(G_i, G_j) \; for \; i \neq j$;
  $\mathbf{X} \leftarrow (\mathbf{X} \setminus G_1^*) \setminus G_2^*$;
  $\mathbf{X} \leftarrow \mathbf{X} \cup \{G_1^* \cup G_2^*\}$;
  $\mathbf{T} \leftarrow \mathbf{X} \cup \{G_1^* \cup G_2^*\}$;
end

Given that euclidean distance is used to calculate distances, the first iteration of the while loop is unambigous as only the distance between pixels is required for execution. However, starting from the second iteration we need to find the distance between pixels, other pixels and the first cluster. Hence, $Dist(G_i, G_j)$ needs to be defined for G_i and/or G_j as clusters. In this thesis three different approaches to determine the above described distance are considered.

1. **Average-link:** $D(G_i, G_j) = D(\{x_k\}_{k=1}^{K}, \{y_s\}_{s=1}^{S}) = \frac{1}{KS} \sum_{k=1}^{K} \sum_{s=1}^{S} ||x_k - y_s||$
2. **Single-link:** $D(\{x_k\}_{k=1}^{K}, \{y_s\}_{s=1}^{S}) = \min_{k,s} ||x_k - y_s||$
3. **Complete-link:** $D(\{x_k\}_{k=1}^{K}, \{y_s\}_{s=1}^{S}) = \max_{k,s} ||x_k - y_s||$

At a later stage the link will be chosen based on prediction accuracy. For illustrative purposes, the clustering for different links and varying numbers of final clusters is displayed.

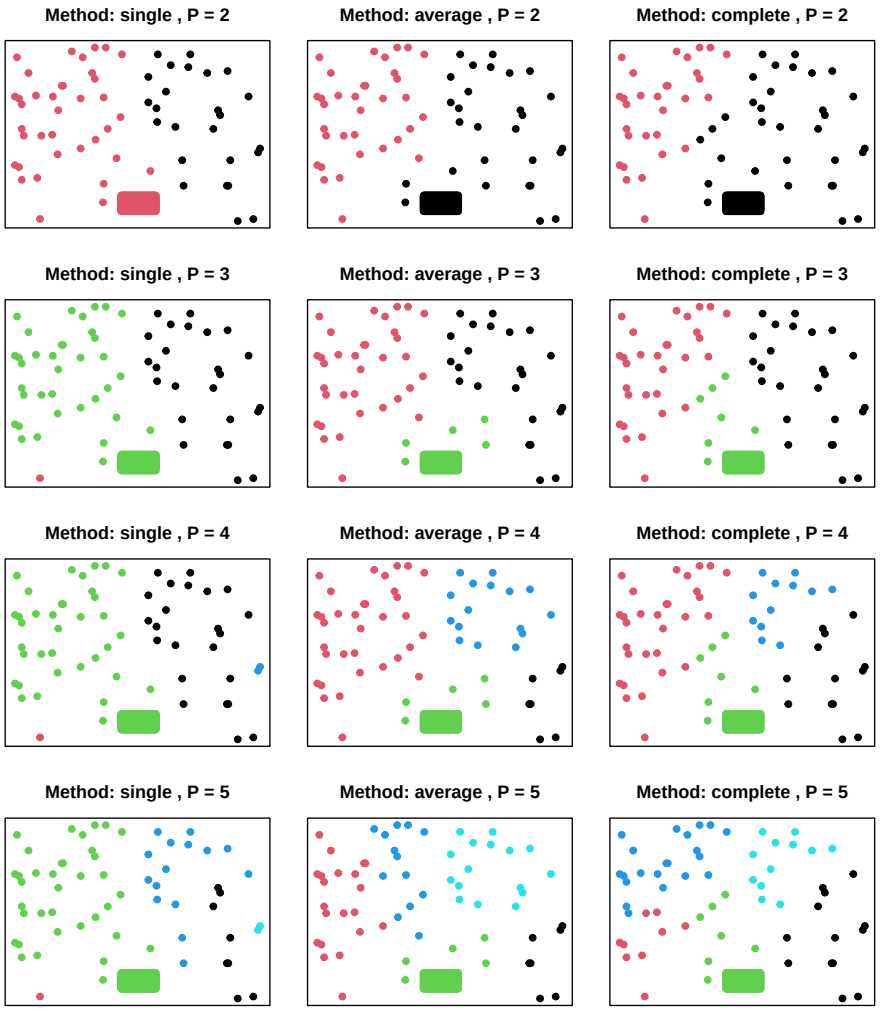

FIG 5. *Example: Different values for P and varying linking methods*

Coming back to the problem at hand, noise reduction, we would like to exclude as many pixels as possible from the coloured square, i.e., keep the cluster that contains the SRRI shade in tact while removing as many pixels that are obviously not part of the SRRI pixel cloud. Figure 5 gives the impression that complete and average linkage seem to be able to better exclude far off pixels from the SRRI pixel cloud. Further, one can observe that for values p between 2 and 5 the SRRI pixel cloud is not split into multiple clusters, thus a higher value for p is probably advisable. Accuracy comparisons accross different values for p and linking methods will show which combination yields the best performance.

5.3.3. *Implementing the AH Clustering Approach.* The function that executes the clustering approach is called SRRI_ext_loc(), as inputs it requires the document and the Hex code of the color of the SRRI pixel shade. Additionally, one may choose the linking method as well as the number of final clusters as explained in the previous subsection. The function first slices the bitmap of the document horizontally above and below the scale that is located by coord_id(). Then using this subset of the original bitmap, all pixels in the color of the SRRI shade are extracted and subsequently clustered into groups in order to reduce noise. Next, all groups containing less than a cutoff value of points are discarded, to then choose the group with the lowest vertical variance in pixel position as the one being the SRRI shade (Dowle & Srinivasan, 2021). The idea is that the removal of all pixel groups below a number of points ensures that clusters with very little observations do not end up having the lowest horizontal variance by chance. Alternatively, one could also generate features that describe each of the clusters, such as number of pixels within a cluster, average distance of points within cluster, average variance of vertical and horizontal pixel coordinates, to then use predictive methods like decision trees or elastic nets to determine which of the clusters most likely contains the SRRI shade.

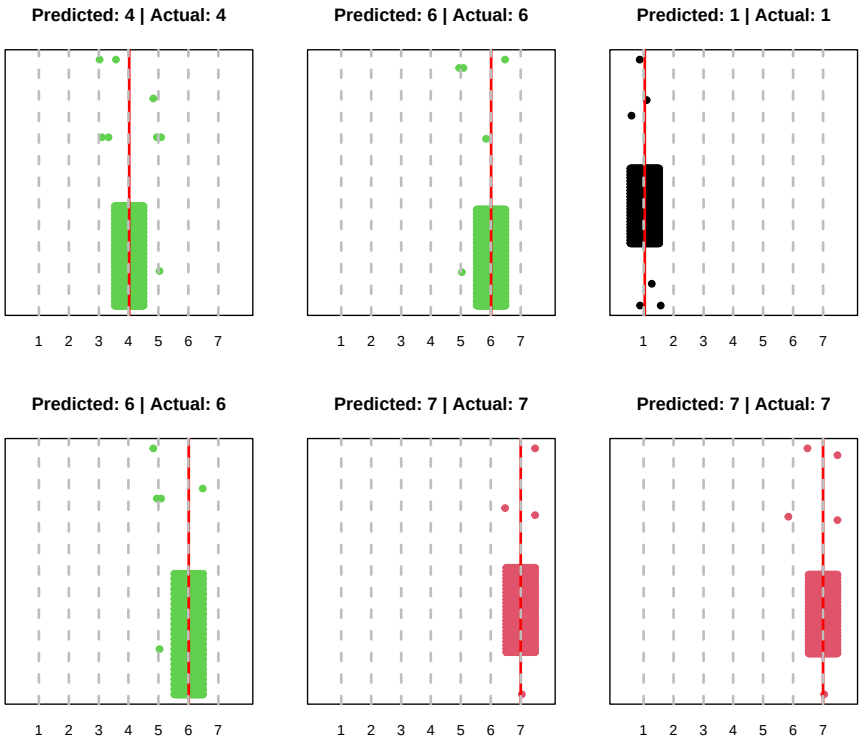

FIG 6. *SRRI extraction via SRRI_ext_loc() using average linkage and* $P = 5$

For the actual proccess of clustering the package fastcluster is used, as hierarchical clustering is not particularly computationally efficient, i.e., depending on the implementation the computational complexity of the algorithm in time is either $\mathcal{O}(n^3)$ or $\mathcal{O}(n^2)$ (Müllner, 2013; Rokach, 2009). For ease of imagination a graphical depiction of Erste KID SRRI extraction via SRRI_ext_loc() is displayed in Figure 6. We observe that the pixel clusters contain very little noise indicating that the function performed as intended.

6. Parameters and Performance.

6.1. *In-sample.* The table below summarizes the performance of the extraction in-sample for different parameter choices. Acc. Gross which represents gross accuracy, measures how accurate the extraction is, based on the total number of predicted cases that did not result in an error. This is the most important measure, since the aim is to automize the task of SRRI extraction. Consequently, the worst outcome is an incorrect readout that does not result in an error and is thus undetectable. Usually, one would aspire to have equal or higher gross performance than a human performing the same task, that way the implementation will drastically increase readout speed, when compared to a human and at the same time the amount of errors would be held constant. Of course, in-sample performance is not representative, hence out of sample performance measures are provided in the next section. Acc. net which stands for net performance, tells us how our algorithms performed after accounting for errors. We oberserve that the naive algorithm outperforms the clustering approach significantly. For the optimal in-sample parameters the single-link clustering approach, utilizing four final clusters and tolerance of 20 pixels results in around 12% less net accuracy.

TABLE 2
SRRI Classification - In-sample Performance

Tolerance	P	Method	Acc. Gross	NA	Acc. Net
-	-	naive	1	0.234	0.766
20	4	single	1	0.356	0.644
20	6	single	1	0.356	0.644
20	8	single	1	0.356	0.644
20	4	average	1	0.356	0.644
20	6	average	1	0.356	0.644
20	4	complete	1	0.356	0.644
20	6	complete	1	0.356	0.644
20	8	average	1	0.467	0.533
30	8	average	1	0.489	0.511
30	8	complete	1	0.500	0.500

Further, it is also interesting to see that the intuition based on Figure 5 is apparently incorrect. The linking method seems to be irrelevant as long as the tolerance is low enough. This might be caused by the fact that the amount of far off pixels that the single link tends to generate more frequently do not have enough weight to create incorrectly classifiied cases.

6.2. *Out-of-sample.* Both the naive- as well as the clustering approach, manage to achieve a gross accuracy of 100%. The net accuracy of the naive method is again drastically higher at 80%, while the clustering approach performs slightly better than in-sample at 70% net accuracy.

7. Diagnostics and Conclusion. Now that the accuracy for both approaches is measured, the error sources have to be discussed. In the test sample of 20 KIDs, the 4 documents for which SRRI_ext_rec() failed the readout of the SRRI, are all scanned pdfs. Consequently, coord_id() fails to identify the position of the SRRI graph. Since SRRI_ext_loc() utilizes the same helper function, this also explains 20% of the failed SRRI extractions of the clustering approach. The remaining 10% are caused by varying colors across different documents, meaning that even though the color of the SRRI shade looks very similar in the documents for the same KAG, the Hex code is actually different. Accordingly, when slicing the bitmap in the extraction proccess the resulting subset is empty. Hence, the required color input is the biggest drawback of the clustering based approach. The reader may rightfully ask, whether this renders the clustering approach obsolete. To an extend it does, for all KIDS that are not scanned the naive approach performs better. However, the clustering approach could in theory allow for the extraction of scanned cases, the idea would be to skip slicing the bitmap above and below the SRRI graph, since localization in scanned pdfs via coord_id() is built on functionalities of the pdftools package that are not available for scanned files. This approach, does however increase the chances of missclassification as the amount of noise increases drastically and for reasons previously outlined the gross accuracy is the most important performance measure. The implementation of this functionality is beyond the scope of this thesis, however, carefully implemented it could potentially increase the net accuracy while hopefully leaving the gross accuracy at the desired level of 100%.

Acknowledgements. I would like to express my appreciation for the feedback I received from the reviewer of my submission. Further, I would also like to thank Univ.-Prof. Dipl. Ing. Dr. Jirak. Both the review as well as the comments provided by Prof. Jirak, allowed me to significantly improve my paper.

APPENDIX A: CODE

A.1. Functions. If possible, the reader is referred to the links of the individual functions provided in the text, as the code is subject to change. The links to GitHub will always link to the most recent versions.

```
# extracts the background color of pdf files

# doc ... input .pdf file
# dpi ... desired input res
# page ... set page
# pass ... assuming the bg-color is white this skips the extraction process

#' @export
bg_col <- function(doc, dpi = dpi, page = page, pass = FALSE){

        # input bitmap
        btmp <- pdftools::pdf_render_page(doc, dpi = dpi, page = page)

        # if the bg-color is assumed to be white without extraction
        if(pass){

                # return white as bg and bitmap
                list("bitmap" = btmp,
                "bgCol" = c("ff", "ff", "ff", "ff"))

        }

        else{

                # flatten array to matrix
                colm <- apply(btmp, 2:3, \(x) paste(x, collapse = ""))

                # identify background color
                colm <- names(which.max(table(colm)))

                # split
                col_split <- gsub("(.{2})", "\\1_", colm) |> strsplit("\\s+") |> unlist()

                # return most common color and bitmap
                list("bitmap" = btmp,
                "bgCol" = col_split)

        }

}

# function that pulls all potential candidates for scale identification of SRRI
# given a KID PDF file

#' @export
scale_cand_coord <- function(file, page = 1){

        # import
        pdf.dat <- pdftools::pdf_data(file)[[page]]

        # filter for 1 to 7
        pdf.dat <- within(pdf.dat,
        ind <- nchar(gsub("\\s", "", text)) == 1 & grepl("[1-7]", text))

        # location
        loc <- with(pdf.dat, pdf.dat[ind, c("x", "y", "text")])

        # return
        return(list(loc, pdf.dat))

}

# helper function for coord.id that calculates the pairwise absolute distance of
# all vector elements

#' @export
abs_dis_vec <- function(x, set.diag = 999){

        # pairwise distance
        dis <- abs(outer(x, x, "-"))

        # set diag
        diag(dis) <- set.diag

        # return
        return(dis)

}

# given a n x 3 matrix of x, y coordinates and the corresponding text this
```

```
# finds the coordinates of the scale of the SRRI no matter if it is horizontal
# or vertical

# loc is the output obtained from scale_cand_coord

# tol ... is the tolerance that is used to match the vertical/horizontal components
# of the scale. The default tolerance is set to three pixels

#' @export
coord_id <- function(loc, tol = 3){

        # all abs diff of horizontal and vertical coords
        lapply(loc[, c("x", "y")], function(x){
                # dis
                tmp <- KIDs::abs_dis_vec(x)

                # name and text
                rownames(tmp) <- loc$text

                # return
                tmp
        }) -> dis


        # identify whether the scale is vertical or horizontal
        lapply(dis, function(x){

                # find all rows which absolute distance is less than three pixels
                apply(x, 1, function(y){
                        sum(y <= tol) == 6
                })

        }) -> ind.int

        # vert or horz
        vh <- which.max(sapply(ind.int, sum))
        ind.fin <- ind.int[[vh]]

        # det vert or horz
        if(vh == 1){dir <- "v"}
        else{dir <- "h"}

        # subset
        scale <- as.data.frame(loc[ind.fin, ])

        # return
        list("Scale" = scale,
        "Alignment" = dir)
}

# as "%in%" does not preserve dimensions we have to write a helper function
# that allows us to use which and obtain the desired array index

#' @export
match_keep_dim <- function(x, y){

        # match
        tmp <- x %in% y

        # reassign dim
        dim(tmp) <- dim(x)

        # return
        return(tmp)

}

# operator
`%IN%` <- function(x,  y){

        # match
        tmp <- x %in% y

        # reassign dim
        dim(tmp) <- dim(x)

        # return
        return(tmp)

}

# doc ... path to file
# col ... HEX code of color shade
# dpi... pixel density
# tol... tolerance when subsetting the bitmap containing the SRRI graph
# p... final number of clusters
# method... linking method forwarded to fastcluster::hclust()
# co... cutoff value for the minimum amount of pixels in SRRI cloud ident.


#' @export
```

```
SRRI_ext_loc <- function(doc, col, dpi = 71.5, tol = 50, p = 5, method = "average", co = 0.2){

        ## DATA ##
        pdf.data  <- KIDs::scale_cand_coord(doc)
        scale.data <- pdf.data[[1]]

        # extract scale
        ext.loc <- KIDs::coord_id(scale.data)

        # horz or vert
        if(ext.loc[[2]] == "h"){dir <- 1} else{dir <- 2}

        # location vertically / horizontally
        cut.off.point <- which.min(sapply(ext.loc[[1]][, c(1, 2)], var))

        # generate bitmap
        bit.map <- pdftools::pdf_render_page(doc, page = 1, dpi = dpi)

        # midpoint of scale
        mid.point <- median(ext.loc[[1]][, cut.off.point])

        # subset bitmap using scale
        bit.map <- bit.map[ , , (mid.point - tol) : (mid.point + tol)]

        ## COLOR ##
        lapply(col, function(x){

                # split HEX
                col.split <- unlist(strsplit(gsub("(.{2})", "\\1_",
                unlist(strsplit(x, "#"))[[2]]), "_"))

                # convert to lower case
                col.split <- tolower(col.split)

                # return
                col.split

        }) -> prep.col.list

        # transpose list
        prep.col.list.t <- data.table::transpose(prep.col.list)

        ## COORDINATES ##

        # Shade
        coo <- which(KIDs::match_keep_dim(bit.map[1, , ], prep.col.list.t[[1]]) &
        KIDs::match_keep_dim(bit.map[2, , ], prep.col.list.t[[2]]) &
        KIDs::match_keep_dim(bit.map[3, , ], prep.col.list.t[[3]]),
        arr.ind = TRUE)

        # stopif no pixels of desired color detected
        if(nrow(coo) < 1) stop("Error: No pixels of given color detected.")

        ## CLASSIFICATIONs ##

        # hierarchical clustering #

        # get grouping
        grps <- fastcluster::hclust(dist(coo, method = "euclidean"), method = method)

        # restrict amnt of groups
        grps <- cutree(grps, k = p)

        # bind
        dat.grps <- as.data.frame(cbind(coo, grps))

        ## IDENTIFY CLUSTER ##

        # find percentage of total points in group
        tbl.rel <- table(dat.grps$grps) / length(dat.grps$grps)

        # grps with less than 20% of all pixels
        excl.nom <- as.numeric(names(tbl.rel[tbl.rel > co]))

        # match and subset
        dat.grps <- dat.grps[dat.grps$grps %in% excl.nom, ]

        # horizontal variance
        which.min(tapply(dat.grps[, 1], dat.grps[, 3], var)) -> rect.grp

        # median
        med.rect.grp <- median(dat.grps[which(dat.grps[, 3] == names(rect.grp)), 1])

        # minimum absolute difference
        dif <- abs(med.rect.grp - ext.loc[[1]][, dir])

        # find minimum
        SRRI <- which.min(dif)

        # return
        list(dif,
```

```
        SRRI,
        dat.grps,
        med.rect.grp,
        ext.loc[[1]][, dir])
}

# doc ... path to file
# dpi ... resolution of the input
# page ... page location of the SRRi shade
# tolh ... horizontal tolerance when extracting the rectangles from the SRRI scale
# tolv ... vertical tolerance when extracting the rectangles from the SRRI scale
# pass ... forwarded to bg_col(), skips bg-color identification

# SRRI extraction based on the relative amount of non-background colored pixels
# in close proximity to the respective SRRI

#' @export
SRRI_ext_rec <- function(doc, dpi = 71.5, page = 1, tolh = 35, tolv = 10.5, pass = FALSE){

        ## load ##

        # bitmap
        btmp_n_bg_col <- KIDs::bg_col(doc, dpi = dpi, page = page, pass = pass)

        # scale cand
        pdf.data  <- KIDs::scale_cand_coord(doc)

        # extract scale
        scale <- KIDs::coord_id(pdf.data[[1]])$Scale

        ## calculate rel. amount of non white pixels in proximity of all scale entries ##

        # build index to cut out respective rectangles around each scale point
        Map(function(x, y){

                # build ranges
                rbind("lower" = floor(x - y),
                "upper" = ceiling(x + y)) |> as.data.frame()


        }, scale[, c("x", "y")], c(tolh, tolv)) |> setNames(c("Horizontal", "Vertical")) -> Sub_ind

        # subset and calculate rel. amount of non-white pixels
        mapply(function(x, y){

                # subset bitmap
                btmp_n_bg_col[["bitmap"]][  , x[1]:x[2], y[1]:y[2]] -> sub.bit

                # calculate percentage of points that are not the background color
                (sub.bit[1, , ] %in% btmp_n_bg_col[["bgCol"]][1] &
                sub.bit[2, , ] %in% btmp_n_bg_col[["bgCol"]][2] &
                sub.bit[3, , ] %in% btmp_n_bg_col[["bgCol"]][3] &
                sub.bit[4, , ] %in% btmp_n_bg_col[["bgCol"]][4]) -> logi

                # rel. amount of non white pixels
                1 - mean(logi)

        }, Sub_ind[[1]], Sub_ind[[2]]) -> p_share

        # return predicted SRRI
        unname(which.max(p_share))

}
```

REFERENCES

Adams, R. P. (2018). Elements of machine learning - hierarchical clustering. *Princeton University*.

Dowle, M., & Srinivasan, A. (2021). *Data.table: Extension of 'data.frame'* [R package version 1.14.0].

FMA. (2011). Ordinance of the financial market authority. *KID-V, BGBI II 265/2011*.

Müllner, D. (2013). Fastcluster: Fast hierarchical, agglomerative clustering routines for r and python. *Journal of Statistical Software*, *53*(9), 1–18.

Ooms, J. (2021). *Pdftools: Text extraction, rendering and converting of pdf documents* [R package version 3.0.1].

R Core Team. (2021). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. Vienna, Austria.

Rokach, L. (2009). A survey of clustering algorithms, 269–298.