Utility functions for experiments on Burrows Delta

Stefan Evert

6 March 2015

Packages

These packages will be available to all documents using the Delta tools and do not need to be imported separately.

```
library(wordspace)
library(cluster)      # for PAM clustering
library(MASS)         # for MDS visualization
library(matrixStats)  # fast row- and column-wise statistics
library(colorspace)   # for well-designed colour palettes
```

Some global variables:

```
## standard colour palettes of Seaborn visualization library
seaborn.pal <- c("#4C72B0","#55A868","#C44E52","#8172B2","#CCB974","#64B5CD")
muted.pal <- c("#4878CF","#6ACC65","#D65F5F","#B47CC7","#C4AD66","#77BEDB")
bright.pal <- c("#003FFF","#03ED3A","#E8000B","#8A2BE2","#FFC400","#00D7FF")
grayscale.pal <- c("black", "#888888", "#555555", "#BBBBBB")
```

The quantitative evaluation of clustering results is based on the adjusted Rand index (Hubert and Arabie 1985). We borrow an implementation from the `mclust` package (which we don't need otherwise) under the name `adjustedRandIndex()`.

```
adjustedRandIndex <- function (x, y) {
  x <- as.vector(x)
  y <- as.vector(y)
  if (length(x) != length(y)) stop("arguments must be vectors of the same length")
  tab <- table(x, y)
  if (all(dim(tab) == c(1, 1))) return(1)
  a <- sum(choose(tab, 2))
  b <- sum(choose(rowSums(tab), 2)) - a
  c <- sum(choose(colSums(tab), 2)) - a
  d <- choose(sum(tab), 2) - a - b - c
  ARI <- (a - (a + b) * (a + c)/(a + b + c + d)) /
         ((a + b + a + c)/2 - (a + b) * (a + c)/(a + b + c + d))
  ARI
}
```

Support functions

Delta measures

The Delta method for authorship attribution computes distances (or similarities) between text samples based on their word frequency profiles. The distances can then be used to cluster texts of unknown authorship or

assign a given text sample to the author of its nearest neighbour in a training corpus. Different variants of Delta (such as Δ_B and Δ_Q) are obtained by setting parameters such as the distance metric, re-scaling of frequency profiles, and the number n_w of most frequent words considered.

The function `delta.dist(M, ...)` encapsulates this aspect of the Delta method. Given a row matrix `M` of frequency profiles (where the columns are feature words ordered according to their assumed relevance), it returns a symmetric distance matrix. Similarity measures (e.g. Cosine Delta) are converted to distances (in this case angular distance) since many clustering and visualization algorithms only accept dissimilarities, and some will even assume metric distances. Further arguments specify the kind of Delta measure to be computed:

- `n`: use first `n` dimensions of the frequency profiles only
- `method`: distance metric; use "manhattan" for original Burrows Delta Δ_B, "euclidean" for (the square root of) Argamon's Quadratic Delta $\sqrt{\Delta_Q}$, "cosine" for Cosine Delta in the form of angular distance $\Delta_\angle = \cos^{-1}(\Delta_C)$
- `p`: exponent of the general "minkowski" metric; defaults to Euclidean distance ($p = 2$)
- `normalize`: normalize feature vectors according to specified metric; note that $\Delta_\angle$ implicitly performs Euclidean normalization, so specifying this option together with `method="cosine"` will have no effect
- `pca=`d: transform vectors into first d principal coordinates using `prcomp()`; since this option is used to implement Rotated Delta Δ_R, optional normalization is applied *after* the PCA transform
- `whiten`: if TRUE, principal coordinates are whitened, i.e. scaled to unit variance; this is the default for Δ_R
- `prenorm`: if TRUE, apply Euclidean normalization *before* the PCA transform (and before centering)
- `transform`: transformation function applied to distance matrix (must preserve structure); defaults to `NULL == identity`
- **TODO:** add further parameters

```
delta.dist <- function (M, n=NA, method="euclidean", p=2, normalize=NA, norm.p=p,
                        pca=NA, whiten=TRUE, prenorm=FALSE, transform=NULL) {
  if (!is.na(n) && n < ncol(M)) M <- M[, 1:n]
  if (!is.na(pca)) {
    if (prenorm) M <- normalize.rows(M, method="euclidean")
    res <- prcomp(M, center=TRUE)
    M <- if (whiten) scaleMargins(res$x, cols=1 / res$sdev) else res$x
    if (pca < ncol(M)) M <- M[, 1:pca]
  }
  if (!is.na(normalize)) M <- normalize.rows(M, method=normalize, p=norm.p)
  res <- dist.matrix(M, method=method, p=p, convert=TRUE)
  if (!is.null(transform)) res <- transform(res)
  res
}
```

Evaluation experiments suggest that Delta measures can be made much more robust – without normalization – if the feature vectors are quantized into overuse vs. underuse. The function `binarize(M, ...)` implements binarization (1 vs. 0), the function `ternarize(M, ...)` implements ternarization into overuse (+1), average use (0) and underuse(−1). Both functions optionally blend from raw feature values to quantized scores over the first `crossover` columns of the matrix.

Ternarization options: - `neutral.p` = proportion of values in neutral range, assuming a standard normal distribution (the default `neutral.p=0` yields a binarized vector) - `neutral.range` = alternatively, cutoff points for the neutral range can be specified directly (overrides `neutral.p`) - `crossover` = blend between original z-scores and ternarized values so that full ternarization is achieved after specified number of MFW

```
ternarize <- function(x, neutral.p=0, neutral.range=-qnorm((1 - neutral.p)/2), crossover=NULL) {
  y <- ifelse(abs(x) <= neutral.range, 0, sign(x))
  if (!is.null(crossover)) {
    if (length(dim(x)) != 2) stop("argument must be a (sparse or dense) matrix if crossover= is specifi
```

```
    n <- seq_len(ncol(x))
    q <- ifelse(n > crossover, 1, n / crossover)
    scaleMargins(y, cols=q) + scaleMargins(x, cols=1-q)
  } else {
    y
  }
}
```

Binarization options: - `threshold` = threshold value θ for the quantization; feature values $x \leq \theta$ are assigned to the underuse category - `underuse.val` = numerical value of the underuse category (typically 0 or -1) - `crossover` = blend between original z-scores and binarized values so that full binarization is achieved after specified number of MFW

```
binarize <- function(x, threshold=0, underuse.val=0, crossover=NULL) {
  y <- ifelse(x <= threshold, underuse.val, 1)
  if (!is.null(crossover)) {
    if (length(dim(x)) != 2) stop("argument must be a (sparse or dense) matrix if crossover= is specifi
    n <- seq_len(ncol(x))
    q <- ifelse(n > crossover, 1, n / crossover)
    scaleMargins(y, cols=q) + scaleMargins(x, cols=1-q)
  } else {
    y
  }
}
```

Logically, ternarization only makes sense if $f_i = 0$ can be interpreted as underuse of a word w_i. This is clearly not the case for lower-frequency content words that occur only in a small number of texts (usually for topical rather than stylistic reasons). We should only distinguish between presence ($+1$) and absence ($0 =$ the regular case) of the word here.

Such considerations suggest a mixed approach that applies either ternarization or binarization depending on the document frequency (df) of an item, implemented by the `binternize(z, f, ...)` function below. In order to compute df reliably, both z-scores z_i (for ternarization) and untransformed raw frequencies f_i have to be passed as arguments. Both arguments must be matrices of the same dimensions.

- `neutral.p`, `neutral.range` = same as in `ternarize()`
- `df` = features with document frequency below this threshold are binarized, otherwise ternarized
- `crossover` = if specified, blend between original z-scores and the fully binternized matrix
- `hapax` = if TRUE, a single occurrence of w_i in a given text is discarded (`f` must contain absolute rather than relative frequencies in this case)

```
binternize <- function(z, f, df=nrow(z)/2, hapax=FALSE, crossover=NULL,
                       neutral.p=1/3, neutral.range=-qnorm((1 - neutral.p)/2)) {
  if (length(dim(z)) != 2) stop("first argument must be a (sparse or dense) matrix")
  if (length(dim(f)) != 2) stop("second argument must be a (sparse or dense) matrix")
  if (!all(dim(z) == dim(f))) stop("the two arguments must be matrices of the same dimensions")
  y3 <- ternarize(z, neutral.range=neutral.range) # ternarized matrix
  y2 <- ifelse(f > 0, 1, 0)                       # binarized matrix
  df.vec <- colSums(y2)                           # document frequencies of all features
  if (hapax) y2 <- ifelse(f > 1, 1, 0)
  y <- scaleMargins(y3, cols=(df.vec >= df)) + scaleMargins(y2, cols=(df.vec < df))
  if (!is.null(crossover)) {
    n <- seq_len(ncol(y))
    q <- ifelse(n > crossover, 1, n / crossover)
    scaleMargins(y, cols=q) + scaleMargins(z, cols=1-q)
  } else {
```

```
    y
  }
}
```

The mixed approach doesn't show good performance in the empirical evaluation, though. **Further investigation is clearly needed.**

Another option is to truncate (or "clamp") the standardized frequencies, e.g. to the range $[-2, 2]$, in order to reduce the influence of "extreme" outlier values.

```
clamp <- function (x, min=-1, max=1) {
  pmax(pmin(x, max), min)
}
```

Visualization

One possibility is to visualize feature vectors directly in the form of spike plots. Since it doesn't seem to be possible to display more than a few hundred dimensions, this approach is mostly useful as an illustration of rescaling techniques. We use a colour palette based on percetual principles to ensure comparable visual contrasts between different pairs of vectors (Zeileis, Hornik, and Murrell 2009).

Arguments and options: - `x`: a vector or matrix of row vectors to be visualized - `col`: a vector of colours for the rows of `x` (recycled as necessary) - `lty`: a vector of line types for the rows of `x` (recycled as necessary) - `lwd`: line width of the spikes (must be a scalar value) - `stride`: stride between vector dimensions (can be set to adjust gaps btw spikes) - `diff`: if TRUE, plot differences between two vectors (`x` must be a two-row matrix, `col` must provide three colours, `lty` is ignored) - `legend`: labels for legend

```
spike.pal <- rainbow_hcl(6, c=60, l=50)[c(1,5,3,2,6,4)] # suitable palette with clear contrasts
spike.plot <- function (x, lwd=1, col=spike.pal, lty="solid", stride=NA,
                        diff=FALSE, legend=NULL, ylim=range(x), ...) {
  if (is.matrix(x)) {
    k <- nrow(x)
    xs <- lapply(1:k, function (i) x[i, ])
  } else {
    k <- 1
    xs <- list(x)
  }
  n <- length(xs[[1]])
  if (diff) {
    if (k != 2) stop("x must be a matrix with two rows if diff=TRUE")
  } else {
    col <- rep(col, length.out=k) # recycle colours / line types and adjust length
    lty <- rep(lty, length.out=k)
  }
  if (is.na(stride)) stride <- if (diff) 1 else k + 1
  x.vals <- stride * ((1:n) - 1) # coordinates of gap before first spike in each group
  plot(0, 0, type="n", xaxs="i", xaxt="n", xlim=c(0, stride * n + 1), xlab="",
       ylim=ylim, ylab="", ...)
  if (diff) {
    ## this is legacy code and may not respect all parameter settings
    y1 <- xs[[1]]
    y2 <- xs[[2]]
    col.vec <- ifelse(y2 > y1, col[3], col[1])
    segments(x.vals + 1, y1, x.vals + 1, y2, col=col.vec, lwd=lwd, lend=1)
    abline(h = 0)
    if (!is.null(legend)) {
```

```
      legend("topright", inset=.02, bg="white", legend=sprintf("%s %s %s", legend[2], c(">", "<"), lege
    }
  } else {
    for (i in 1:k) {
      segments(x.vals + i, rep(0, n), x.vals + i, xs[[i]], col=col[i], lty=lty[i], lwd=lwd, lend=1)
    }
    abline(h=0)
    if (!is.null(legend)) {
      legend("topright", inset=.02, bg="white", legend=legend, col=col, lty=lty, lwd=lwd + 1)
    }
  }
}
# spike.plot(zDE[1:2,1:50], diff=TRUE, lwd=3, legend=c("A", "B"), ylim=c(-2.5, 2.5))  # for testing
```

The authorship attribution task

We can operationalize authorship attribution as a supervised learning task, using a nearest-neighbour (NN) classifier with leave-one-out cross-validation. This operationalization is easy to implement, can be evaluated intuitively in terms of classification accuracy, and enables straightforward discussion and interpretation of classification errors. Unlike a bag-of-words classifier (e.g. MaxEnt or linear SVM), the NN classifier is not prone to overfitting, even if n_w is very large.

The function `nn.classify(M, gold, ...)` defined below takes a matrix of feature vectors (rows) and a corresponding vector of gold labels. It returns the classification accuracy achieved by a NN classifier in leave-one-out cross-validation. Any additional arguments not listed here are passed on to the `delta.dist()` function:

- `predicted`: if TRUE, return vector of classifier predictions rather than evaluation score

```
nn.classify <- function (M, gold, ..., predicted=FALSE) {
  stopifnot(nrow(M) == length(gold))
  gold <- as.character(gold) # in case gold is a factor
  if (is.null(names(gold))) names(gold) <- rownames(M)
  DM <- delta.dist(M, ...)
  ## nearest neighbour of each vector = result of NN classifier in leave-one-out CV
  neighbours <- nearest.neighbours(DM, rownames(DM), n=1, method=method)
  nn.predict <- gold[sapply(neighbours, names)] # classifier predictions
  accuracy <- 100 * sum(nn.predict == gold) / length(gold)
  if (predicted) nn.predict else accuracy
}
```

A more common operationalization casts authorship attribution as a completely unsupervised clustering task. We use partitioning around medians (PAM, described in Kaufman and Rousseeuw 1990, Ch. 2), a flat clustering method that has produced very good results in other NLP tasks. Quantitative evaluation is carried out in terms of the adjusted Rand index between automatic clusters and gold standard categories.

The function `pam.cluster(M, gold, clusters, ...)` defined below has the same arguments and options as `nn.classify()`. It returns the adjusted Rand index as a percentage value.

- `clusters`: desired number of clusters; defaults to the number of distinct categories in the gold standard; if multiple values are specified, best clustering is automatically selected based on average silhouette width
- `predicted`: if TRUE, return data frame with cluster numbers (column `cluster`) and majority cluster labels (column `label`) rather than evaluation score
- `clust.method`: can be used to select other clustering techniques for comparison; most options implement hierarchical clustering with different merging criteria using the `agnes()` function; "hclust" is the Fortran

implementation of Ward clustering in `hclust()`, which – surprisingly – seems to be perform better than `agnes()`

```
pam.cluster <- function (M, gold, ..., clusters=NULL, predicted=FALSE,
                         clust.method=c("pam", "ward", "complete", "single", "average", "hclust")) {
  stopifnot(nrow(M) == length(gold))
  clust.method <- match.arg(clust.method)
  gold <- as.character(gold) # in case gold is a factor
  if (is.null(clusters)) clusters <- length(unique(gold))
  if (is.null(names(gold))) names(gold) <- rownames(M)
  DM <- delta.dist(M, ...)
  if (clust.method == "pam") {
    if (length(clusters) > 1) {
      clustering.list <- lapply(clusters, function (k) pam(DM, k, diss=TRUE, cluster.only=TRUE))
      sil.vec <- sapply(clustering.list, function (cl) summary(silhouette(cl, dmatrix=DM))$avg.width)
      idx <- which.max(sil.vec)
      clusters <- clusters[idx]
      clustering <- clustering.list[[idx]]
    } else {
      clustering <- pam(DM, clusters, diss=TRUE, cluster.only=TRUE)
    }
  } else {
    if (clust.method == "hclust") {
      hcl <- hclust(as.dist(DM), method="ward.D")
    } else {
      hcl <- agnes(DM, diss=TRUE, method=clust.method)
    }
    if (length(clusters) > 1) {
      sil.vec <- sapply(clusters, function (cl) summary(silhouette(cutree(hcl, k=cl), dmatrix=DM))$avg.w
      clusters <- clusters[which.max(sil.vec)]
    }
    clustering <- cutree(hcl, k=clusters)
  }
  if (predicted) {
    ct <- table(gold, clustering) # gold author vs. cluster number
    labels <- rownames(ct)[apply(ct, 2, which.max)] # find majority label for each cluster number
    res <- data.frame(cluster=clustering, label=labels[clustering],
                      row.names=rownames(M), stringsAsFactors=FALSE)
  } else {
    res <- 100 * adjustedRandIndex(clustering, gold)
  }
  sil <- summary(silhouette(clustering, dmatrix=DM))$avg.width
  structure(res, n.clusters=clusters, avg.width=sil)
}
```

The function `evaluate(M, gold, ...)` carries out a combined evaluation with both the NN classifier (accuracy) and the PAM clustering (adjusted Rand index). It returns the evaluation results combined into single-row data frame for printing.

- `n` (n_w), `p` (exponent for Minkowski metric) and `norm.p` (Minkowski metric for normalization) can be vectors, which must be parallel and will be recycled as necessary; returns data frame with multiple rows in this case; note that `p` and `norm.p` only have an effect if `method="minkowski"` and `normalize="minkowski"` are also specified, respectively
- `label` is a string used as as row label for the resulting data frame; the label is automatically extended with the dimensionality n_w of the feature vectors (but not with the other vectorized parameters)

- `do.nn` and `do.cluster` can be used to skip one of the evaluation tasks for improved performance
- further arguments are passed on to `delta.dist()` and `pam.cluster()' (the`clusters‘ value)

```
evaluate <- function (M, gold, n=NA, p=2, norm.p=2, clusters=NULL, label=NULL,
                      do.nn=TRUE, do.cluster=TRUE, clust.method="pam", ...) {
  if (all(is.na(n))) n <- ncol(M)
  n.runs <- max(length(n), length(p), length(norm.p))
  n <- rep(n, length.out=n.runs)
  p <- rep(p, length.out=n.runs)
  norm.p <- rep(norm.p, length.out=n.runs)
  if (!is.null(label)) label <- sprintf("%s | n=%d", label, n)
  res <- data.frame(n=n, p=p, norm.p=norm.p, row.names=label)
  if (do.nn) {
    acc <- sapply(1:n.runs, function (i) {
      nn.classify(M, gold, n=n[i], p=p[i], norm.p=norm.p[i], ...) })
    res$accuracy <- round(acc, 2)
  }
  if (do.cluster) {
    rand <- sapply(1:n.runs, function (i) {
      pam.cluster(M, gold, n=n[i], p=p[i], norm.p=norm.p[i],
                  clusters=clusters, clust.method=clust.method, ...) })
    res$adj.rand <- round(rand, 2)
  }
  res
}
```

References

Hubert, Lawrence, and Phipps Arabie. 1985. “Comparing Partitions.” *Journal of Classification* 2: 193–218.

Kaufman, Leonard, and Peter J. Rousseeuw. 1990. *Finding Groups in Data: An Introduction to Cluster Analysis*. New York: John Wiley; Sons.

Zeileis, Achim, Kurt Hornik, and Paul Murrell. 2009. “Escaping RGBland: Selecting Colors for Statistical Graphics.” *Computational Statistics & Data Analysis* 53: 3259–70.