Anleitung PLSR.Kronsbein

Ein R-Package zur multivariaten Datenanalyse

Klaus Kronsbein
21.08.2020

Inhalt

1 Anleitung R-Package „PLSR.Kronsbein“

Das Package „PLSR.Kronsbein“ dient dazu, ein multivariates Modell mithilfe einer PLSR oder PCR zu erstellen. Hierfür müssen die Daten in ein bestimmtes Format eingelesen werden und können über einige Funktionen vorverarbeitet werden. Zusätzlich ist es möglich, eine PCA durchzuführen, um die Daten zu untersuchen.

Für alle folgenden Beispiele wird der Gasoline Datensatz genutzt (siehe Anhang 4.1 Gasoline Dataset).

1.1 Genutzte R-Packages

Die folgenden R-Packages wurden im „PLSR.Kronsbein“ Package genutzt. Es ist kurz der nutzen für dieses beschrieben.

caret (Max Kuhn (2020)):
: Dieses Package ermöglicht das Aufteilen von Datensätzen in Trainings- und Testdaten für das Modell.

crayon (Gábor Csárdi (2017)):
: Dieses Package ermöglicht farbige Ausgaben in der Konsole.

data.table (Matt Dowle and Arun Srinivasan (2019)):
: Dieses Package ermöglicht die Speicherung von Datensätzen in „Datatables“. „Datatables“ sind notwendig im Package „pls“.

installr (Tal Galili (2019)):
: Aus diesem Package wird nur die Funktion is.empty() genutzt. Diese dient zur Überprüfung von Variablen, ob diese keine Daten enthalten.

kernlab (Alexandros Karatzoglou, Alex Smola, Kurt Hornik, Achim Zeileis (2004)):
: Dieses Package wird für die Kernel-PLS benötigt, um die entsprechenden Kernels zu berechnen.

parallel (R Core Team (2020)):
: Dieses Package ermöglicht die Parallelisierung von Berechnungen, sodass mehrere Prozessorkerne gleichzeitig genutzt werden können. „parallel“ ist ein Teil von R.

pls (Bjørn-Helge Mevik, Ron Wehrens and Kristian Hovde Liland (2019)):
: Mithilfe dieses Packages kann die PCR und PLSR Modellberechnung und Validierung durchgeführt werden. „PLSR.Kronsbein“ baut auf dem Package „pls“ auf.

pracma (Hans W. Borchers (2019)):
: Dieses Package fügt einige mathematische Funktionen hinzu. Genutzt wird rmserr() zur Berechnung des RMSE.

prospectre (Antoine Stevens and Leornardo Ramirez-Lopez (2020)):
: Aus diesem Package wird die Funktion savitzkyGolay() genutzt, um eine Spektrenglättung und -ableitung durchzuführen.

R6 (Winston Chang (2019)):
: Dieses Package dient dazu, eine neue Art von Klassen und Objekten (objektorientierte Programmierung) einzuführen. R6 Klassen ähneln der Klassen in C++ stark.

stats (R Core Team (2020)):
: Dieses Package ermöglicht den Zugriff auf viele Statistikfunktionen in R.

1.2 Vorbereitungen für „PLSR.Kronsbein“

Zuerst muss R installiert werden, hierfür die URL: https://www.R-project.org/ aufrufen. Dort unter Download dem Link zu CRAN folgen und einen der Downloadserver nutzen.

Als nächstes muss RStudio installiert werden, hierfür die URL: http://www.rstudio.com/ aufrufen. Dort unter Download „RStudio Desktop Version“ herunterladen. Ich empfehle die englische Sprachversion zu nutzen.

Nun RStudio öffnen und unter „File à New Project“ ein neues Projekt erstellen. Hierfür ein sinnvolles Verzeichnis erstellen oder öffnen, in dem später alle Arbeiten, während dieses Projektes, durchgeführt werden sollen.

Nun müssen alle benötigten R-Packages installiert werden. Da dies nur einmal gemacht werden muss, ist es sinnvoll, die Installation von der Konsole aus durchzuführen. Hierfür alle folgenden Codezeilen in die Konsole eingeben:

```
install.packages(„crayon“)
install.packages(„caret“)
install.packages(„pls“)
install.packages(„data.table“)
install.packages(„kernlab“)
install.packages(„prospectr“)
install.packages(„pracma“)
install.packages(„R6“)
install.packages(„installr“)
```

Jetzt muss noch das Package PLSR.Kronsbein installiert werden. Hierfür wird die Datei „PLSR.Kronsbein_0.1.0.tar.gz“ benötigt (Versionsnummer kann variieren).
Diese ist auf folgender Webseite verfügbar:
https://github.com/alex146784/PLSR.Kronsbein

Die Datei muss in einem Ordner abgelegt werden.
Nun kann das PLSR.Kronsbein Package durch Eingabe folgender Zeile in der Konsole installiert werden. Hierbei ist der Pfad zur Datei entsprechend des Ablageortes anzupassen (Wichtig: R nutzt Slash „/“ anstatt dem Backslash „\“ für die Pfadangabe). Außerdem ist es eventuell notwendig die Versionsnummer anzupassen.

```
install.packages("C:/Bachelorarbeit/R/00_SOURCECODE/PLSR.Kronsbein_0.1.0.tar.gz",  repos = NULL)
```

Die folgenden Dinge sollten bei jeder Ausführung eines Skripts durchgeführt werden, daher ist es sinnvoll, diese an den Anfang des auszuführenden Skriptes zu schreiben. Hierfür unter „File à New File à R Script“ eine neue Datei öffnen und diese bei Bedarf abspeichern. Zuerst sollte am Anfang immer das Arbeitsverzeichnis (engl. working directory) gesetzt werden, da es sonst zu Fehlern kommen kann. Hierfür am besten das Verzeichnis, welches oben bei der Erstellung des Projektes angegeben wurde, nutzen.

```
setwd("C:/Bachelorarbeit/R/gasoline")
```

Nun kann das Package PLSR.Kronsbein aktiviert werden, dabei werden automatisch alle anderen benötigten Packages aktiviert. Zusätzlich ist es nützlich, die Funktion gener.prep() auszuführen (für Details siehe 1.7.1 gener.prep()).

```
library(PLSR.Kronsbein)
gener.prep()
```

Nun kann mit den folgenden Funktionen gearbeitet werden. Hierfür ist es aber notwendig, grundlegende Programmier-, besser noch grundlegende R-Kenntnisse zu besitzen. Hierbei ist es entscheidend, Daten in das richtige Format bringen zu können, falls keine der Einlesefunktionen genau passt.
Für Einsteiger kann das Buch „Hands-On Programming with R“ (Grolemund 2019) hilfreich sein. Zu finden ist dies unter folgendem Link: https://rstudio-education.github.io/hopr/index.html.

1.3 Einlesefunktionen

Die Einlesefunktionen dienen dazu die Daten einzulesen und ins richtige Format zu bringen. Dieses Datenformat wird im folgendem „PLSR.Kornsbein Datensatz“ genannt. Die Daten liegen dann im Speicher als eine Liste mit allen Informationen (siehe1.9 PLSR.Kronsbein Datensatz).

Aufgabe	**Funktionsaufruf**	**Name im Speicher**	**Quellcodedatei**
Generelle Einlesefunktion	dataprep.plsr()	dataprep.plsr	data_preparation.R
Einlesefunktion für als .csv vorliegende Daten	fread.dataprep.plsr()	dataprep.plsr	data_preparation.R
Einlesefunktion für als .csv vorliegende Daten, welche aus Export aus OPUS stammen	fread.MPA2.dataprep.plsr()	dataprep.plsr	data_preparation.R

Tabelle 1: Übersicht Einlesefunktionen

1.3.1 dataprep.plsr()

Die Funktion dataprep.plsr() ist die Allgemeinste der Einlesefunktionen. Sie wird von den anderen Einlesefunktionen als interne Funktion aufgerufen. Sie kann aber auch vom Nutzer aufgerufen werden, hierfür müssen die Spektrendaten (X-Daten) und Responsedaten (Y-Daten) als Matrizen an die Funktion übergeben werden. Sinnvoll ist dies, wenn keine andere Einlesefunktion passt, dann kann der Nutzer die Daten selbst importieren und über diese Funktion in das richtige Format bringen.

Parameter

X.values (numeric matrix):

Hier werden die X-Daten (die Y-Daten der Spektren) als Matrix für die spätere Modellbildung übergeben. Die Spalten sind die einzelnen Variablen und die Reihen die einzelnen Spektren.

Y.values = NULL(numeric matrix):

Hier werden die Y-Daten als Matrix für die spätere Modellbildung übergeben. Die Spalten sind die vorherzusagenden Variablen (nur 1 Spalte für PLSR1, mehrere für PLSR2) und die Reihen die entsprechenden Werte für jedes Spektrum (wichtig: Reihenfolge muss zu Spektren passen). Es ist auch möglich die Y-Daten weg zu lassen, wenn der Datensatz nur für eine PCA oder als Hilfsdatensatz in der IIF-Korrektur verwendet wird.

wavelengths (numeric vector):

Hier werden die X-Daten der Spektren übergeben. Also die Wellenlängen oder Wellenzahlen als Vektor aus numerischen Werten. Wenn nicht spektrale multivariate Datensätze genutzt werden, dann können auch einfach die Variablen durchnummeriert werden (1 bis Anzahl der Variablen).

centerY = TRUE (boolean):

Parameter, durch den die Y-Daten zentriert werden (wird bei allen Ausgaben rückgängig gemacht). centerY sollte auf TRUE belassen werden, da sonst die plsr.ex() Funktion nicht richtig arbeitet.

originalY = TRUE (boolean):

Parameter, welcher bestimmt ob die Ausgabefunktionen mit den originalen Y-Werten arbeiten oder transferierte Werte nutzen. Normalerweise sollte dieser Wert auf TRUE belassen werden, sodass die originalen Werte genutzt werden.

add.infos = NULL (list):

Parameter, welcher vom Nutzer ignoriert werden kann. Hier werden Information von anderen Einlesefunktionen für die Auswertung übergeben, falls diese intern aufgerufen werden sollte.

UnitspecY = „“ (character):

Hier kann den Spektren-Y-Daten (X.values) eine Einheit/Beschriftung zugewiesen werden. Wenn in Extinktion gemessen wurde, sollte "Extinction" angegeben werden und falls in Transmission gemessen wurde, "Transmission".

UnitspecX = „“ (character):

Hier kann den Spektren-X-Daten (wavelengths) eine Einheit/Beschriftung zugewiesen werden. Wenn als Einheit die Wellenlänge genutzt wurde, sollte "Wavelength [nm]" angeben werden und falls die Wellenzahl genutzt wurde, "Wavenumber [cm^-1]".

repetitions = NULL (numeric):

Hier kann die Anzahl an Wiederholungen von gleichen Datenpunkten als numerischer Wert angegeben werden. Zum Beispiel, wenn die gleiche Probe mehrmals gemessen wurde. Dies wird innerhalb anderer Funktionen beachtet. Auf NULL belassen, wenn keine Wiederholungen vorhanden sind.

printplots.TF = FALSE (boolean):

Wenn TRUE, dann werden die Spektren im Plots Fenster angezeigt.

colnames.X = NULL (character vector):

Falls die Matrix, übergeben durch X.values, keine Spaltennamen enthält, dann können sie hier als Vektor hinzugefügt werden.

rownames.X.Y = NULL (character vector):

Falls die Matrizen, übergeben durch X.values und Y.values, keine Reihennamen enthalten, dann können sie hier als Vektor hinzugefügt werden.

colnames.Y = NULL (character):

Falls die Matrix, übergeben durch Y.values, keine Spaltennamen enthält, dann können sie hier als Vektor hinzugefügt werden.

Beispiel

```
wavelengths <- seq(from = 900, to = 1700, by = 2)
Gasoline <- dataprep.plsr(X.values = gasoline$NIR, Y.values = matrix(gasoline$octane, ncol = 1),
    wavelengths = wavelengths, UnitspecY = "Extinction" , UnitspecX = "Wavelength [nm]",
    colnames.X = paste0("NIR.",wavelengths, "nm"))
```

Zusätzliche Erklärung:

Im Beispiel wird ein Vektor wavelengths erzeugt, mit der entsprechenden Wellenlänge für jede Variable. Als X- und Y-Werte werden die entsprechenden Teile des „gasoline“ Datensatzes genutzt. Der Vektor gasoline$octane muss noch in eine Matrix mit einer Spalte transformiert werden, dies geschieht durch matrix(). Die Spaltennamen von X wurden durch colnames.X geändert. Hierfür wurde der Vektor durch paste0() mit anderen Zeichenketten zusammengefügt. Der erhaltene Datensatz wird dann unter der Variable Gasoline abgespeichert.

1.3.2 fread.dataprep.plsr()

Die Funktion fread.dataprep.plsr() dient dazu, Daten aus .csv Dateien einzulesen. Die X- und Y-Daten müssen entweder gemeinsam in einer Datei stehen oder in zwei getrennten Dateien. Welche Spalten aus den .csv Dateien genutzt werden sollen, kann über X.col und Y.col bestimmt werden. Wichtig ist, dass man sich gerade in dem Arbeitsverzeichnis befindet, indem auch die .csv Dateien liegen. Dies ist normalerweise der Ort, wo auch die .Rproj Datei liegt, wenn er nicht geändert wurde.

Parameter

file = NULL (character):

Name der .csv Datei, welche die X- und Y-Daten für die spätere Modellbildung enthält. Welche Spalten die X- und Y-Daten enthalten muss, wird über X.col und Y.col spezifiziert werden.

Xfile = NULL (character):

Name der .csv Datei, welche nur die X-Daten für die spätere Modellbildung enthält. Wenn nur ein Teil der Spalten genutzt werden soll, können diese über X.col ausgewählt werden.

Yfile = NULL (character):

Name der .csv Datei, welche nur die Y-Daten für die spätere Modellbildung enthält. Wenn nur ein Teil der Spalten genutzt werden soll, können diese über Y.col ausgewählt werden.
Es ist auch möglich die Y-Daten weg zu lassen, wenn der Datensatz nur für eine PCA oder als Hilfsdatensatz in der IIF-Korrektur verwendet wird.

X.col = NULL (numeric vector):

Ein Vektor, der diejenigen Spalten enthält, welche als X-Daten genutzt werden sollen.

Y.col = NULL (numeric vector):

Ein Vektor, der diejenigen Spalten enthält, welche als Y-Daten genutzt werden sollen.

wavelengths = NULL (numeric vector):

Hier werden die X-Daten der Spektren übergeben. Also die Wellenlänge oder Wellenzahl als Vektor aus numerischen Werten. Wenn nicht spektrale multivariate Datensätze genutzt werden, dann können auch einfach die Variablen durchnummeriert werden (1 bis Anzahl der Variablen).

wavelengths.file = NULL (character):

Alternative zu wavelengths. Hier können die X-Daten der Spektren als .csv Datei übergeben werden.

centerY = TRUE (boolean):

Parameter, durch den die Y-Daten zentriert werden (wird bei allen Ausgaben rückgängig gemacht). centerY sollte auf TRUE belassen werden, da sonst die plsr.ex() Funktion nicht richtig arbeitet.

originalY = TRUE (boolean):

Parameter, welcher bestimmt, ob die Ausgabefunktionen mit den originalen Y-Werten arbeiten oder transferierte Werte nutzen. Normalerweise sollte dieser Wert auf TRUE belassen werden, sodass die originalen Werte genutzt werden.

UnitspecY = „“ (character):

Hier kann den Spektren-Y-Daten (X.values) eine Einheit/Beschriftung zugewiesen werden. Wenn in Extinktion gemessen wurde, sollte "Extinction" angegeben werden und falls in Transmission gemessen wurde, "Transmission".

UnitspecX = „“ (character):

Hier kann den Spektren-X-Daten (wavelengths) eine Einheit/Beschriftung zugewiesen werden. Wenn als Einheit die Wellenlänge genutzt wurde, sollte "Wavelength [nm]" angeben werden und falls die Wellenzahl genutzt wurde, "Wavenumber [cm^-1]".

repetitions = NULL (numeric):

Hier kann die Anzahl an Wiederholungen von gleichen Datenpunkten als numerischer Wert

angegeben werden. Zum Beispiel, wenn die gleiche Probe mehrmals gemessen wurde. Dies wird bei anderen Funktionen dann beachtet. Auf NULL belassen, wenn keine Wiederholungen vorhanden sind.

printplots.TF = FALSE (boolean):

Wenn TRUE, dann werden die Spektren im Plots Fenster angezeigt.

colnames.X = NULL (character):

Falls die Matrix, übergeben durch X.values, keine Spaltennamen enthält, dann können sie hier als Vektor hinzugefügt werden.

rownames.X.Y = NULL (character):

Falls die Matrizen, übergeben durch X.values und Y.values, keine Reihennamen enthalten, dann können sie hier als Vektor hinzugefügt werden.

colnames.Y = NULL (character):

Falls die Matrix, übergeben durch Y.values, keine Spaltennamen enthält, dann können sie hier als Vektor hinzugefügt werden.

Beispiel:

```
wavelengths <- seq(from = 900, to = 1700, by=2)
Gasoline <- fread.dataprep.plsr(file = "gasoline.csv" ,X.col = 3:403, Y.col = 2, wavelengths = wavelengths,
    UnitspecY = "Extinction", UnitspecX = "Wavelength [nm]")
```

Zusätzliche Informationen:

Um die Funktion, wie dargestellt anwenden zu können, muss die Datei „gasoline.csv“ im Arbeitsverzeichnis liegen, wobei die 2. Spalte die Oktanzahlen und die 3. bis 403. Spalte die NIR-Spektrendaten enthalten muss. Die Tabelle sollte so aussehen, wie in Abbildung 1 gezeigt. In der ersten Zeile stehen die Überschriften der Spalten, welche auch so übernommen werden.

	A	B	C	D	E	F	G
1		octane	NIR.900 nm	NIR.902 nm	NIR.904 nm	NIR.906 nm	NIR.908 nm
2	1	85.3	-0.050193	-0.045903	-0.042187	-0.037177	-0.033348
3	2	85.25	-0.044227	-0.039602	-0.035673	-0.030911	-0.026675
4	3	88.45	-0.046867	-0.04126	-0.036979	-0.031458	-0.02652
5	4	83.4	-0.046705	-0.04224	-0.038561	-0.034513	-0.030206
6	5	87.9	-0.050859	-0.045145	-0.041025	-0.036357	-0.032747

Abbildung 1: Ausschnitt von „gasoline.csv“

1.3.3 fread.MPA2.dataprep.plsr()

fread.MPA.2.dataprep.plsr() ist eine, an die durch OPUS exportierten Daten des FT-NIR Gerätes MPA2, angepasste Einlesefunktion. Die Daten müssen vorher durch ein VBS-Skript (erhältlich durch den Bruker Support) in OPUS exportiert werden. Danach liegen die Spektren als einzelne .csv Dateien vor, diese sollten am besten in einem Unterordner des Arbeitsverzeichnisses gespeichert werden. Die Y-Daten müssen in eine extra .csv Datei gespeichert werden.

Parameter

dirspectra (character):

Hier wird der Pfad angegeben, in dem die einzelnen Spektrendateien als .csv gespeichert wurden. Dabei müssen sie nicht zwingend die Endung .csv besitzen, diese wird durch das Exportskript nicht ergänzt. Wenn sich die Spektrendaten in einem Unterordner des Arbeitsverzeichnisses befinden, dann ist es ausreichend, den Namen dieses Ordners zu übergeben. Zu beachten ist, dass die Dateien die richtige Reihenfolge aufweisen, sie werden in alphabetischer Reihenfolge eingelesen (siehe 1.12.5 Richtige Sortierung von Dateien (fread.MPA2.dataprep.plsr())).

Yfile = NULL (character):

Der Name der .csv Datei wird hier übergeben, welche nur die Y Daten für die spätere Modellbildung enthält. Wenn nur ein Teil der Spalten genutzt werden soll, können diese über Y.col ausgewählt werden. Die Reihenfolge muss mit der Spektrenreihenfolge in dirspectra übereinstimmen. Es ist auch möglich die Y-Daten weg zu lassen, wenn der Datensatz nur für eine PCA oder als Hilfsdatensatz in der IIF-Korrektur verwendet wird.

Y.col = NULL (numeric vector):

Ein Vektor, der diejenigen Spalten enthält, welche als Y-Daten genutzt werden sollen.

repeatYvalues = FALSE (boolean):

Wenn TRUE, dann wird jeder Wert in Y.file so oft wiederholt wie in repetitions angegeben. Hilfreich, wenn für die gleiche Probe mehrere Spektren aufgenommen werden und nicht jeder Y-Wert mehrmals in der .csv Datei steht.

centerY = TRUE (boolean):

Parameter, durch den die Y-Daten zentriert werden (wird bei allen Ausgaben rückgängig gemacht). centerY sollte auf TRUE belassen werden, da sonst die plsr.ex() Funktion nicht richtig arbeitet.

originalY = TRUE (boolean):

Parameter, welcher bestimmt, ob die Ausgabefunktionen mit den originalen Y-Werten arbeiten oder transferierte Werte nutzen. Normalerweise sollte dieser Wert auf TRUE belassen werden, sodass die originalen Werte genutzt werden.

UnitspecY = „“ (character):

Hier kann den Spektren-Y-Daten (X.values) eine Einheit/Beschriftung zugewiesen werden. Wenn in Extinktion gemessen wurde, sollte "Extinction" angegeben werden und falls in Transmission gemessen wurde, "Transmission".

UnitspecX = „“ (character):

Hier kann den Spektren-X-Daten (wavelengths) eine Einheit/Beschriftung zugewiesen werden. Wenn als Einheit die Wellenlänge genutzt wurde, sollte "Wavelength [nm]" angeben werden und falls die Wellenzahl genutzt wurde, "Wavenumber [cm^-1]".

repetitions = NULL (numeric):

Hier kann die Anzahl an Wiederholungen von gleichen Datenpunkten als numerischer Wert angegeben werden. Zum Beispiel, wenn die gleiche Probe mehrmals gemessen wurde. Dies wird bei anderen Funktionen dann beachtet. Auf NULL belassen, wenn keine Wiederholungen vorhanden sind.

printplots.TF = FALSE (boolean):

Wenn TRUE, dann werden die Spektren im Plots Fenster angezeigt.

colnames.X = NULL (character):

Falls die Matrix, übergeben durch X.values, keine Spaltennamen enthält, dann können sie hier als Vektor hinzugefügt werden.

rownames.X.Y = NULL (character):

Falls die Matrizen, übergeben durch X.values und Y.values, keine Reihennamen enthalten, dann können sie hier als Vektor hinzugefügt werden.

colnames.Y = NULL (character):

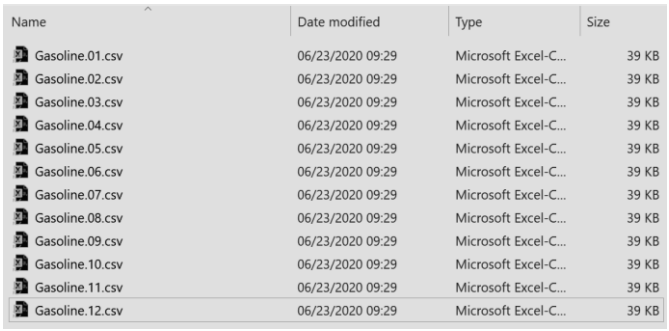
Falls die Matrix, übergeben durch Y.values, keine Spaltennamen enthält, dann können sie hier als Vektor hinzugefügt werden.

Beispiel

```
Gasoline <- fread.MPA2.dataprep.plsr(dirspectra = "Gasoline_spectra", Yfile = "Octane.csv",
    UnitspecY = "Extinction", UnitspecX = "Wavelength [nm]", Y.col = 2 )
```

Zusätzliche Informationen:

Die Spektren liegen im Unterordner der Arbeitsverzeichnisses „Gasoline_spectra“. Es können nur .csv Dateien eingelesen werden (nur das Dateiformat ist entscheidend, nicht die Dateiendung). Hierbei ist wichtig, dass die Benennung alphabetisch und nummerisch die richtige Reihenfolge der Spektren besitzt (siehe Abbildung 2, sodass die Spektren zu den Y-Werten aus „Octane.csv“ zusammenpassen. Der Inhalt einer Spektren Datei muss so, wie in Abbildung 3 dargestellt, aussehen. Die erste Spalte muss den X-Wert des Spektrums enthalten und die zweite Spalte den Y-Wert. Die Datei für die Y-Daten kann so, wie in Abbildung 4 dargestellt, aussehen. In diesem Fall wird nur die zweite Spalte von Octane.csv benötigt, dies wird durch Y.col = 2 angegeben.



Name	Date modified	Type	Size
Gasoline.01.csv	06/23/2020 09:29	Microsoft Excel-C...	39 KB
Gasoline.02.csv	06/23/2020 09:29	Microsoft Excel-C...	39 KB
Gasoline.03.csv	06/23/2020 09:29	Microsoft Excel-C...	39 KB
Gasoline.04.csv	06/23/2020 09:29	Microsoft Excel-C...	39 KB
Gasoline.05.csv	06/23/2020 09:29	Microsoft Excel-C...	39 KB
Gasoline.06.csv	06/23/2020 09:29	Microsoft Excel-C...	39 KB
Gasoline.07.csv	06/23/2020 09:29	Microsoft Excel-C...	39 KB
Gasoline.08.csv	06/23/2020 09:29	Microsoft Excel-C...	39 KB
Gasoline.09.csv	06/23/2020 09:29	Microsoft Excel-C...	39 KB
Gasoline.10.csv	06/23/2020 09:29	Microsoft Excel-C...	39 KB
Gasoline.11.csv	06/23/2020 09:29	Microsoft Excel-C...	39 KB
Gasoline.12.csv	06/23/2020 09:29	Microsoft Excel-C...	39 KB

Abbildung 2: Beispielspektrendatensatz fread.MPA2.dataprep.plsr()

	A	B
1	11544	-0.23539
2	11540	-0.23535
3	11536	-0.23527
4	11532	-0.23521
5	11528	-0.23516
6	11524	-0.23511
7	11520	-0.2351
8	11516	-0.23511
9	11512	-0.23511
10	11508	-0.23508
11	11504	-0.23505
12	11500	-0.23505

Abbildung 3: Ausschnitt Gasoline.01.csv

	A	B
1	Probe	octane
2	1	85.3
3	2	85.25
4	3	88.45
5	4	83.4
6	5	87.9
7	6	85.5
8	7	88.9
9	8	88.3
10	9	88.7
11	10	88.45
12	11	88.75
13	12	88.25

Abbildung 4: Ausschnitt Octane.csv

1.4 Datenvorbehandlungsfunktionen

Die Datenvorbehandlungsfunktionen dienen dazu, die Daten so zu verarbeiten, dass die Modellanpassung eine bessere Vorhersagegenauigkeit erreichen kann. Dies muss aber nicht der Fall sein, es können dabei auch wichtige Informationen verloren gehen.

Aufgabe	**Funktionsaufruf**	**Name im Speicher**	**Quellcodedatei**
Zentrieren der X-Daten	centerX()	centerX	centerX_scaleX_ standardizeX.R
Skalieren der X-Daten	scaleX()	scaleX	centerX_scaleX_ standardizeX.R
Standardisieren der X-Daten	standardizeX()	standardizeX	centerX_scaleX_ standardizeX.R
Zentrieren der Y-Daten	centerY()	centerY	centerY_scaleY_ standardizeY.R
Skalieren der Y-Daten	scaleY()	scaleY	centerY_scaleY_ standardizeY.R
Standardisieren der Y-Daten	standardizeY()	standardizeY	centerY_scaleY_ standardizeY.R
Spektrum beschneiden	cutspectrum()	cutspectrum	cut_spectrum.R
MSC Korrektur durchführen	msc.adapted()	MSC.adapted	MSC_functions.R
Polynomial Baseline Correction durchführen	polynomial_baseline_ correction()	pbc	polynomial_base line_correction.R
Spektrum durch Polynomglättung glätten (Savitzky Golay)	smoothspectrums. polynomial()	smoothing.sg	smoothing_ derivation_ savitzkygolay.R
Spektrum durch Polynomglättung glätten und ableiten (Savitzky Golay)	derivatespectrums()	derivation.sg	smoothing_ derivation_ savitzkygolay.R
Spektren mithilfe des Mittelwertes glätten	smoothspectrums. mean()	smoothing.mean	smoothing_ mean.R
SNV Korrektur durchführen	SNV()	SNV	SNV.R
Umrechnen der X-Achse der Spektren von der Wellenzahl [cm^-1] in die Wellenlänge [nm]	Wavenumber_to_ Wavelength()	Wavenumber_to_ Wavelength	spect_ transformation_ UnitspecX.R
Umrechnen der X-Achse der Spektren von der Wellenlänge [nm] in die Wellenzahl [cm^-1]	Wavelength_to_ Wavenumber()	Wavelength_to_ Wavenumber	spect_ transformation_ UnitspecX.R
Umrechnen der Y-Achse der Spektren von der Transmission in die Extinktion	Transmission_to_ Extinction()	Transmission_to_ Extinction	spect_transforma tion_UnitspecY.R
Umrechnen der Y-Achse der Spektren von der Extinktion in die Transmission	Extinction_to_ Transmission()	Extinction_to_ Transmission	spect_transforma tion_UnitspecY.R
Teile die Daten in Test und Trainingsdaten auf	generate.split_and_ splitdata. externalvalidation()	splitdata.exval	splitdata.R
Variablenselektion anhand verschiedener Algoritmen (CARS, Procrustes, PCA-Procrustes)	variableselection()	CARS; Procrustes; PCA.procrustes	variable selection.R
Eine gespeicherte Variablenselektion laden	load. variableselection()	load. variableselection	variableselection_ functions.R
Reduktion der Variablen, zur Beschleunigung folgender Berechnungen	reduce_variables_ spectra()	reduce_variables_ spectra	reduce_variables_ spectra.R

Die Mittelwertspektren aus einer festgelegten Anzahl an Spektren berechnen	calc.meanspectra()	calc.meanspectra	mean_spectra.R
IIR-Korrektur der Daten (Interferenzen modellieren und entfernen)	IIR.correction()	IIR.correction	IIR_correction.R
Transformierung der Y-Daten	Ydata_transformation()	Ydata_ tranformation	Ydata_trans formation.R
Den Kalibrierbereich begrenzen	reduce.calibration range()	reduce.calibration range	rduce_calibration range.R

Tabelle 2: Übersicht Datenvorbehandlungsfunktionen

1.4.1 centerX()

centerX() ist eine Funktion, um die X-Daten zu zentrieren. Für nähere Informationen siehe 2.3.1 Zentrierung, Standardisierung & Skalierung.

Parameter

data (PLSR.Kronsbein Datensatz):

Datensatz, welcher durch eine Einlesefunktion erzeugt wurde, übergeben.

printplots.TF = FALSE (boolean):

Wenn TRUE, dann werden die Spektren vor und nach der Datenvorbehandlung im Plots Fenster angezeigt.

savedata.TF = TRUE (boolean):

Dieser Wert sollte nicht auf FALSE gesetzt werden, ansonsten erscheint diese Datenvorbehandlung nicht in der Auswertung. Wird benötigt für die Vorhersagefunktion predict.ex().

referencedata = NULL (numeric vector):

Dieser Parameter wird vom Nutzer nicht benötigt. Er dient in der Vorhersagefunktion dafür die originalen Mittelwertspektren zu übergeben.

Beispiel

```
Gasoline <- centerX(data = Gasoline)
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen.
Das Ergebnis der Funktion ist in Abbildung 5 zu sehen.

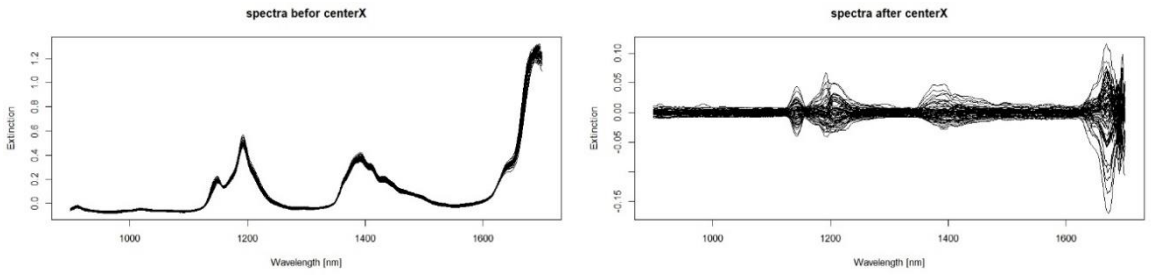


Abbildung 5: Spektren bevor (links) und nach (rechts) centerX()

1.4.2 scaleX()

scaleX() ist eine Funktion, um die X-Daten zu skalieren/gewichten. Für nähere Informationen siehe 2.3.1 Zentrierung, Standardisierung & Skalierung.

Parameter

data (PLSR.Kronsbein Datensatz):

Datensatz, welcher durch eine Einlesefunktion erzeugt wurde, übergeben.

scales = NULL (numeric vector):

Hier werden die Skalierungen/Gewichtungen für die X-Variablen als Vektor übergeben.

printplots.TF = FALSE (boolean):

Wenn TRUE, dann werden die Spektren vor und nach der Datenvorbehandlung im Plots Fenster angezeigt.

savedata.TF = TRUE (boolean):

Dieser Wert sollte nicht auf FALSE gesetzt werden, ansonsten erscheint diese Datenvorbehandlung nicht in der Auswertung. Wird benötigt für die Vorhersagefunktion predict.ex().

Beispiel

```
scales <- rep(1, 401)
scales[330:401] <- 0.3
scales[110:175] <- 2
Gasoline <- scaleX(data = Gasoline, scales = scales)
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. Für die Skalierung wird ein Skalierungsvektor scales benötigt. Dieser muss genauso lang sein, wie die Anzahl an Variablen im Spektrum. Das Ergebnis dieses Beispiels lässt sich in Abbildung 6 sehen. Die linke große Doppelbande wurde stärker gewichtet und die ganz rechte verrauschte Bande wurde weniger stark gewichtet.

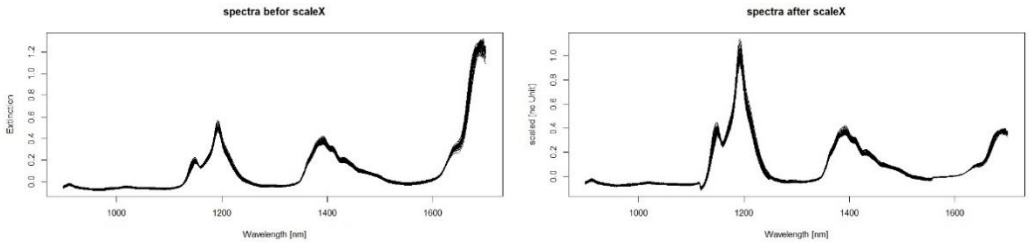


Abbildung 6: Spektren bevor (links) und nach (rechts) scaleX()

1.4.3 standardizeX()

standardizeX() ist eine Funktion, um die X-Daten mithilfe der Standardabweichung zu standardisieren. Für nähere Informationen siehe 2.3.1 Zentrierung, Standardisierung & Skalierung.

Parameter

data (PLSR.Kronsbein Datensatz):

Datensatz, welcher durch eine Einlesefunktion erzeugt wurde, übergeben.

printplots.TF = FALSE (boolean):

Wenn TRUE, dann werden die Spektren vor und nach der Datenvorbehandlung im Plots Fenster angezeigt.

savedata.TF = TRUE (boolean):

Dieser Wert sollte nicht auf FALSE gesetzt werden, ansonsten erscheint diese Datenvorbehandlung nicht in der Auswertung. Wird benötigt für die Vorhersagefunktion predict.ex().

referencedata = NULL (numeric vector):

Dieser Parameter wird vom Nutzer nicht benötigt. Er dient in der Vorhersagefunktion dafür, die originalen Standardabweichungen zu übergeben.

Beispiel

```
Gasoline <- standardizeX(data = Gasoline)
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. Zusätzlich wurde Gasoline noch zentriert über centerX(). Es ist sinnvoller, zentrierte Daten zu standardisieren, als nicht zentrierte. Das Ergebnis der Funktion ist in Abbildung 7 zu sehen. Nach der Standardisierung ist im Spektrum nicht mehr viel zu erkennen. In der Regel ergibt es auch keinen Sinn, Spektren zu standardisieren, da somit Bereiche bestehend aus Rauschen, genauso stark wie Banden gewichtet werden.

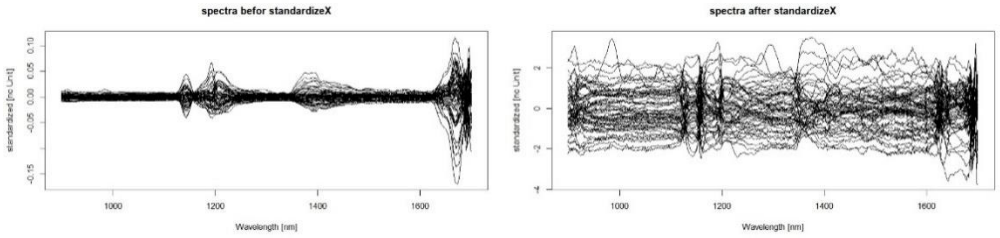


Abbildung 7: Spektren bevor (links) und nach (rechts) standardizeX()

1.4.4 centerY()

centerY() ist eine Funktion, um die Y-Daten zu zentrieren. Für nähere Informationen siehe 2.3.1 Zentrierung, Standardisierung & Skalierung. Falls die Daten schon innerhalb einer Einlesefunktion durch den Parameter centerY = TRUE zentriert wurden, wird dies rückgängig gemacht. Der Unterschied zu centerY = TRUE besteht darin, dass die Zentrierung durch centerY() während der Auswertung nicht Rückgängig gemacht wird.

Parameter

data (PLSR.Kronsbein Datensatz):

Datensatz, welcher durch eine Einlesefunktion erzeugt wurde, übergeben.

savedata.TF = TRUE (boolean):

Dieser Wert sollte nicht auf FALSE gesetzt werden, ansonsten erscheint diese Datenvorbehandlung nicht in der Auswertung. Wird benötigt für die Vorhersagefunktion predict.ex().

referencedata = NULL (numeric vector):

Dieser Parameter wird vom Nutzer nicht benötigt. Er dient in der Vorhersagefunktion dafür, die originalen Mittelwerte zu übergeben.

silent = FALSE (boolean):

Wenn dieser Wert auf TRUE gesetzt wird, dann wird nichts in der Konsole ausgegeben. Dies ist hilfreich, wenn die Funktion als interne Funktion aufgerufen wird.

Beispiel

```
Gasoline <- centerY(data = Gasoline)
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. Ein Ausschnitt der zentrierten Y-Werte ist in Abbildung 8 zu sehen.

	A	B	C
1	sample	octane	octane (centered)
2	1	85.3	-1.8775
3	2	85.25	-1.9275
4	3	88.45	1.2725
5	4	83.4	-3.7775
6	5	87.9	0.7225
7	6	85.5	-1.6775
8	7	88.9	1.7225
9	8	88.3	1.1225
10	9	88.7	1.5225
11	10	88.45	1.2725
12	11	88.75	1.5725
13	12	88.25	1.0725

Abbildung 8: Die Y-Werte der ersten 12 Proben vor (Spalte B) und nach (Spalte C) der Zentrierung.

1.4.5 scaleY()

scaleY() ist eine Funktion, um die Y-Daten zu skalieren/gewichten. Für nähere Informationen siehe 2.3.1 Zentrierung, Standardisierung & Skalierung.

Parameter

data (PLSR.Kronsbein Datensatz):

Datensatz, welcher durch eine Einlesefunktion erzeugt wurde, übergeben.

scales = NULL (numeric vector):

Hier werden die Skalierungen/Gewichtungen für die Y-Variablen als Vektor übergeben.

savedata.TF = TRUE (boolean):

Dieser Wert sollte nicht auf FALSE gesetzt werden, ansonsten erscheint diese Datenvorbehandlung nicht in der Auswertung. Wird benötigt für die Vorhersagefunktion predict.ex().

silent = FALSE (boolean):

Wenn dieser Wert auf TRUE gesetzt wird, dann wird nichts in der Konsole ausgegeben. Dies ist hilfreich, wenn die Funktion als interne Funktion aufgerufen wird.

Beispiel

```
Beispiel <- scaleY(data = Beispiel, scales = c(1, 0.1, 100))
```

Zusätzliche Informationen:

Als data wird Beispiel übergeben. Beispiel ist ein fiktiver Datensatz mit 3 Y-Variablen, welcher durch eine Einlesefunktion erstellt wurde. Diese Funktion ist nicht sinnvoll für Gasoline, da Gasoline nur eine Y-Variable besitzt. Ein mögliches Beispiel für die Skalierung der Y-Werte ist in Abbildung 9 zu sehen. Die zweite Variable wurde dabei schwächer gewichtet und die dritte deutlich stärker.

	A	B	C	D	E	F	G
1	**sample**	**Y1**	**Y2**	**Y3**	**Y1(scaled)**	**Y2(scaled)**	**Y3(scaled)**
2	1	79	-24	9	79	-2.4	900
3	2	39	35	7	39	3.5	700
4	3	25	-17	7	25	-1.7	700
5	4	16	25	7	16	2.5	700
6	5	16	-27	2	16	-2.7	200
7	6	8	-9	9	8	-0.9	900
8	7	25	-47	8	25	-4.7	800
9	8	5	14	3	5	1.4	300
10	9	74	36	10	74	3.6	1000
11	10	54	6	7	54	0.6	700
12	11	75	45	10	75	4.5	1000
13	12	6	-16	2	6	-1.6	200

Abbildung 9: Die Y-Werte von Beispiel vor (Spalte B bis D) und nach (Spalte E bis G) nach der Skalierung.

1.4.6 standardizeY()

Funktion, um die Y-Daten mithilfe der Standardabweichung zu standardisieren. Für nähere Informationen siehe 2.3.1 Zentrierung, Standardisierung & Skalierung.

Parameter

data (PLSR.Kronsbein Datensatz):

Datensatz, welcher durch eine Einlesefunktion erzeugt wurde, übergeben.

savedata.TF = TRUE (boolean):

Dieser Wert sollte nicht auf FALSE gesetzt werden, ansonsten erscheint diese Datenvorbehandlung nicht in der Auswertung. Wird benötigt für die Vorhersagefunktion predict.ex().

referencedata = NULL (numeric vector):

Dieser Parameter wird vom Nutzer nicht benötigt. Er dient in der Vorhersagefunktion dafür die originalen Standardabweichungen zu übergeben.

silent = FALSE (boolean):

Wenn dieser Wert auf TRUE gesetzt wird, dann wird nichts in der Konsole ausgegeben. Dies ist hilfreich, wenn die Funktion als interne Funktion aufgerufen wird.

Beispiel

```
Beispiel <- centerY(data = Beispiel)
Beispiel <- standardizeY(data = Beispiel)
```

Zusätzliche Informationen:

Hier wird zuerst die Zentrierung und dann die Standardisierung durchgeführt. Eine Standardisierung ist sinnvoller mit vorangestellter Zentrierung. Als data wird Beispiel übergeben. Beispiel ist ein fiktiver Datensatz mit 3 Y-Variablen, welcher durch eine Einlesefunktion erstellt wurde. Diese Funktion ist nicht sinnvoll für Gasoline, da Gasoline nur eine Y-Variable besitzt. Ein mögliches Beispiel für die Zentrierung und Standardisierung der Y-Werte ist in Abbildung 10 zu sehen.

	A	B	C	D	E	F	G	H	I	J
1	sample	Y1	Y2	Y3	Y1(centered)	Y2(centered)	Y3(centered)	Y1(centered&standardized)	Y2(centered&standardized)	Y3(centered&standardized)
2	1	26	5	4	-13.000	-1.083	0.250	-0.525	-0.032	0.081
3	2	46	50	0	7.000	43.917	-3.750	0.283	1.315	-1.218
4	3	36	18	0	-3.000	11.917	-3.750	-0.121	0.357	-1.218
5	4	20	33	8	-19.000	26.917	4.250	-0.768	0.806	1.381
6	5	23	-49	1	-16.000	-55.083	-2.750	-0.646	-1.649	-0.893
7	6	85	-45	7	46.000	-51.083	3.250	1.859	-1.529	1.056
8	7	59	6	8	20.000	-0.083	4.250	0.808	-0.002	1.381
9	8	63	49	3	24.000	42.917	-0.750	0.970	1.285	-0.244
10	9	17	25	0	-22.000	18.917	-3.750	-0.889	0.566	-1.218
11	10	28	-34	5	-11.000	-40.083	1.250	-0.444	-1.200	0.406
12	11	0	11	3	-39.000	4.917	-0.750	-1.576	0.147	-0.244
13	12	65	4	6	26.000	-2.083	2.250	1.051	-0.062	0.731

Abbildung 10: Der Y-Werte von Beispiel ohne Datenvorbehandlung (Spalte B bis D), nach der Zentrierung (Spalte E bis G) und nach der Zentrierung und Standardisierung (Spalte H bis J)

1.4.7 cutspectrum()

Diese Funktion dient dazu, ein Spektrum vor der weiteren Verarbeitung zurechtzuschneiden. Dies kann hilfreich sein, wenn Bereiche gestört oder uninteressant sind. Es ist möglich dies über einen interaktiven Modus durchzuführen oder indem die Schnittpunkte als Matrix durch cuts.n oder cuts.w übergeben werden. Es wird immer der ausgewählte Bereich aus dem Spektrum entfernt.

Parameter

data (PLSR.Kronsbein Datensatz):

Datensatz, welcher durch eine Einlesefunktion erzeugt wurde, übergeben.

printplots.TF = FALSE (boolean):

Wenn TRUE, dann werden die Spektren vor und nach der Datenvorbehandlung im Plots Fenster angezeigt.

cuts.n = NULL (numeric matrix):

Hier die gewünschten Schnitte als Matrix übergeben. Die Matrix muss zwei Spalten besitzen, die erste für den Anfang des Schnitts und die zweite für das Ende des Schnitts. In jeder Reihe kann ein Schnitt eingetragen werden. Als Schnittpunkte bitte die Spalte der X-Datenmatrix, in der die entsprechende Variable steht, als numerischen Wert übergeben. Für die erste und letzte Variable kann einfach 0 eingetragen werden. Bitte nur cuts.n oder cuts.w nutzen.

cuts.w = NULL (numeric matrix):

Hier die gewünschten Schnitte als Matrix übergeben. Die Matrix muss zwei Spalten besitzen, die erste für den Anfang des Schnitts und die zweite für das Ende des Schnitts. In jeder Reihe kann ein Schnitt eingetragen werden. Als Schnittpunkte die Wellenlänge oder Wellenzahl, der entsprechenden Variable eintragen, je nachdem welche Einheit genutzt wird. Für die erste und letzte Variable kann einfach 0 eingetragen werden. Bitte nur cuts.n oder cuts.w nutzen.

interactive = FALSE (boolean):

Wenn TRUE, dann wird der interaktive Modus genutzt, wobei der Nutzer die Schnitte setzen kann und sich das Ergebnis anschauen kann.

plottype = „p“ (character):

Hier kann die Art der Plots verändert werden. „p“ für Punte, „l“ für Linien und „b“ für Beide.

plotpch = 20 (numeric):

Hier kann die Größe der Schrift im Plot verändert werden.

plotcex = 0.3 (numeric):

Hier kann die Größe der Datenpunkte im Plot verändert werden.

savedata.TF = TRUE (boolean):

Dieser Wert sollte nicht auf FALSE gesetzt werden, ansonsten erscheint diese Datenvorbehandlung nicht in der Auswertung. Wird benötigt für die Vorhersagefunktion predict.ex().

Beispiele

Variante 1: interaktiv

```
Gasoline <- cutspectrum(data = Gasoline, interactive = TRUE)
```

Variante 2: vorher definierte Schnittpunkte über die Wellenlänge

```
cuts.w <- matrix(data = c(0,1100,1250,1340,1550,0), ncol = 2, byrow = TRUE)
Gasoline <- cutspectrum(data = Gasoline, cuts.w = cuts.w)
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen.
In Variante 1 wird der interaktive Modus gestartet. Alles Weitere wird während der Programmausführung erklärt. Die Eingaben werden in der Konsole durch die Eingabe der entsprechenden Optionen getätigt.
In Variante 2 werden die Schnitte durch die vorher erstellte Matrix cuts.w getätigt. Die cuts.w Matrix ist in Abbildung 11 zu sehen. In jeder Zeile wird ein Schnitt getätigt. 0 steht entweder für den ersten oder letzten Wert, je nachdem welcher sinnvoll ist. Das Ergebnis der Funktion aus Variante 2 ist in Abbildung 12 zu sehen.

	A	B	C
1		**von**	**bis**
2	**Schnitt 1**	0	1100
3	**Schnitt 2**	1250	1340
4	**Schnitt 3**	1550	0

Abbildung 11: cuts.w Matrix für cutspectrum()

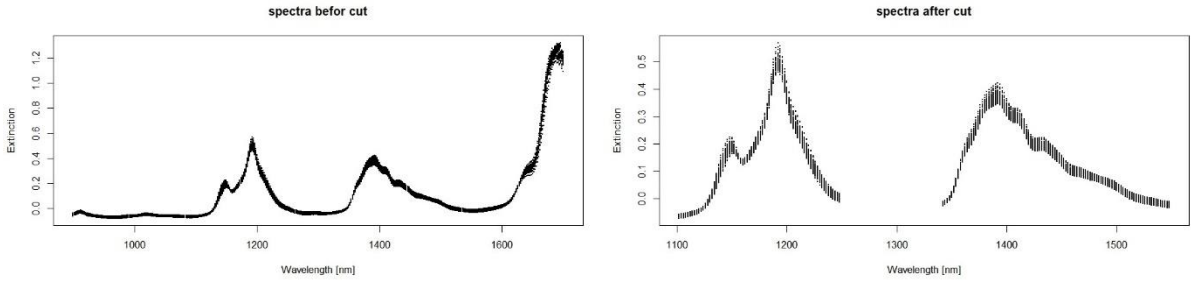


Abbildung 12: Spektren bevor (links) und nach (rechts) cutspectrum()

1.4.8 msc.adapted()

Diese Funktion dient dazu eine MSC-Korrektur der Spektren durchzuführen. Für nähere Informationen siehe 2.3.3 MSC.

Parameter

data (PLSR.Kronsbein Datensatz):

Datensatz, welcher durch eine Einlesefunktion erzeugt wurde, übergeben.

printplots.TF = FALSE (boolean):

Wenn TRUE, dann werden die Spektren vor und nach der Datenvorbehandlung im Plots Fenster angezeigt.

savedata.TF = TRUE (boolean):

Dieser Wert sollte nicht auf FALSE gesetzt werden, ansonsten erscheint diese Datenvorbehandlung nicht in der Auswertung. Wird benötigt für die Vorhersagefunktion predict.ex().

own.reference = NULL (numeric vector):

Hier kann ein eigenes Referenzspektrum übergeben werden, ansonsten wird das Mittelwertspektrum genutzt. Normalerweise wird das Mittelwertspektrum genutzt, da das Idealspektrum nicht bekannt ist.

Beispiel

```
Gasoline <- msc.adapted(data = Gasoline)
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. Das Ergebnis der Funktion ist in Abbildung 13 zu sehen.

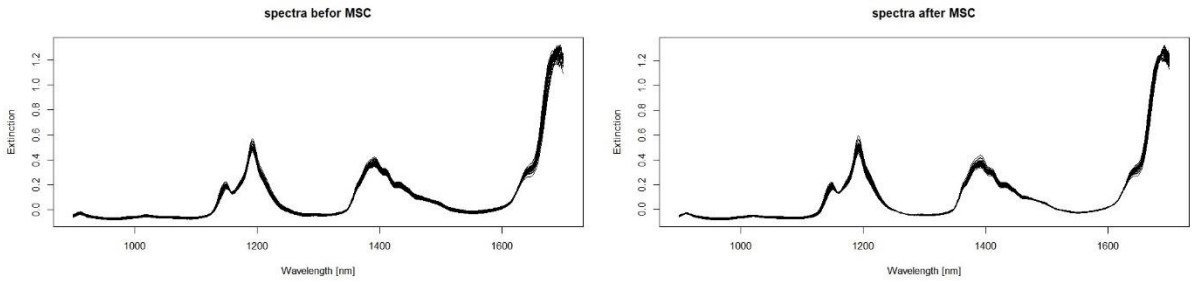


Abbildung 13: Spektren bevor (links) und nach (rechts) msc.adapted()

1.4.9 polynomial_baseline_correction()

Diese Funktion dient dazu eine Basislinienkorrektur der Spektren durchzuführen. Für nähere Informationen siehe 2.3.6 Basislinienkorrektur. Ebenfalls ist ein interaktiver Modus verfügbar, bei dem verschiedene Modelle und thresholds getestet werden können, bis man eine passende Einstellung gefunden hat.

Parameter

data (PLSR.Kronsbein Datensatz):

Datensatz, welcher durch eine der Einlesefunktionen erzeugt wurde, übergeben.

model = 0 (numeric):

Hier kann der Grad (0 bis 10) der Funktion des Modells der Basislinie angegeben werden.

threshold = 0.001 (numeric):

Der threshold bestimmt die Genauigkeit der Anpassung der Basislinie. Je niedriger der threshold, desto besser die Anpassung, aber desto höher die Berechnungszeit.

interactive = FALSE (boolean):

Wenn TRUE, wird der interaktive Modus genutzt, bei dem Modell und threshold angepasst werden können.

savedata.TF = TRUE (boolean):

Dieser Wert sollte nicht auf FALSE gesetzt werden, ansonsten erscheint diese Datenvorbehandlung nicht in der Auswertung. Wird benötigt für die Vorhersagefunktion predict.ex().

printplots.TF = FALSE (boolean):

Wenn TRUE, dann werden die Spektren vor und nach der Datenvorbehandlung im Plots Fenster angezeigt.

multithread = FALSE (boolean):

Wenn TRUE, dann werden alle verfügbaren Threads des Computers für die Berechnungen genutzt. Dies ist bei großen Datensätzen bedeutend schneller. Bei sehr kleinen Datensätzen wird die Berechnung aufgrund von Kopiervorgängen jedoch etwas verlängert. Der Multithreadmodus ist sehr RAM-intensiv.

Beispiel

Variante 1:

```
Gasoline <- polynomial_baseline_correction(data = Gasoline, model = 1)
```

Variante 2: mit interaktivem Modus

```
Gasoline <- polynomial_baseline_correction(data = Gasoline, model = 1, interactive = TRUE)
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. In Variante 1 wurde die Basislinienkorrektur mit einem linearen Modell durchgeführt. In Variante 2 wurde zusätzlich der interaktive Modus aktiviert, sodass es möglich ist, wenn einem das Ergebnis nicht gefällt, das Modell und den Threshold zu ändern. Das Ergebnis der Funktion aus Variante 1 ist in Abbildung 14 zu sehen.

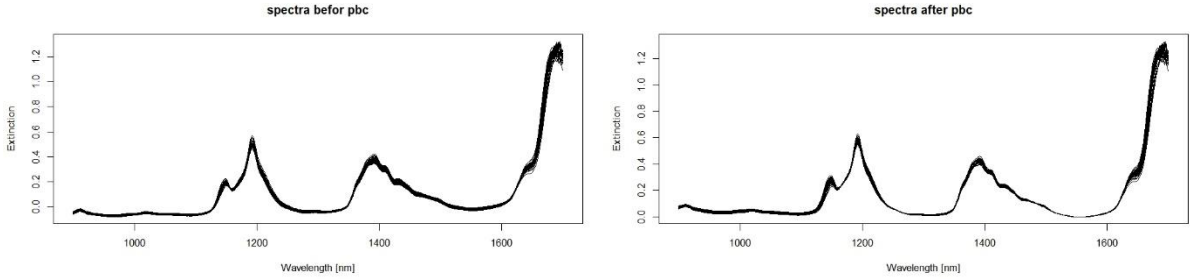


Abbildung 14: Spektren bevor (links) und nach (rechts) polynomial_baseline_correction()

1.4.10 smoothspectrums.polynomial()

Mit dieser Funktion werden die Spektren mithilfe des Savitzky-Golay-Verfahrens geglättet. Für nähere Informationen siehe 2.3.4 Glättung (Savitzky Golay).

Parameter

data (PLSR.Kronsbein Datensatz):

Datensatz, welcher durch eine der Einlesefunktionen erzeugt wurde, übergeben.

degreeofsmoothing = 5 (numeric):

Dieser Parameter bestimmt die Stärke der Glättung, also über wie viele Datenpunkte die Glättung erfolgt. Je größer degreeofsmoothing, desto stärker die Glättung. degreeofsmoothing muss ein ungerader positiver ganzzahliger Wert sein.

model = 2 (numeric):

Hier kann der Grad (positiver ganzzahliger Wert) der Funktion des Glättungsmodells angegeben werden.

printplots.TF = FALSE (boolean):

Wenn TRUE, dann werden die Spektren vor und nach der Datenvorbehandlung im Plots Fenster angezeigt.

savedata.TF = TRUE (boolean):

Dieser Wert sollte nicht auf FALSE gesetzt werden, ansonsten erscheint diese Datenvorbehandlung nicht in der Auswertung. Wird benötigt für die Vorhersagefunktion predict.ex().

Beispiel

```
Gasoline <- smoothspectrums.polynomial(data = Gasoline, degreeofsmoothing = 15, model = 4)
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. Die Glättung erfolgt über 15 Datenpunkte mit einem Modell vierten Grades. Das Ergebnis der Funktion ist in Abbildung 15 zu sehen.

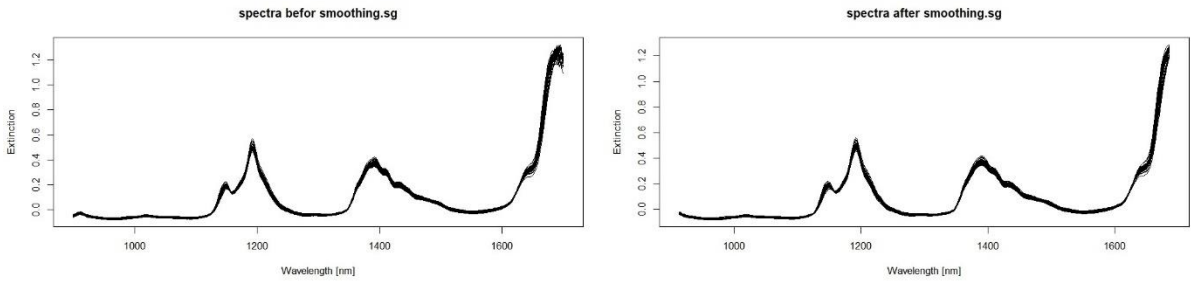


Abbildung 15: Spektren bevor (links) und nach (rechts) smoothspectrums.polynomial()

1.4.11 derivatespectrums()

Mit dieser Funktion werden die Spektren mithilfe des Savitzky-Golay-Verfahrens abgeleitet. Für nähere Informationen siehe 2.3.5 Ableitung (Savitzky-Golay). Dabei erfolgt eine Glättung nach Savitzky-Golay und dann die Ableitung der Glättungsfunktion.

Parameter

data (PLSR.Kronsbein Datensatz):

Datensatz, welcher durch eine der Einlesefunktionen erzeugt wurde, übergeben.

degreeofsmoothing = 5 (numeric):

Dieser Parameter bestimmt die Stärke der Glättung, also über wie viele Datenpunkte die Glättung erfolgt. Je größer degreeofsmoothing, desto stärker die Glättung. degreeofsmoothing muss ein ungerader positiver ganzzahliger Wert sein.

model = 2 (numeric):

Hier kann der Grad (positiver ganzzahliger Wert) der Funktion des Glättungsmodells angegeben werden.

derivation = 1 (numeric):

Hier kann der Grad der Ableitung bestimmt werden. Der Grad der Ableitung darf nicht größer sein als der Grad des Modells und muss größer als 0 sein.

repeatfirstderivation.TF = FALSE (boolean):

Wenn TRUE, dann wird die erste Ableitung wiederholt, entsprechend des Wertes von derivation. Dies bewirkt eine immer wiederkehrende Glättung des Rauschens aufgrund der Ableitung.

printplots.TF = FALSE (boolean):

Wenn TRUE, dann werden die Spektren vor und nach der Datenvorbehandlung im Plots Fenster angezeigt.

savedata.TF = TRUE (boolean):

Dieser Wert sollte nicht auf FALSE gesetzt werden, ansonsten erscheint diese Datenvorbehandlung nicht in der Auswertung. Wird benötigt für die Vorhersagefunktion predict.ex().

Beispiel

```
Gasoline <- derivatespectrums(data = Gasoline, degreeofsmoothing = 15, model = 4, derivation = 2,
    repeatfirstderivation.TF = TRUE)
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. Die Ableitung erfolgt durch zweimaliges durchführen der 1. Ableitung. Die erste Ableitung wird von einem Modell vierten

Grades gebildet, welches über 15 Datenpunkte angepasst wurde. Das Ergebnis der Funktion ist in Abbildung 16 zu sehen.

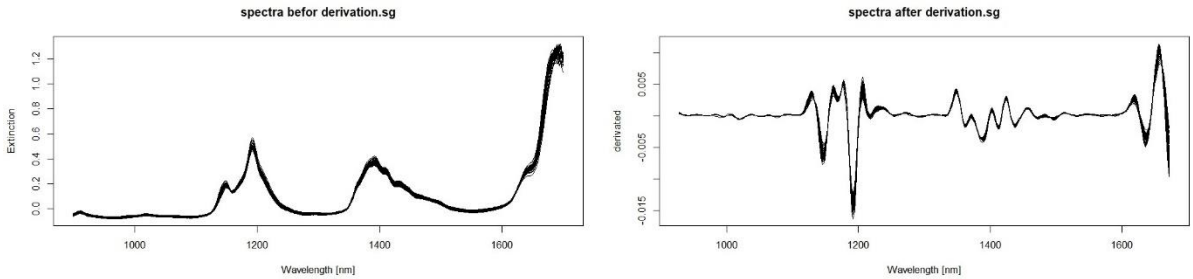


Abbildung 16: Spektren bevor (links) und nach (rechts) derivatespectrums()

1.4.12 smoothspectrums.mean()

Einfache Version von smoothspectrums.polynomial() bei der nur über den Mittelwert geglättet wird.

Parameter

data (PLSR.Kronsbein Datensatz):

Datensatz, welcher durch eine der Einlesefunktionen erzeugt wurde, übergeben.

degreeofsmoothing = 3 (numeric):

Dieser Parameter bestimmt die Stärke der Glättung, also über wie viele Datenpunkte die Glättung erfolgen soll. Je größer degreeofsmoothing, desto stärker die Glättung. Bei degreeofsmoothing muss es sich um einen ungeraden positiven ganzzahligen Wert handeln.

printplots.TF = FALSE (boolean):

Wenn TRUE, dann werden die Spektren vor und nach der Datenvorbehandlung im Plots Fenster angezeigt.

savedata.TF = TRUE (boolean):

Dieser Wert sollte nicht auf FALSE gesetzt werden, ansonsten erscheint diese Datenvorbehandlung nicht in der Auswertung. Wird benötigt für die Vorhersagefunktion predict.ex().

Beispiel

```
Gasoline <- smoothspectrums.mean(data = Gasoline, degreeofsmoothing = 7)
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. Die Glättung mittels Mittelwert erfolgt über 7 Datenpunkte. Das Ergebnis der Funktion ist in Abbildung 17 zu sehen.

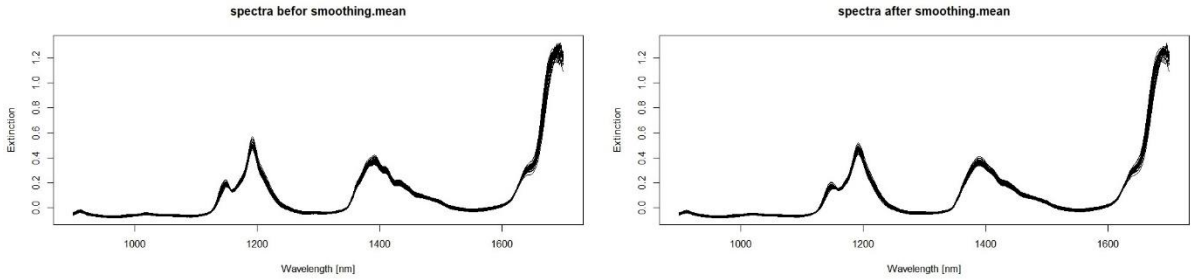


Abbildung 17: Spektren bevor (links) und nach (rechts) smoothspectrums.mean()

1.4.13 SNV()

Diese Funktion dient dazu, eine SNV-Korrektur der Spektren durchzuführen. Für nähere Informationen siehe 2.3.2 SNV.

Parameter

data (PLSR.Kronsbein Datensatz):

Datensatz, welcher durch eine der Einlesefunktionen erzeugt wurde, übergeben.

printplots.TF = FALSE (boolean):

Wenn TRUE, dann werden die Spektren vor und nach der Datenvorbehandlung im Plots Fenster angezeigt.

savedata.TF = TRUE (boolean):

Dieser Wert sollte nicht auf FALSE gesetzt werden, ansonsten erscheint diese Datenvorbehandlung nicht in der Auswertung. Wird benötigt für die Vorhersagefunktion predict.ex().

Beispiel

```
Gasoline <- SNV(data = Gasoline)
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. Es wird eine SNV-Korrektur durchgeführt. Das Ergebnis der Funktion ist in Abbildung 18 zu sehen.

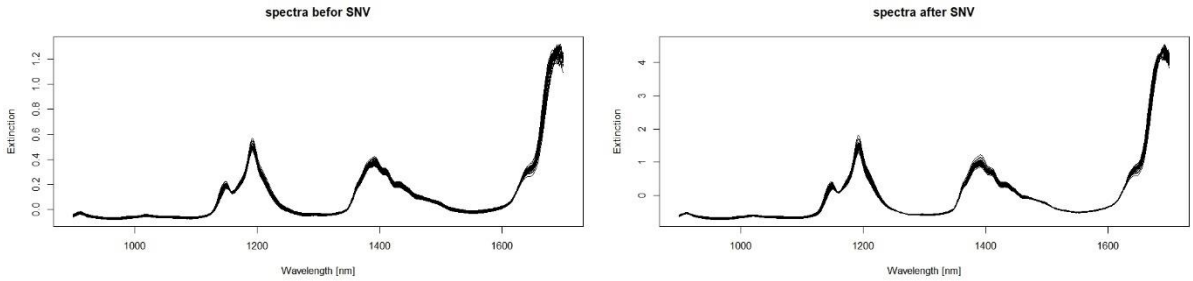


Abbildung 18: Spektren bevor (links) und nach (rechts) SNV()

1.4.14 Wavenumber_to_Wavelength()

Funktion zur Umrechnung der Spektren von der Wellenzahl in die Wellenlänge. Wichtig hierfür ist, dass die Einheit für die X-Achse (Einstellbar durch UnitspecX in einer der

Einlesefunktionen) auf "Wavenumber [cm^-1]" gesetzt ist. Für nähere Informationen siehe 2.1.1.2 Physikalische Zusammenhänge.

Parameter

data (PLSR.Kronsbein Datensatz):

Datensatz, welcher durch eine der Einlesefunktionen erzeugt wurde, übergeben.

printplots.TF = FALSE (boolean):

Wenn TRUE, dann werden die Spektren vor und nach der Datenvorbehandlung im Plots Fenster angezeigt.

savedata.TF = TRUE (boolean):

Dieser Wert sollte nicht auf FALSE gesetzt werden, ansonsten erscheint diese Datenvorbehandlung nicht in der Auswertung. Wird benötigt für die Vorhersagefunktion predict.ex().

Beispiel

```
Gasoline <- Wavenumber_to_Wavelength(data = Gasoline)
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. Zusätzlich wurde die Funktion Wavelength_to_Wavenumber() angewendet, damit die Funktion Wavenumber_to_Wavelength() durchführbar ist. Die Funktion bewirkt die Umrechnung der Wellenzahl in die Wellenlänge. Das Ergebnis der Funktion ist in Abbildung 19 zu sehen.

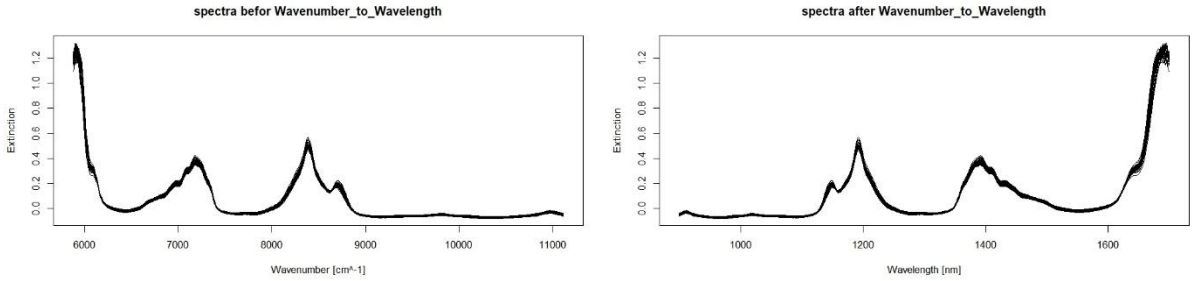


Abbildung 19: Spektren bevor (links) und nach (rechts) Wavenumber_to_Wavelength()

1.4.15 Wavelength_to_Wavenumber()

Funktion zur Umrechnung der Spektren von der Wellenlänge in die Wellenzahl. Wichtig hierfür ist, dass die Einheit für die X-Achse (Einstellbar durch UnitspecX in einer der

Einlesefunktionen) auf "Wavelength [nm]" gesetzt ist. Für nähere Informationen siehe 2.1.1.2 Physikalische Zusammenhänge.

Parameter

data (PLSR.Kronsbein Datensatz):

Datensatz, welcher durch eine der Einlesefunktionen erzeugt wurde, übergeben.

printplots.TF = FALSE (boolean):

Wenn TRUE, dann werden die Spektren vor und nach der Datenvorbehandlung im Plots Fenster angezeigt.

savedata.TF = TRUE (boolean):

Dieser Wert sollte nicht auf FALSE gesetzt werden, ansonsten erscheint diese Datenvorbehandlung nicht in der Auswertung. Wird benötigt für die Vorhersagefunktion predict.ex().

Beispiel

```
Gasoline <- Wavelength_to_Wavenumber(data = Gasoline)
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. Die Wellenlänge wird in die Wellenzahl umgerechnet. Das Ergebnis der Funktion ist in Abbildung 20 zu sehen.

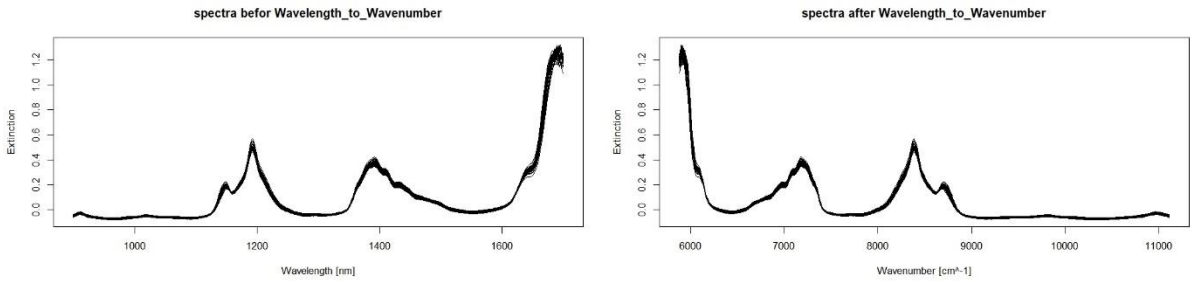


Abbildung 20: Spektren bevor (links) und nach (rechts) Wavelength_to_Wavenumber()

1.4.16 Transmission_to_Extinction()

Funktion zur Umrechnung der Spektren von der Transmission in die Extinktion. Wichtig hierfür ist, dass die Einheit für die Y-Achse (Einstellbar durch UnitspecY in einer der

Einlesefunktionen) auf "Transmission" gesetzt ist. Für nähere Informationen siehe 2.1.1.3 Transmissions Spektroskopie.

Parameter

data (PLSR.Kronsbein Datensatz):

Datensatz, welcher durch eine der Einlesefunktionen erzeugt wurde, übergeben.

printplots.TF = FALSE (boolean):

Wenn TRUE, dann werden die Spektren vor und nach der Datenvorbehandlung im Plots Fenster angezeigt.

savedata.TF = TRUE (boolean):

Dieser Wert sollte nicht auf FALSE gesetzt werden, ansonsten erscheint diese Datenvorbehandlung nicht in der Auswertung. Wird benötigt für die Vorhersagefunktion predict.ex().

Beispiel

```
Gasoline <- Transmission_to_Extinction(data = Gasoline)
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. Zusätzlich wurde die Funktion Extinction_to_Transmission() angewendet, damit die Funktion Transmission_to_Extinction() durchführbar ist. Die ausgeführte Funktion bewirkt die Umrechnung von der Transmission in die Extinktion. Das Ergebnis der Funktion ist in Abbildung 21 zu sehen.

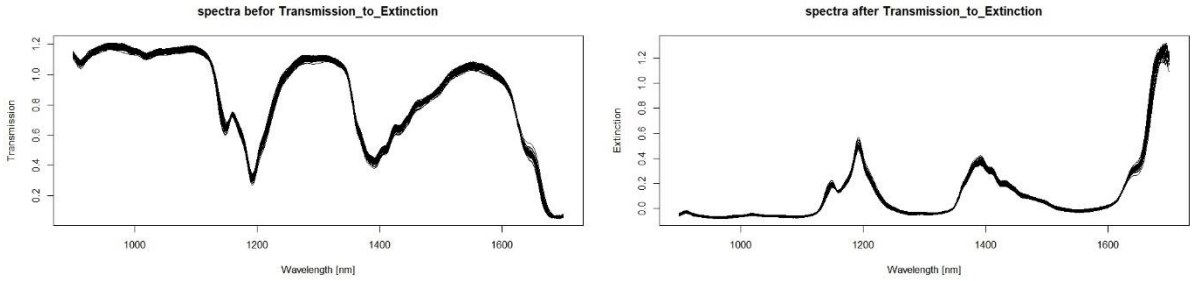


Abbildung 21: Spektren bevor (links) und nach (rechts) Transmission_to_Extinction()

1.4.17 Extinction_to_Transmission()

Funktion zur Umrechnung der Spektren von der Extinktion in die Transmission. Wichtig hierfür ist, dass die Einheit für die Y-Achse (Einstellbar durch UnitspecY in einer der

Einlesefunktionen) auf "Extinction" gesetzt ist. Für nähere Informationen siehe 2.1.1.3 Transmissions Spektroskopie.

Parameter

data (PLSR.Kronsbein Datensatz):

Datensatz, welcher durch eine der Einlesefunktionen erzeugt wurde, übergeben.

printplots.TF = FALSE (boolean):

Wenn TRUE, dann werden die Spektren vor und nach der Datenvorbehandlung im Plots Fenster angezeigt.

savedata.TF = TRUE (boolean):

Dieser Wert sollte nicht auf FALSE gesetzt werden, ansonsten erscheint diese Datenvorbehandlung nicht in der Auswertung. Wird benötigt für die Vorhersagefunktion predict.ex().

Beispiel

```
Gasoline <- Extinction_to_Transmission(data = Gasoline)
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. Die ausgeführte Funktion bewirkt die Umrechnung von der Extinktion in die Transmission. Das Ergebnis der Funktion ist in Abbildung 22 zu sehen.

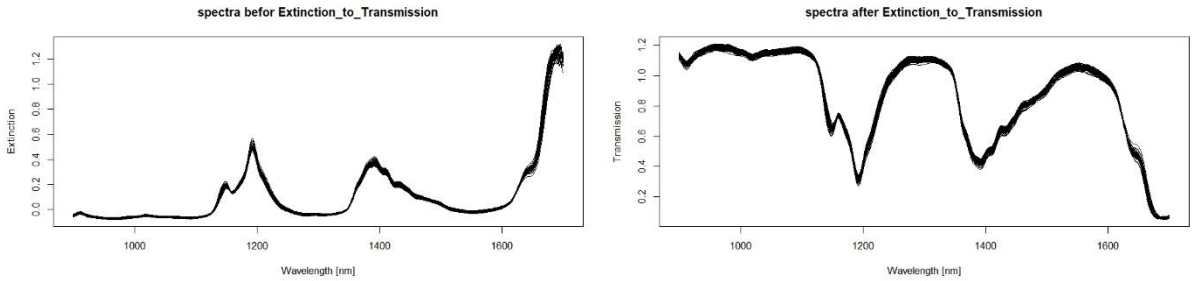


Abbildung 22: Spektren bevor (links) und nach (rechts) Extinction_to_Transmission()

1.4.18 generate.split_and_splitdata.externalvalidation()

Diese Funktion dient dazu, den Datensatz in einen Test und einen Trainingsdatensatz aufzusplitten. Der Trainingsdatensatz dient dazu, ein Modell zu erstellen. Der Testdatensatz kann später genutzt werden, um eine externe Validierung durchzuführen. Wird kein neuer Testdatensatz angegeben, wird dann automatisch, der hier erstellte genutzt.
Diese Funktion sollte als erste Datenvorbereitungsfunktion durchgeführt werden.

Parameter

data = NULL (PLSR.Kronsbein Datensatz):

Datensatz, welcher durch eine Einlesefunktion erzeugt wurde, übergeben.

trainvalues.ratio = 0.75 (numeric):

Der Anteil der Daten der als Trainingsdatensatz dienen soll. Der Rest wird zum Testdatensatz. Der Wert muss im Bereich von 0 bis 1 liegen.

whichY = 1 (numeric):

Welche der Y-Variablen soll im Trainingsdatensatz möglichst gleichmäßig vertreten sein. Dies ist nur bei einem PLS2 (mehrere Y-Variablen) Datensatz relevant. Es ist nicht möglich mehrere Variablen auf einmal auszuwählen.

repetitions = data$data.info$read.repetitions() (numeric):

Hier können die Anzahl an Wiederholungen gleicher Datenpunkte angegeben werden. Dies ist aber nicht notwendig, da die Funktion diese automatisch aus dem angegebenen Wert in der Einlesefunktion ausliest.

savedata.TF = TRUE (boolean):

Dieser Wert sollte nicht auf FALSE gesetzt werden, ansonsten erscheint diese Datenvorbehandlung nicht in der Auswertung. Wird benötigt für die Vorhersagefunktion predict.ex().

Beispiel

```
Gasoline <- generate.split_and_splitdata.externalvalidation(data = Gasoline, trainvalues.ratio = 0.75)
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. Später kann dann plsr.ex() mit validation = „ex“ ausgeführt werden, um eine externe Validierung mit dem abgetrennten Testdatensatz durchzuführen.

1.4.19 variableselection()

Diese Funktion dient dazu, mithilfe verschiedener Methoden die Variablen auszuwählen, welche die vorherzusagenden Variablen am besten beschreiben (siehe 2.3.7 Variablenselektion).
Die wichtigsten Daten werden während der Durchführung mitgeschrieben unter history. Es wird mitgeschrieben, welche Variable hinzugefügt oder geändert wurde. Ebenfalls wird mitgeschrieben, welchen RMSEP ein PLS-Modell bei diesem Variablensubset besitzen würde, bestimmt durch eine gewählte Validierungsmethode. Zusätzlich werden bei den Procrustes Methoden die Procrustes Distanz und der Goodness of Fit Wert mitgeschrieben.

Parameter

data = NULL (PLSR.Kronsbein Datensatz):

Datensatz, welcher durch eine der Einlesefunktionen erzeugt wurde, übergeben.

selectionalgorithm = „CARS“ (character):

Durch diesen Parameter wird bestimmt welcher Selektionsalgorithmus genutzt wird.

- „CARS“ → Variablenselektion durch den CARS-Algorithmus (siehe 2.3.7.3 CARS); ncomp sollte gesetzt werden
- „Procrustes“ → Variablenselektion durch die Procrustes-Analyse (siehe 2.3.7.1 Procrustes Analysis); centerX sollte TRUE sein
- „PCA.procrustes“ → Variablenselektion durch die PCA-Procrustes-Analyse (siehe 2.3.7.2 PCA - Procrustes Analyse); ncomp.PCA.PA muss gesetzt werden; centerX sollte TRUE sein

Tipp: PCA.procrustes ist der schnellste Algorithmus. Der Procrustes Algorithmus ist bei über 600 Variablen nahezu nicht mehr zu gebrauchen, da er zu langsam ist. Um die Algorithmen zu beschleunigen, ist es sinnvoll, die Anzahl der Variablen vorher zu verringern (siehe 1.4.21 reduce_variables_spectra() oder 1.4.7 cutspectrum())

direction = „backwards“ (character):

Durch diesen Parameter wird bestimmt, in welcher Richtung die Variablenselektion erfolgen soll.

- „backwards“ → Rückwärtseliminierung wird durchgeführt, die Variablen werden nach und nach einzeln entfernt.
- „forwards“ → Vorwärtsselektion wird durchgeführt, die Variablen werden nach und nach einzeln hinzugefügt.

validation = „LOO“ (character):

Dieser Parameter bestimmt, welche Validierungsmethode genutzt wird, um den RMSEP zu bestimmen.

„LOO“ → leave one out Kreuzvalidierung bei der immer nur eine Variable weggelassen wird

„CV“ → Kreuzvalidierung: Durchführung wird definiert durch segments.CV, segments.type.CV und repetitions. Intern wird dann generate.segments() aufgerufen (siehe 1.5.6 generate.segments()) .

„ex“ → Externe Validierung: Hierbei wird der Datensatz in einen Trainings- und einen Testdatensatz aufgespalten. Das Modell wird mit dem Trainingsdatensatz erstellt und die Validierung wird mit dem Testdatensatz durchgeführt. Der Anteil der Daten, der als Trainingsdaten genutzt wird, wird mit trainvalues.ratio bestimmt.

PLS.method = „simpls“ (character):

Dieser Parameter bestimmt, durch welches Verfahren die Modellbildung erfolgen soll.

„simpls“ → Der SIMPLS-Algorithmus wird genutzt, um ein PLS-Modell zu berechnen (siehe 2.2.3.2 SIMPLS).

„nipals“ → Der NIPALS-Algorithmus wird genutzt, um ein PLS-Modell zu berechnen (siehe 2.2.3.1 NIPALS).

„kernelpls“ → Der Kernel-PLS Algorithmus 1 (Dayal und MacGregor 1997) wird genutzt, um ein PLS-Modell zu berechnen. (Achtung: Es handelt sich hierbei nur um einen anderen Algorithmus, es wird kein Kerneltrick genutzt.)

„widekernelpls“ → Der Kernel-PLS Algorithmus für sehr viel Variablen (Rännar et al. 1994) wird genutzt, um ein PLS-Modell zu berechnen. (Achtung: Es handelt sich hierbei nur um einen anderen Algorithmus, es wird kein Kerneltrick genutzt.)

break.up = „interactive“ (character):

Dieser Parameter bestimmt, wie die Anzahl an Variablen bestimmt werden soll.

„specify.n.vari“ → Setze die Anzahl an übrigen Variablen im Vorhinein fest, sodass solange Variablen hinzugefügt oder entfernt werden, bis die gewünschte Anzahl erreicht wurde. n.variables muss gesetzt werden.

„RMSEP.decrease.break“ → Der Variablenselektionsalgorithmus wird sofort unterbrochen, sobald einmal der RMSEP im Vergleich zum vorherigen Schritt schlechter wird. Dies ist die schnellste Variante, aber die Wahrscheinlichkeit ist hoch nur ein lokales Optimum zu finden.

„best.overall“ → Bei dieser Variante läuft der Algorithmus einmal komplett durch und im Nachhinein wird anhand der „history“ die Anzahl an Variablen ausgewählt, bei der der RMSEP am geringsten war.

„interactive“ → Hierbei wird einem die „history“ angezeigt und man kann selbst anhand dieser entscheiden, welche Anzahl an Variablen genutzt werden soll.

multithread = TRUE (boolean):

Wenn TRUE, dann werden alle verfügbaren Threads des Computers für die Berechnungen genutzt. Dies ist bei großen Datensätzen bedeutend schneller. Bei sehr kleinen Datensätzen

wird die Berechnung jedoch aufgrund von Kopiervorgängen etwas verlängert. Der Multithreadmodus ist sehr RAM-intensiv.

n.variables = NULL (numeric):

Hier kann die Anzahl an übrigen Variablen für break.up = „specify.n.vari“ festgelegt werden. Bei den anderen break.up Einstellungen kann so die Anzahl an Variablen festgelegt werden, bis zu der maximal gerechnet werden soll. So können die Verfahren beschleunigt werden.

ncomp.PCA.PA = 1 (numeric):

Dieser Parameter ist wichtig, wenn selectionalgorithm = „PCA.procrustes“ gewählt wurde, ansonsten kann er ignoriert werden. ncomp.PCA.PA legt fest für wie viele Komponenten die Scores der PCA über die Procrustes Analyse verglichen werden.

centerX = FALSE (boolean):

Wenn TRUE, dann werden die X-Daten nur für die Variablenselektion zentriert. Dies ist für beide Procrustes-Analysen sinnvoll.

standardizeX = FALSE (boolean):

Wenn TRUE, dann werden die X-Daten nur für die Variablenselektion standardisiert.

ncomp = NULL (numeric):

Hier kann eine maximale Anzahl an Komponenten festgelegt werden, die durch die PLSR berechnet wird. Sehr empfehlenswert für den CARS-Algorithmus, um ihn zu beschleunigen.

trainvalues.ratio = 0.75 (numeric):

Dieser Parameter ist nur relevant, wenn validation = „ex“.
Er bestimmt, welcher Anteil der Daten als Trainingsdatensatz dienen soll. Der Rest wird zum Testdatensatz. Der Wert kann im Bereich von 0 bis 1 liegen.

segments.CV = NULL (numeric oder list of segments):

Dieser Parameter ist nur relevant, wenn validation = “CV”.
Dieser muss nicht zwingend gesetzt werden. Wird er nicht gesetzt, dann wird er durch generate.segments() (siehe 1.5.6 generate.segments()) angepasst.
Er bestimmt in wie viele Segmente der Datensatz aufgespalten werden soll.
Alternativ kann hier auch eine Liste, erzeugt durch generate.segments(), übergeben werden.

segments.type.CV = NULL (character):

Dieser Parameter ist nur relevant, wenn validation = “CV”.
Dieser muss nicht zwingend gesetzt werden. Wird er nicht gesetzt, dann wird er durch generate.segments() (siehe 1.5.6 generate.segments()) angepasst. Er bestimmt wie die Aufteilung der Segmente durchgeführt werden soll.

„random“	→	zufällige Zuordnung der Proben zu den Segmenten
„consecutive“	→	die Reihenfolge der Proben wird beibehalten und immer die nebenaneinander liegenden werden einem Segment zugeordnet Bsp: 1,2,3,4,5,6 in 2 Segmente → Segment 1: 1,2,3; Segment 2: 4,5,6
„interleaved“	→	immer eine Probe wird zum 1. Segment zugeordnet, die nächste zum 2. und so weiter, sodass die Proben gleichmäßig auf die Segmente verteilt werden. Bsp: 1,2,3,4,5,6 in 2 Segmente → Segment 1: 1,3,5; Segment 2: 2,4,6

repetitions = data$data.info$read.repetitions() (numeric):

Hier können die Anzahl an Wiederholungen gleicher Datenpunkte angegeben werden. Dies ist aber nicht notwendig, da die Funktion diesen automatisch aus dem angegebenen Wert in der

Einlesefunktion ausliest.
Dieser Wert wirkt sich auf generate.segments() (siehe 1.5.6 generate.segments()) für die Kreuzvalidierung aus.

printplots.TF = FALSE (boolean):
Wenn TRUE, dann werden die Spektren vor und nach der Datenvorbehandlung im Plots Fenster angezeigt.

savedata.TF = TRUE (boolean):
Dieser Wert sollte nicht auf FALSE gesetzt werden, ansonsten erscheint diese Datenvorbehandlung nicht in der Auswertung. Wird benötigt für die Vorhersagefunktion predict.ex().

centeredY = data$data.info$read.centeredY() (boolean);
Dieser Wert sollte nicht von Nutzer verändert werden. Wenn TRUE, dann wird die Zentrierung der Y-Daten in den Einlesefunktionen, für die Auswertung rückgängig gemacht.

Beispiel:

Variante 1:

```
Gasoline <- variableselection(data = Gasoline, selectionalgorithm = "CARS", direction = "backwards",
    validation = "CV", break.up = "best.overall", multithread = TRUE, ncomp = 10, segments.CV = 10,
    segments.type.CV = "interleaved")
```

Variante 2:

```
Gasoline <- variableselection(data = Gasoline, selectionalgorithm = "PCA.procrustes", direction = "forwards",
    break.up = "interactive", multithread = TRUE, n.variables = 150, ncomp.PCA.PA = 10, centerX = TRUE,
    ncomp = 10)
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen.
In Variante 1 wird der CARS-Algorithmus zur Rückwärtseliminierung der Variablen genutzt. Als Validierungsmethode wurde die Kreuzvalidierung mit 10 Segmenten genutzt. Diese wurden über den Segmenttype „interleaved“ aufgeteilt. Abgebrochen wurde erst nach dem kompletten Durchlauf des Algorithmus und dann die Anzahl an Komponenten ausgewählt, sodass der RMSEP minimal ist. Das Ergebnis von Variante 1 ist in Abbildung 23 zu sehen.
In Variante 2 wird der PCA-Procurstes Algorithmus zur Vorwärtsselektion der Variablen genutzt. Hierfür werden die ersten 10 Komponenten der PCA genutzt. Abgebrochen wird, sobald 150 Variablen ausgewählt wurden. Da als break.up „interactive“ gewählt wurde, kann der Nutzer nun die Anzahl der Variablen, anhand der aufgezeichneten Daten, auswählen. Das Ergebnis der Variante 2 ist in Abbildung 24 zu sehen.

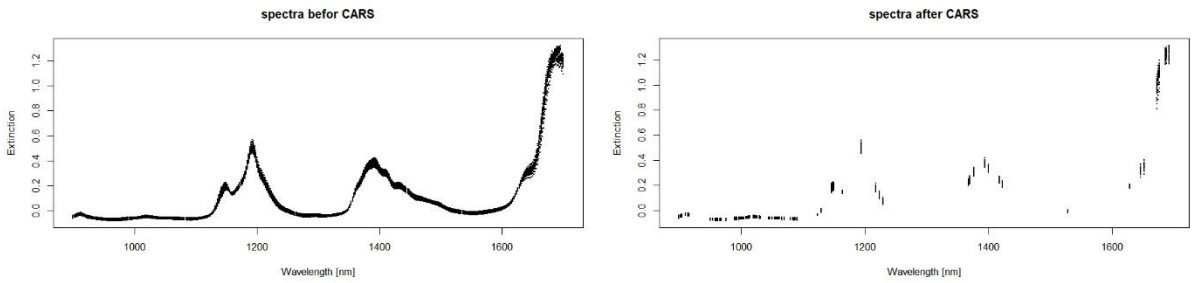


Abbildung 23: Spektren bevor (links) und nach (rechts) CARS Rückwärtseliminierung durch variableselection()

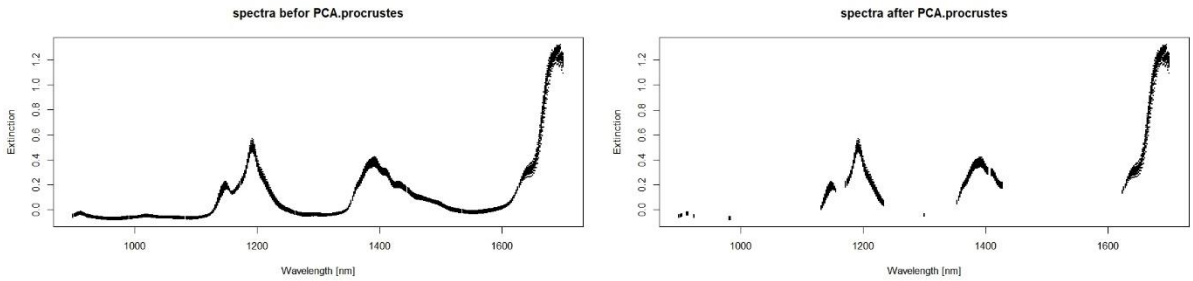


Abbildung 24: Spektren bevor (links) und nach (rechts) PCA-Procrustes Vorwärtsselektion durch variableselection()

1.4.20 load.variableselection()

Diese Funktion dient dazu, eine vorher durch 1.7.4 save.variableselection() gespeicherte Variablenselektion zu laden und auf den Datensatz anzuwenden.

Parameter

data = NULL (PLSR.Kronsbein Datensatz):

Datensatz, welcher durch eine der Einlesefunktionen erzeugt wurde, übergeben.

newNumberofchangedVariables = NULL (numeric):

Wenn die Anzahl an geänderten Variablen, im Vergleich zur gespeicherten Variablenselektion, geändert werden soll, kann dies hier angegeben werden. Bei einer durchgeführten Vorwärtsselektion wird diese Anzahl an Variablen genutzt. Bei einer durchgeführten Rückwärtselimierung wird diese Anzahl an Variablen entfernt.

directory = NULL (character):

Hier kann das Verzeichnis angegeben werden, indem die gespeicherte Variablenselektion liegt. Wenn NULL, dann muss die gespeicherte Variablenselektion im working directory liegen.

filename = "selectedvariables.RData" (character):

Hier kann der Dateiname der gespeicherten Variablenselektion angegeben werden. Falls nicht angegeben, dann wird der Standardname "selectedvariables.RData" genutzt.

savedata.TF = TRUE (boolean):

Dieser Wert sollte nicht auf FALSE gesetzt werden, ansonsten erscheint diese Datenvorbehandlung nicht in der Auswertung. Wird benötigt für die Vorhersagefunktion predict.ex().

Beispiele:

Vorher durchgeführt Funktionen:

```
Gasoline <- variableselection(data = Gasoline, selectionalgorithm = "CARS", direction = "backwards",
        validation = "CV", break.up = "best.overall", multithread = TRUE, ncomp = 10, segments.CV = 10,
        segments.type.CV = "interleaved")
save.variableselection(data = Gasoline, selectionalgorithm = "CARS", filename = "CARS.selection.RData",
        directory = "test1")
```

Variante 1:

```
Gasoline <- load.variableselection(data = Gasoline, directory = "test1",  filename = "CARS.selection.RData")
```

Variante 2:

```
Gasoline <- load.variableselection(data = Gasoline, newNumberofchangedVariables = 373, directory = "test1",
        filename = "CARS.selection.RData")
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. Damit load.variableselection durchgeführt werden kann, muss eine Variablenselektion abgespeichert worden sein, wie unter den vorher durchgeführten Funktionen dargestellt. Die Variablenselektion wurde im Unterordner des working directorys „test1“ in der Datei „CARS.selection.RData“ abgespeichert.

Die durchgeführte Variablenselektion kann auf den aktuellen Datensatz, wie in Variante 1 beschrieben angewendet werden. Es ist aber auch möglich, die Anzahl der Variablen über newNumberofchangedVariables zu ändern. In Variante 2 sollen 373 Variablen geändert werden, sodass 28 von 401 Variablen übrigbleiben. Das Ergebnis der Variante 2 ist in Abbildung 25 zu sehen.

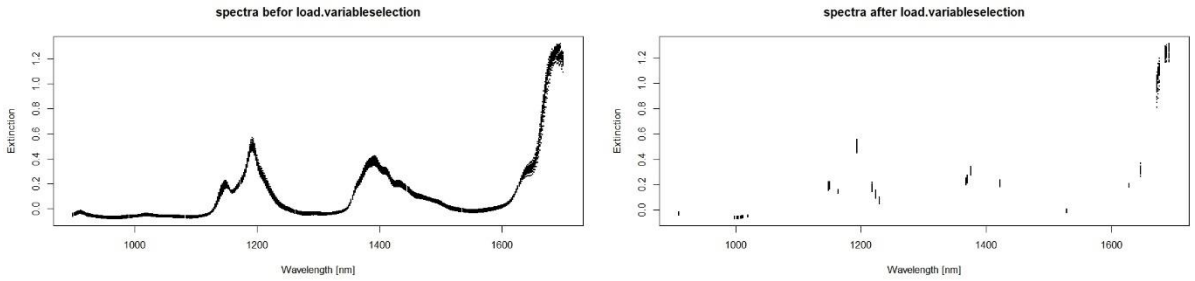


Abbildung 25: Spektren bevor (links) und nach (rechts) der Variante 2 von load.variableselection()

1.4.21 reduce_variables_spectra()

Diese Funktion dient dazu, die Anzahl an Variablen in Spektren „auszudünnen“. Dies kann sinnvoll sein, um andere Funktionen zu beschleunigen, wie zum Beispiel die Variablenselektion. Hierbei wird einfach ein Teil der Variablen weggelassen („nur jede so und so vielte Variable wird übrig gelassen“). Sinnvoll ist es vorher, eine Glättung durch smoothspectrums.polynomial() oder smoothspectrums.mean durchzuführen.

Parameter

data = NULL (PLSR.Kronsbein Datensatz):

Datensatz, welcher durch eine der Einlesefunktionen erzeugt wurde, übergeben.

degreeofreduction = 2 (numeric):

Dieser Parameter legt fest, welche wievielte Variable behalten wird. Er muss ganzzahlig und größer als 1 sein. Es wird immer versucht die mittlere Variable zu behalten.

Bsp: Variable 1,2,3,4,5,6,7,8 bei degreeofreduction = 2 → Variable 1,3,5,7
Variable 1,2,3,4,5,6,7,8 bei degreeofreduction = 3 → Variable 2,5,8

printplots.TF = FALSE (boolean):

Wenn TRUE, dann werden die Spektren vor und nach der Datenvorbehandlung im Plots Fenster angezeigt.

savedata.TF = TRUE (boolean):

Dieser Wert sollte nicht auf FALSE gesetzt werden, ansonsten erscheint diese Datenvorbehandlung nicht in der Auswertung. Wird benötigt für die Vorhersagefunktion predict.ex().

Beispiel

```
Gasoline <- reduce_variables_spectra(data = Gasoline, degreeofreduction = 3)
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. Es werden 2 von 3 Variablen bei dieser Konfiguration entfernt. Dies empfiehlt sich, um eine Variablenselektion zu

beschleunigen. Es ist sinnvoll vor der Reduktion der Variablen eine Glättungsfunktion durchzuführen. Das Ergebnis der Reduktion der Variablen ist in Abbildung 26 zu sehen.

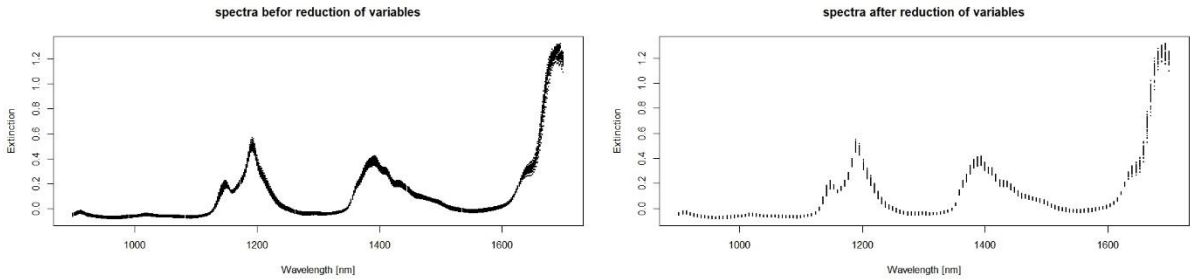


Abbildung 26: Spektren bevor (links) und nach (rechts) reduce_variables_spectra()

1.4.22 calc.meanspectra()

Diese Funktion dient dazu, immer eine gewisse Anzahl an Spektren zu mitteln. Dies kann sinnvoll sein, um zufälliges Rauschen zu minimieren.

Parameter

data = NULL (PLSR.Kronsbein Datensatz):
Datensatz, welcher durch eine der Einlesefunktionen erzeugt wurde, übergeben.

mean.n.spectra = NULL (numeric):
Die Anzahl an aufeinander folgenden Spektren, welche gemittelt werden sollen.

all.repetitions = FALSE (boolean):
Wenn dieser Wert TRUE ist, dann wird mean.n.spectra auf den Wert von repetitions gesetzt. Eingaben in mean.n.spectra werden ignoriert.

repetitions = data$data.info$read.repetitions() (numeric):
Hier kann die Anzahl an Wiederholungen gleicher Datenpunkte angegeben werden. Dies ist aber nicht notwendig, da die Funktion diese automatisch aus dem angegebenen Wert in der Einlesefunktion ausliest.

printplots.TF = FALSE (boolean):
Wenn TRUE, dann werden die Spektren vor und nach der Datenvorbehandlung im Plots Fenster angezeigt.

savedata.TF = TRUE (boolean):
Dieser Wert sollte nicht auf FALSE gesetzt werden, ansonsten erscheint diese Datenvorbehandlung nicht in der Auswertung. Wird benötigt für die Vorhersagefunktion predict.ex().

called.with.predict = FALSE (boolean):
Dieser Wert sollte nicht auf TRUE gesetzt werden. Er wird nur beim internen Aufruf in der predict.ex() Funktion benötigt.

Beispiel

Vorher durchgeführt Funktionen:

```
Bespiel <- dataprep.plsr(X.values = Beispiel.X, Y.values = Beispiel.Y, wavelengths = wavelengths,
        UnitspecY = "Extinction", UnitspecX = "Wavelength [nm]", repetitions = 9)
```

Variante 1:

```
Beispiel <- calc.meanspectra(data = Beispiel, all.repetitions = TRUE)
```

Variante 2:

```
Beispiel <- calc.meanspectra(data = Beispiel, mean.n.spectra = 3)
```

Zusätzliche Informationen:

Bei Beispiel handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. In diesem Datensatz wurde die Messung für jeden Y-Datenpunkt 9 mal wiederholt, wie unter vorher durchgeführte Funktionen zu sehen. In Variante 1 wurden alle 9 wiederholten Spektren zu einem neuen Mittelwertspektrum gemittelt. In Variante 2 wurden immer 3 Spektren zu einem Mittelwertspektrum gemittelt, sodass aus 9 wiederholten Spektren noch 3 wiederholte Mittelwertspektren übrigbleiben.

1.4.23 IIR.correction()

Diese Funktion dient dazu, eine independent interference reduction (siehe 2.3.8.1 IIR) durchzuführen. Hierfür wird ein normaler Spektrendatensatz, erzeugt durch Messen verschiedener Proben, benötigt. Dieser dient dazu, später ein Modell zu erstellen. Zusätzlich ist ein einfacher Spektrendatensatz mit vielen Wiederholmessung derselben Probe notwendig. Dieser wird benötigt, um die Interferenzen zu modellieren.

Parameter

data = NULL (PLSR.Kronsbein Datensatz):

Datensatz, welcher durch eine der Einlesefunktionen erzeugt wurde, übergeben. Hier wird der Datensatz übergeben, mit dem die spätere Kalibration erfolgen soll.

data.simple = NULL (PLSR.Kronsbein Datensatz):

Datensatz, welcher durch eine der Einlesefunktionen erzeugt wurde, übergeben. Hier wird der Datensatz übergeben, welcher die Interferenz abbildet. Die Spektren müssen alle von der gleichen Probe stammen, sodass die Varianz in den Spektren nur aus der Interferenz stammt.

ncomp = NULL (numeric):

Während der IIR werden Eigenvektoren berechnet durch eine PCA genutzt. Hier kann angegeben werden, welche Anzahl an Hauptkomponenten der Eigenvektorenmatrix genutzt werden sollen. Wird kein Wert angegeben, dann werden alle genutzt.

comp.explained.variance = NULL (numeric):

Hier kann eine Zahl zwischen 0 und 1 angegeben werden. Die Anzahl der genutzten Komponenten wird dann dementsprechend gewählt, sodass mindestens dieser Anteil der Varianz durch die Komponenten erklärt wird.

printplots.TF = FALSE (boolean):

Wenn TRUE, dann werden die Spektren vor und nach der Datenvorbehandlung im Plots Fenster angezeigt.

savedata.TF = TRUE (boolean):

Dieser Wert sollte nicht auf FALSE gesetzt werden, ansonsten erscheint diese Datenvorbehandlung nicht in der Auswertung. Wird benötigt für die Vorhersagefunktion predict.ex().

Beispiel

```
Beispiel <- IIR.correction(data = Beispiel, data.simple = Beispiel.simple, ncomp = 5)
```

Zusätzliche Informationen:

Bei Beispiel und bei Beispiel.einfach handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. Bei Beispiel handelt es sich um den Datensatz, welcher dazu dienen soll, eine Regression zu erstellen. Beispiel.einfach hingegen wird nur während der IIR benötigt, um die Interferenzen, welche in Beispiel

vorhanden sind, zu entfernen. Hierfür wurde die gleiche Probe sehr häufig gemessen. ncomp = 5 gibt an, dass die ersten 5 Komponenten der Eigenvektormatrix, während der IIF Berechnung genutzt werden, sollen.

1.4.24 Ydata_transformation()

Diese Funktion dient dazu, die Y-Daten zu transformieren.

Parameter:

data = NULL (PLSR.Kronsbein Datensatz):

Datensatz, welcher durch eine der Einlesefunktionen erzeugt wurde, übergeben.

fun = NULL (character):

Hier können vorgefertigte Transformationsfunktionen genutzt werden. Die passende Rücktransformationsfunktion wird ebenfalls automatisch genutzt.

„inverse“ → $X = 1/X$
„log10“ → $X = \log_{10}(X)$
„square“ → $X = X^2$
„root“ → $X = X^{0.5}$

transformationFUN = NULL (function):

Hier kann eine eigene Transformationfunktion übergeben werden. Falls dies getan wird, ist die Einstellung, welche über fun getätigt wird, irrelevant.

backtransformationFUN = NULL (function):

Hier sollte eine Rücktransformationsfunktion übergeben werden, falls eine eigene Transformationsfunktion genutzt wurde. Wenn dies nicht möglich ist, dann sollte originalY in der genutzten Einlesefunktion auf FALSE gesetzt werden.

whichYvalues = 1:ncol(data$prepdata$Y) (numeric vector):

Hier kann ausgewählt werden, welche Spalten der Y-Datenmatrix transformiert werden sollen. Standardmäßig werden alle transformiert.

savedata.TF = TRUE (boolean):

Dieser Wert sollte nicht auf FALSE gesetzt werden, ansonsten erscheint diese Datenvorbehandlung nicht in der Auswertung. Wird benötigt für die Vorhersagefunktion predict.ex().

centeredY = data$data.info$read.centeredY() (boolean):

Dieser Wert sollte nicht vom Nutzer verändert werden. Wenn TRUE, dann wird die Zentrierung der Y-Daten in den Einlesefunktionen für die Auswertung rückgängig gemacht.

othermean_for_centerY = NULL (numeric):

Dieser Parameter wird nur beim internen Aufruf, während der externen Validierung benötigt.

silent = FALSE (boolean):

Wenn TRUE, dann werden von der Funktion keine Ausgaben in der Konsole getätigt.

Beispiel:

Variante 1:

```
Beispiel <- Ydata_transformation(data = Beispiel, fun = „square“)
```

Variante 2:

```
transformationFUN <- function(X){
 X <- 1/(X^0.5)
 return(X)}
```

```
backtransformationFUN <- function(X){
 X <- (1/X)^2
 return(X)}
Beispiel <- Ydata_transformation(data = Beispiel, transformationFUN = transfomationFUN,
          backtransformationFUN = backtransformationFUN)
```

Zusätzliche Informationen:
Bei Beispiel handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. In Variante 1 werden alle Y-Werte quadriert. In Variante 2 wird eine individuelle Transformationsfunktion und die entsprechende Rücktransformationsfunktion genutzt.

1.4.25 reduce.calibrationrange()

Diese Funktion dient dazu, einen Teil des Datensatzes auszuwählen, um den Kalibrierbereich zu beschränken. Dies kann in manchen Fällen sinnvoll, um ein lineares PLS Modell zu erhalten.

Parameter:

data = NULL (PLSR.Kronsbein Datensatz):
Datensatz, welcher durch eine der Einlesefunktionen erzeugt wurde, übergeben.

newcalrange = NULL (numeric vector length 2):
Hier kann der neue Kalibrierbereich anhand von Y-Daten begrenzt werden. Dies geschieht durch Übergabe eines Vektors der Länge 2. Die beiden Werte stehen für den minimalen und maximalen Y-Wert.

Ycol = 1 (numeric):
Hier wird festgelegt, auf welche Spalte der Y-Datenmatrix sich newcalrange bezieht.

whichY = NULL (boolean vector):
Dieser Parameter wird nur beim Aufruf als interne Funktion genutzt.

pred = FALSE (boolean):
Dieser Parameter wird nur beim Aufruf als interne Funktion genutzt.

centeredY = data$data.info$read.centeredY() (boolean):
Dieser Wert sollte nicht von Nutzer verändert werden. Wenn TRUE, dann wird die Zentrierung der Y-Daten in den Einlesefunktionen für die Auswertung rückgängig gemacht.

printplots.TF = FALSE (boolean):
Wenn TRUE, dann werden die Spektren vor und nach der Datenvorbehandlung im Plots Fenster angezeigt.

savedata.TF = TRUE (boolean):
Dieser Wert sollte nicht auf FALSE gesetzt werden, ansonsten erscheint diese Datenvorbehandlung nicht in der Auswertung. Wird benötigt für die Vorhersagefunktion predict.ex().

Beispiel:

```
Gasoline <- reduce.calibrationrange(data = Gasoline, newcalrange = c(85,88))
```

Zusätzliche Informationen:
Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. Es werden nur die Spektren ausgewählt, welche einen Y-Wert von 85 bis 88 besitzen. Dies wird durch den Vektor festgelegt, welche mit dem Befehl c(85,88) erzeugt wurde.

1.5 PCA-, PCR-, PLSR-, Validierungs- und Vorhersagefunktionen

Diese Kategorie deckt alle Funktionen ab, welche für die Modellbildung, Validierung und Vorhersage mithilfe eines bestehenden Modells zuständig sind.

Aufgabe	Funktionsaufruf	Name im Speicher	Quellcodedatei
Alle PCA Daten berechnen	PCA.calc()	PCA.calc	PCA.R
Berechnung des Modells (PLSR; PCR) und Validierung	plsr.ex()	plsr.ex; (zusätzlich bei Validierung: crossvalidation; externalvalidation)	plsr.functions.R
Externe Validierung im Nachhinein	externalvalidation()	externalvalidation	externalvalidation.R
Segmente für die Kreuzvalidierung erzeugen	generate.segments()		crossvalidation.R
Vorhersage mithilfe eines Modells und neuer X-Daten	predict.ex()	prediction	predict.R
Berechnung Kernel PLS	kpls()	plsr.ex	kpls.R
Kreuvalidierung im nach hinein	crossvalidation()	crossvalidation	crossvalidation.R

Tabelle 3: Übersicht PCA-, PCR-, PLSR-, Validierungs- und Vorhersagefunktionen

1.5.1 PCA.calc()

Diese Funktion dient dazu, die Hauptkomponenten der PCA und alle anderen wichtigen Werte zu berechnen.
Diese Funktion greift auf die prcomp() Funktion aus dem stats-Package (R Core Team (2020)) zu.

Parameter

data = NULL (PLSR.Kronsbein Datensatz):

Datensatz, welcher durch eine der Einlesefunktionen erzeugt wurde, übergeben.

center = FALSE (boolean):

Wenn TRUE, dann werden die X-Daten nur für die PCA zentriert.

standardize = FALSE (boolean):

Wenn TRUE, dann werden die X-Daten nur für die PCA standardisiert.

savedata.TF = TRUE (boolean):

Dieser Wert sollte nicht auf FALSE gesetzt werden, ansonsten erscheint die PCA-Berechnung nicht in der Auswertung.

Beispiel

```
Gasoline <- PCA.calc(data = Gasoline, center = TRUE)
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. Die Auswertung der PCA erfolgt durch die Auswertefunktion evaluation().

1.5.2 plsr.ex()

Diese Funktion dient dazu, aus einem vorbereiteten Datensatz, ein PLSR oder PCR Modell zu berechnen. Hierbei kann auch gleich eine Validierung durchgeführt werden.

Diese Funktion greift auf die plsr() Funktion aus dem pls-Package (Bjørn-Helge Mevik, Ron Wehrens and Kristian Hovde Liland (2019)) zu.

Parameter

data = NULL (PLSR.Kronsbein Datensatz):

Datensatz, welcher durch eine Einlesefunktion erzeugt wurde, übergeben.

validation = „none“ (character):

Dieser Parameter bestimmt, welche Validierungsmethode genutzt wird, um den RMSEP zu bestimmen.

„none“	→	keine Validierung wird durchgeführt
„LOO“	→	leave one out Kreuzvalidierung, bei der immer nur eine Variable weggelassen wird
„CV“	→	Kreuzvalidierung: Durchführung wird definiert durch segments.CV, segments.type.CV und repetitions. Intern wird dann generate.segments() (siehe 1.5.6 generate.segments()) aufgerufen.
„ex“	→	Externe Validierung: Hierfür muss vor dem Aufrufen der Funktion plsr.ex(), die Funktion generate.split_and_splitdata.externalvalidation() aufgerufen worden sein. Diese spaltet den Datensatz in einen Trainingsdatensatz für die Modellbildung und einen Testdatensatz für die externe Validierung auf.

ncomp = NULL (numeric):

Hier kann die Anzahl der Komponenten festgelegt werden, welche berechnet werden soll. Wenn der Wert nicht gesetzt wird, dann wird die maximal mögliche Anzahl an Komponenten berechnet.

model.calc = „simpls“ (character):

Dieser Parameter bestimmt, durch welches Verfahren die Modellbildung erfolgen soll.

„simpls“	→	Der SIMPLS-Algorithmus wird genutzt, um ein PLS-Modell zu berechnen (siehe 2.2.3.2 SIMPLS).
„nipals“	→	Der NIPALS-Algorithmus wird genutzt, um ein PLS-Modell zu berechnen (siehe 2.2.3.1 NIPALS).
„kernelpls“	→	Der Kernel-PLS Algorithmus 1 (Dayal und MacGregor 1997) wird genutzt, um ein PLS-Modell zu berechnen. (Achtung: Es handelt sich hierbei nur um einen anderen Algorithmus, es wird kein Kerneltrick genutzt.)
„widekernelpls”	→	Der Kernel-PLS Algorithmus für sehr viel Variablen (Rännar et al. 1994) wird genutzt, um ein PLS-Modell zu berechnen. (Achtung: Es handelt sich hierbei nur um einen anderen Algorithmus, es wird kein Kerneltrick genutzt.)
“PCR”	→	Die PCA und dann die MLR wird genutzt, um ein PCR-Modell zu berechnen (siehe 2.2.2 PCR).

centeredY = data$data.info$read.centeredY() (boolean);

Dieser Wert sollte nicht vom Nutzer verändert werden. Wenn TRUE, dann wird die Zentrierung der Y-Daten in den Einlesefunktionen für die Auswertung rückgängig gemacht.

segments.CV = NULL (numeric oder list of segments):

Dieser Parameter ist nur relevant, wenn validation = “CV”.

Dieser muss nicht zwingend gesetzt werden. Wird er nicht gesetzt, dann wird er durch generate.segments() (siehe 1.5.6 generate.segments()) angepasst.

Er bestimmt in wie viele Segmente der Datensatz aufgespalten werden soll.
Alternativ kann hier auch eine Liste, erzeugt durch generate.segments(), übergeben werden.

segments.type.CV = NULL (character):

Dieser Parameter ist nur relevant, wenn validation = “CV”.
Dieser muss nicht zwingend gesetzt werden. Wird er nicht gesetzt, dann wird er durch generate.segments() (siehe 1.5.6 generate.segments()) angepasst. Er bestimmt wie die Aufteilung der Segmente durchgeführt werden soll.

„random“ → zufällige Zuordnung der Proben zu den Segmenten

„consecutive“ → die Reihenfolge der Proben wird beibehalten und immer die nebenaneinander liegenden werden einem Segment zugeordnet
Bsp.: 1,2,3,4,5,6 in 2 Segmente → Segment 1: 1,2,3; Segment 2: 4,5,6

„interleaved“ → immer eine Probe wird zum 1. Segment zugeordnet, die nächste zum 2. und so weiter, sodass die Proben gleichmäßig auf die Segmente verteilt werden.
Bsp.: 1,2,3,4,5,6 in 2 Segmente → Segment 1: 1,3,5; Segment 2: 2,4,6

repetitions = data$data.info$read.repetitions() (numeric):

Hier können die Anzahl an Wiederholungen gleicher Datenpunkte angegeben werden. Dies ist aber nicht notwendig, da die Funktion diese automatisch aus dem angegebenen Wert in der Einlesefunktion ausliest.
Dieser Wert wirkt sich auf generate.segments() (siehe 1.5.6 generate.segments()) für die Kreuzvalidierung aus.

savedata.TF = TRUE (boolean):

Dieser Wert sollte nicht auf FALSE gesetzt werden, ansonsten erscheint die Modellbildung nicht in der Auswertung.

Beispiel

Variante 1:

```
Gasoline <- plsr.ex(data = Gasoline, validation = "CV", ncomp = 10, model.calc = "simpls", segments.CV = 20,
        segments.type.CV = "random")
```

Variante 2:

```
Gasoline <- generate.split_and_splitdata.externalvalidation(data = Gasoline, trainvalues.ratio = 0.75)
Gasoline <- plsr.ex(data = Gasoline, validation = "ex", ncomp = 10, model.calc = "nipals")
```

Variante 3:

```
Gasoline <- plsr.ex(data = Gasoline, validation = "LOO", ncomp = 10, model.calc = "PCR")
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen.
In Variante 1 wird ein PLS-Modell mithilfe des SIMPLS-Algorithmus berechnet. Für die Validierung wird eine Kreuzvalidierung mit 20 Segmenten genutzt, diese werden zufällig ausgewählt.
In Variante 2 wird ein PLS-Modell mithilfe des NIPALS-Algorithmus berechnet. Für die Validierung wird die externe Validierung genutzt. Dafür wurde der Datensatz zuvor durch generate.split_and_splitdata.externalvalidation() in einen Trainings- und Testdatensatz aufgeteilt.
In Variante 3 wird ein PCR-Modell berechnet und mittels Kreuzvalidierung validiert. Es wurde „LOO“ gewählt, somit wird immer nur eine Probe weggelassen.
Die Auswertung der PLSR oder PCR erfolgt durch die Auswertefunktion evaluation().

1.5.3 kpls()

Diese Funktion dient dazu, eine Kernel-PLS durchzuführen. Dabei wird eine Kernelmatrix, aufgrund einer gewählten Kernelfunktion berechnet. Danach erfolgt die Berechnung des PLS-Modells, mittels

angepasstem NIPALS-Algorithmus.
Die Funktion kpls() ist komplementär zur plsr.ex() Funktion und wird im Speicher unter demselben Namen plsr.ex abgespeichert. So wird erreicht, dass die Validierungs-, Vorhersage- und Auswertefunktionen genau gleich auf ein Kernel-PLS Modell, wie auch auf ein normales PLS-Modell, angewendet werden können.

Parmameter

data = NULL (PLSR.Kronsbein Datensatz):

Datensatz, welcher durch eine der Einlesefunktionen erzeugt wurde, übergeben.

ncomp = NULL (numeric):

Hier kann die Anzahl der Komponenten festgelegt werden, welche berechnet werden soll. Wenn der Wert nicht gesetzt wird, dann wird die maximal mögliche Anzahl an Komponenten berechnet.

kfun = „linear“ (character):

Hier wird der Typ der Kernelfunktion, der genutzt werden soll, festgelegt. Alternativ kann hier auch eine Kernelfunktion übergeben werden, dann wird diese genutzt. Die genauen Kernelfunktion sind bitte 2.2.4.1 Kernelfunktionen zu entnehmen. In Klammern stehen die Parameter, welche einen Einfluss auf die Kernelfunktion haben. Die anderen Kernel-Parameter werden ignoriert.

„linear“	→	lineare Kernelfunktion ()
„gauss“	→	radiale Kernelfunktion, Gaußfunktion (sigma)
„poly“	→	polynomische Kernelfunktion (degree, scale, offset)
„tan“	→	Hyperbolische Tangens Kernelfunktion (scale, offset)
„laplace“	→	Laplace Kernelfunktion (sigma)
„bessel“	→	Bessel‘sche Kernelfunktion (sigma, order, degree)
„anova“	→	ANOVA Kernelfunktion (sigma, degree)
„spline“	→	Spline Kernelfunktion ()

sigma = 1 (numeric):

Parameter für die Kernelfunktionen, welcher die Kernweite einer Funktion beschreibt.

degree = 1 (numeric):

Parameter für die Kernelfunktion, welcher den Grad der Funktion beschreibt. Es muss sich um einen positiven ganzzahligen Wert handeln.

scale = 1 (numeric):

Parameter für die Kernelfunktion, um den Kernel zu skalieren.

offset = 1 (numeric):

Parameter für die Kernelfunktion, welcher den Versatz in der Funktion beschreibt.

order = 1 (numeric):

Parameter für die Kernelfunktion, welcher die Ordnung einer Differentialfunktion beschreibt.

centeredY = data$data.info$read.centeredY() (boolean);

Dieser Wert sollte nicht vom Nutzer verändert werden. Wenn TRUE, dann wird die Zentrierung der Y-Daten in den Einlesefunktionen, für die Auswertung, rückgängig gemacht.

repetitions = data$data.info$read.repetitions() (numeric):

Hier können die Anzahl an Wiederholungen gleicher Datenpunkte angegeben werden. Dies ist

aber nicht notwendig, da die Funktion diese automatisch aus dem angegebenen Wert in der Einlesefunktion ausliest.

savedata.TF = TRUE (boolean):
Dieser Wert sollte nicht auf FALSE gesetzt werden, ansonsten erscheint die Modellbildung nicht in der Auswertung.

fast = FALSE (boolean):
Dieser Wert sollte nicht auf TRUE gesetzt werdem. Er ist nur TRUE, wenn diese Funktion intern, während der Kreuzvalidierung aufgerufen wird, um die Berechnung zu beschleunigen.

Beispiel

Beispiel <- kpls(data = Beispiel, ncomp = 10, kfun = "poly", degree = 2, offset = 0)

Zusätzliche Informationen:
Bei Beispiel handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. Die X-Datenmatrix aus Beispiel wird über die polynomische Kernelfunktion 2. Grades in eine Kernelmatrix transformiert. Der Offset beträgt dabei 0. Aus der Kernelmatrix und den Y-Daten wird dann mittels angepasstem NIPALS-Algoithmus ein Modell berechnet.

1.5.4 crossvalidation()

Diese Funktion dient dazu, nach erfolgter Modellbildung, eine Kreuzvalidierung durchzuführen. Sie wird in manchen Fällen auch bei der Validierung in plsr.ex() aufgerufen. Falls möglich, wird aber die Validierungsfunktion aus dem Package pls (Bjørn-Helge Mevik, Ron Wehrens and Kristian Hovde Liland (2019)) genutzt, da diese schneller ist.

Parameter

model = NULL (PLSR.Kronsbein Datensatz):
Datensatz, bei dem ein Modell durch plsr.ex oder kpls erzeugt wurde, übergeben.

validation = „CV“ (character):
Dieser Parameter bestimmt, welche Validierungsmethode genutzt wird, um den RMSEP zu bestimmen.

„LOO“	→	leave one out Kreuzvalidierung, bei der immer nur eine Variable weggelassen wird
„CV“	→	Kreuzvalidierung: Die Durchführung wird definiert durch segments.CV, segments.type.CV und repetitions. Intern wird dann generate.segments() (siehe 1.5.6 generate.segments()) aufgerufen.

segments.CV = NULL (numeric oder list of segments):
Dieser Parameter ist nur relevant, wenn validation = “CV”.
Dieser muss nicht zwingend gesetzt werden. Wird er nicht gesetzt, dann wird er durch generate.segments() (siehe 1.5.6 generate.segments()) angepasst.
Er bestimmt in wie viele Segmente der Datensatz aufgespalten werden soll.
Alternativ kann hier auch eine Liste, erzeugt durch generate.segments(), übergeben werden.

segments.type.CV = NULL (character):
Dieser Parameter ist nur relevant, wenn validation = “CV”.
Dieser muss nicht zwingend gesetzt werden. Wird er nicht gesetzt, dann wird er durch generate.segments() (siehe 1.5.6 generate.segments()) angepasst. Er bestimmt wie die Aufteilung der Segmente durchgeführt werden soll.

„random“	→	zufällige Zuordnung der Proben zu den Segmenten
„consecutive“	→	die Reihenfolge der Proben wird beibehalten und immer die

nebenaneinander liegenden werden einem Segment zugeordnet
Bsp.: 1,2,3,4,5,6 in 2 Segmente → Segment 1: 1,2,3; Segment 2: 4,5,6

„interleaved“ → immer eine Probe wird zum 1. Segment zugeordnet, die nächste zum 2. und so weiter, sodass die Proben gleichmäßig auf die Segmente verteilt werden.
Bsp.: 1,2,3,4,5,6 in 2 Segmente → Segment 1: 1,3,5; Segment 2: 2,4,6

repetitions = data$data.info$read.repetitions() (numeric):

Hier kann die Anzahl an Wiederholungen gleicher Datenpunkte angegeben werden. Dies ist aber nicht notwendig, da die Funktion diese automatisch aus dem angegebenen Wert in der Einlesefunktion ausliest.
Dieser Wert wirkt sich auf generate.segments() (siehe 1.5.6 generate.segments()) für die Kreuzvalidierung aus.

savedata.TF = TRUE (boolean):

Dieser Wert sollte nicht auf FALSE gesetzt werden, ansonsten erscheint die Validierung nicht in der Auswertung.

centeredY = data$data.info$read.centeredY() (boolean);

Dieser Wert sollte nicht vom Nutzer verändert werden. Wenn TRUE, dann wird die Zentrierung der Y-Daten in den Einlesefunktionen, für die Auswertung rückgängig gemacht.

which.n.method = NULL (numeric):

Dieser Parameter muss nur beachtet werden, wenn mehrere Modelle berechnet wurden. Dann nutzt diese Funktion automatisch immer das letzte berechnete Modell. Soll die externe Validierung jedoch für ein vorheriges Modell durchgeführt werden, dann kann hier die entsprechende Nummer angegeben werden (1 für das erste berechnete Modell, 2 für das Zweite, usw.).

Beispiel

```
Gasoline <- crossvalidation(model = Gasoline, validation = "CV", segments.CV = 20, segments.type.CV =
    "random")
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen.
Der Datensatz Gasoline muss schon ein PLSR.ex oder kpls Modell enthalten. Für dieses Modell wird eine Kreuzvalidierung durchgeführt. Hierfür wird der Datensatz in 20 Segmente unterteilt, welche in diesem Fall zufällig ausgewählt werden.

1.5.5 externalvalidation()

Diese Funktion dient dazu, nach der Modellbildung mit einem neuen Test/Validierungsdatensatz eine externe Validierung durchzuführen. Sie wird auch in plsr.ex() bei validation = „ex“ aufgerufen.

Parameter

model = NULL (PLSR.Kronsbein Datensatz):

Datensatz, bei dem ein Modell durch plsr.ex oder kpls erzeugt wurde, übergeben.

testdata = NULL (numeric matrix):

Hier kann ein neuer Testdatensatz als Matrix übergeben werden. Wichtig ist, dass der Testdatensatz die gleiche Anzahl an Variablen, wie der ursprüngliche Trainingsdatensatz besitzt.
Alternativ kann aber auch testdata.source genutzt werden oder keins von beidem, wenn der

ursprüngliche Datensatz durch generate.split_and_splitdata.externalvalidation() aufgeteilt wurde.

Ytestdata = NULL (numeric matrix):
Hier können die entsprechenden Y-Werte für den Testdatensatz übergeben werden. Wichtig ist, dass der der Testdatensatz die gleiche Anzahl an Y-Variablen, wie der ursprüngliche Trainingsdatensatz besitzt.
Alternativ kann aber auch Ytestdata.source genutzt werden oder keines von beidem, wenn der ursprüngliche Datensatz durch generate.split_and_splitdata.externalvalidation() aufgeteilt wurde.

testdata.source = NULL (character):
Alternative zu testdata. Hier kann der Dateiname einer .csv Datei übergeben werden, in der X-Variablen des Testdatensatzes stehen.

Ytestdata.source = NULL (character):
Alternative zu Ytestdata. Hier kann der Dateiname einer .csv Datei übergeben werden, in der die Y-Variablen des Testdatensatzes stehen.

savedata.TF = TRUE (boolean):
Dieser Wert sollte nicht auf FALSE gesetzt werden, ansonsten erscheint die Validierung nicht in der Auswertung.

centeredY = data$data.info$read.centeredY() (boolean):
Dieser Wert sollte nicht vom Nutzer verändert werden. Wenn TRUE, dann wird die Zentrierung, der Y-Daten in den Einlesefunktionen, für die Auswertung rückgängig gemacht.

which.n.method = NULL (numeric):
Dieser Parameter muss nur beachtet werden, wenn mehrere Modelle berechnet wurden. Dann nutzt diese Funktion automatisch immer das letzte berechnete Modell. Soll die externe Validierung jedoch für ein vorheriges Modell durchgeführt werden, dann kann hier die entsprechende Nummer angegeben werden (1 für das erste berechnete Modell, 2 für das Zweite, usw.).

Beispiel

Variante 1:

```
Gasoline <- generate.split_and_splitdata.externalvalidation(data = Gasoline, trainvalues.ratio = 0.75)
Gasoline <- plsr.ex(data = Gasoline, validation = "none", ncomp = 10)
Gasoline <- externalvalidation(model = Gasoline)
```

Variante 2:

```
Gasoline <- plsr.ex(data = Gasoline, validation = "none", ncomp = 10)
Gasoline <- externalvalidation(model = Gasoline, testdata = testdata, Ytestdata = Ytestdata)
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen.
In Variante 1 wurde zuerst der Datensatz in einen Trainings- und einen Testdatensatz aufgeteilt. Danach wurde eine Modellberechnung mit den Trainingsdaten ohne Validierung durchgeführt. Da bei der externen Validierung keine Testdaten angegeben wurden, sucht die Funktion nach Testdaten im Gasolinedatensatz und führt die externe Validierung damit durch.
In Variante 2 wurde ein Modell mit dem kompletten Gasolinedatensatz berechnet. Bei der externen Validierung werden neue X-Daten (testdata) und Y-Daten (Ytestdata) als Matrix an die Funktion übergeben und damit die externe Validierung durchgeführt.

1.5.6 generate.segments()

Diese Funktion dient dazu, Segmente für die Kreuzvalidierung zu erstellen. Es wird dann jeweils ein gesamtes Segment, während der Kreuzvalidierung, weggelassen. Normalerweise wird diese Funktion als interne Funktion aufgerufen. Sie kann aber auch vom Nutzer aufgerufen werden und dann das Ergebnis an andere Funktionen über segments.CV übergeben werden.

Parameter

data = NULL (PLSR.Kronsbein Datensatz):

Datensatz, welcher durch eine der Einlesefunktionen erzeugt wurde, übergeben.

segments.CV = NULL (numeric):

Hier kann die gewünschte Anzahl an Segmenten übergeben werden. Wenn repetitions = NULL, dann ist der Standardwert 10. Ansonsten werden so viele Segmente erzeugt, sodass in jedem Segment alle Wiederholungen enthalten sind.

segments.type.CV = NULL (character):

Hier kann der Type angegeben werden, wie die Segmente erzeugt werden sollen. Wenn repetitions = NULL dann ist der Standard „interleaved“, ansonsten „consecutive“.

- „random“ → zufällige Zuordnung der Proben zu den Segmenten
- „consecutive“ → die Reihenfolge der Proben wird beibehalten und immer die nebenaneinander liegenden werden einem Segment zugeordnet
  Bsp.: 1,2,3,4,5,6 in 2 Segmente → Segment 1: 1,2,3; Segment 2: 4,5,6
- „interleaved“ → immer eine Probe wird zum 1. Segment zugeordnet, die nächste zum 2. und so weiter, sodass die Proben gleichmäßig auf die Segmente verteilt werden.
  Bsp.: 1,2,3,4,5,6 in 2 Segmente → Segment 1: 1,3,5; Segment 2: 2,4,6

repetitions = data$data.info$read.repetitions() (numeric):

Hier können die Anzahl an Wiederholungen gleicher Datenpunkte angegeben werden. Dies ist aber nicht notwendig, da die Funktion diese automatisch aus dem angegebenen Wert in der Einlesefunktion ausliest. Es werden immer alle Wiederholungen in ein Segment gepackt.

Beispiel

```
segments <- generate.segments(data = Beispiel, segments.CV = 15, segments.type.CV = "interleaved",
    repetitions = 2)
```

Zusätzliche Informationen:

Bei Beispiel handelt es sich um einen fiktiven PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. Der Datensatz enthält 60 Spektren, wobei für jede Probe 2 Spektren aufgenommen wurde. Der Datensatz soll in 15 Segmente über die Methode „interleaved“ aufgeteilt werden. Dabei wird beachtet, dass die wiederholten Messungen immer in einem Segment bleiben. Eine Aufteilung sieht dann wie in Abbildung 27 dargestellt aus.

segments	list [15]	List of length 15
V1	double [4]	1 2 31 32
V2	double [4]	3 4 33 34
V3	double [4]	5 6 35 36
V4	double [4]	7 8 37 38
V5	double [4]	9 10 39 40
V6	double [4]	11 12 41 42
V7	double [4]	13 14 43 44
V8	double [4]	15 16 45 46
V9	double [4]	17 18 47 48
V10	double [4]	19 20 49 50
V11	double [4]	21 22 51 52
V12	double [4]	23 24 53 54
V13	double [4]	25 26 55 56
V14	double [4]	27 28 57 58
V15	double [4]	29 30 59 60

Abbildung 27: Aufteilung der Segmente durch generate.segments(). In „List of length 15“ stehen die Zusammensetzungen der einzelnen Segmente.

1.5.7 predict.ex()

Diese Funktion dient dazu, Vorhersagen mittels eines durch plsr.ex() berechneten Modells und eines neuen Datensatzes für die X-Variablen zu treffen.

Parameter

model = NULL (PLSR.Kronsbein Datensatz):

Datensatz, bei dem ein Modell durch plsr.ex oder kpls erzeugt wurde, übergeben.

ncomp = NULL (numeric vector):

Die Anzahl an Komponenten des Modells die genutzt werden soll, um die Vorhersage zu treffen.

newdata = NULL (numeric matrix):

Hier kann ein neuer Datensatz als Matrix übergeben werden, für den Vorhersagen getroffen werden sollen. Wichtig ist, dass der neue Datensatz die gleiche Anzahl an Variablen wie der ursprüngliche Trainingsdatensatz besitzt.
Alternativ kann aber auch newdata.source genutzt werden. Wenn keines von Beidem angegeben wird, dann werden die Vorhersagen für die ursprünglichen Trainingsdaten ausgegeben.

newdata.source = NULL (character):

Alternative zu newdata. Hier kann der Dateiname einer .csv Datei übergeben werden, in der der neue Datensatz steht.

centeredY = data$data.info$read.centeredY() (boolean);

Dieser Wert sollte nicht vom Nutzer verändert werden. Wenn TRUE, dann wird die Zentrierung der Y-Daten, in den Einlesefunktionen für die Auswertung, rückgängig gemacht.

savedata.TF = TRUE (boolean):

Dieser Wert sollte nicht auf FALSE gesetzt werden, ansonsten erscheint die Vorhersage nicht in der Auswertung.

which.n.method = NULL (numeric):

Dieser Parameter muss nur beachtet werden, wenn mehrere Modelle berechnet wurden. Dann nutzt diese Funktion automatisch immer das letzte berechnete Modell. Soll die Vorhersage jedoch mit einem vorherigen Modell durchgeführt werden, dann kann hier die

entsprechende Nummer angegeben werden (1 für das erste berechnete Modell, 2 für das Zweite, usw.).

Beispiel

Zuvor durchgeführt:

```
Gasoline <- plsr.ex(data = Gasoline, validation = "none", ncomp = 10)
```

Variante 1:

```
Gasoline <- predict.ex(model = Gasoline, ncomp = 1:10)
```

Variante 2:

```
Gasoline <- predict.ex(model = Gasoline, ncomp = 7, newdata = newXdata)
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. Wie unter „Zuvor durchgeführt“ dargestellt, muss ein Modell berechnet worden sein, damit es möglich ist, Vorhersagen zu treffen.

In Variante 1 werden keine neuen X-Daten über newdata übergeben. Deshalb werden als X-Daten die ursprünglichen X-Daten, welche auch zur Modellberechnung dienten, genutzt. Für die Vorhersagen werden 1 bis 10 Komponenten genutzt.

In Variante 2 werden neue X-Daten als Matrix übergeben. Diese müssen die gleiche Anzahl an Variablen aufweisen, wie der ursprüngliche Datensatz. Die Vorhersage wird mit 7 Komponenten getroffen.

1.6 Auswertefunktionen

Diese Funktionen dienen dazu, die Auswertung der genutzten Methoden durchzuführen. Die wichtigste Funktion hierbei ist evaluation(). Diese Funktion generiert und speichert sehr viel hilfreiche Grafiken und Datensätze ab. Die anderen Funktionen dienen nur dazu, sich gewisse Dinge gezielt anzuschauen.

Aufgabe	**Funktionsaufruf**	**Name im Speicher**	**Quellcodedatei**
Allgemeine Auswertefunktion, alle Auswertungen können hierrüber erledigt werden	evaluation()		evaluation_generalfunction.R
Scoreplot der PCA	PCA.scoreplot()	PCA.calc	evaluation_subfunctions_pca.R
Scoreplot der PCA (eine Komponente gegen die Y-Werte der PLSR)	PCA.scoreplot. singlecomp()	PCA.calc	evaluation_subfunctions_pca.R
Loadingplot der PCA	PCA.loadingplot()	PCA.calc	evaluation_subfunctions_pca.R
Loadingplot der PCA (eine Komponente gegen die Variablen)	PCA.loadingplot. singlecomp()	PCA.calc	evaluation_subfunctions_pca.R
Screeplot der PCA (erklärte Varianz pro Komponente)	PCA.screeplot()	PCA.calc	evaluation_subfunctions_pca.R
Scoreplot der PLSR	PLSR.scoreplot()	plsr.ex	evaluation_subfunctions_plsr.R
Scoreplot der PLSR (eine Komponente gegen die Y-Werte der PLSR)	PLSR.scoreplot. singlecomp()	plsr.ex	evaluation_subfunctions_plsr.R
Loadingplot der PLSR	PLSR.loadingplot()	plsr.ex	evaluation_subfunctions_plsr.R
Loadingplot der PLSR (eine Komponente gegen die Variablen)	PLSR.loadingplot. singlecomp()	plsr.ex	evaluation_subfunctions_plsr.R
Screeplot der PLSR (erklärte Varianz pro Komponente)	PLSR.screeplot()	plsr.ex	evaluation_subfunctions_plsr.R
Predictionplot der PLSR (gemessene gegen vorhergesagte Werte)	PLSR.predictionplot()	plsr.ex	evaluation_subfunctions_plsr.R

Tabelle 4: Übersicht Auswertefunktionen

1.6.1 evaluation()

Diese Funktion dient dazu, die komplette Auswertung aller durchgeführten Funktionen durchzuführen. Dafür werden alle Ergebnisse in einem Ordner abgespeichert. Jede Funktion erhält ihren eigenen Unterordner. In diesen Unterordner werden verschiedene Plots als .jpg Datei, verschiedene Tabellen als .csv Datei und weitere Informationen als .txt Datei abgespeichert. Die Reihenfolge der Unterordner entspricht der Reihenfolge der durchgeführten Methoden.

Parameter

data = NULL (PLSR.Kronsbein Datensatz):

Datensatz, welcher durch eine der Einlesefunktionen erzeugt wurde, übergeben.

last.of.this.method = NULL (character):

Hier kann der Name einer Methode angegeben werden. Für diese wird dann nur die letzte Ausführung ausgewertet. Kann normalerweise ignoriert werden.

directory = NULL (character):

Hier muss der Name eines Verzeichnisses angegeben werden, in dem alle Daten gespeichert werden sollen.

projectname = NULL (character):

Hier kann ein Projektname vergeben werden, welcher dann in die infofile.txt geschrieben wird. Wenn dieser nicht angegeben wurde wird directory als projectname übernommen.

decision.dir.exists = NULL (boolean):

Dieser Wert kann standardmäßig auf NULL belassen werden. Falls dann ein Verzeichnis schon existiert, wird nachgefragt, ob dieses gelöscht werden soll oder die Ausführung der Funktion abgebrochen werden soll. Ist der Wert TRUE, dann wird das Verzeichnis ohne Nachfrage gelöscht, und ist der Wert FALSE, dann wird die Ausführung der Funktion ohne Nachfrage abgebrochen.

which.comp = NULL (numeric vector):

Für welche Komponenten des Modells soll die Auswertung erfolgen. Hierbei sind maximal 25 Komponenten möglich. Wird kein Wert angegeben, dann werden alle verfügbaren Komponenten ausgewertet, solange dies nicht mehr als 25 sind.

save.dataset.TF = TRUE (boolean):

Ist dieser Wert TRUE, dann wird der gesamte Datensatz als .RData Datei abgespeichert, sodass er später zum Beispiel für Vorhersagen wieder geladen werden kann.

Beispiel:

Variante 1:

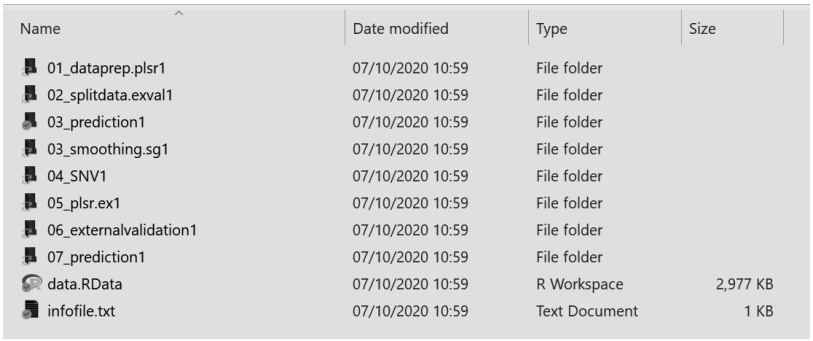
```
evaluation(Gasoline, projectname = "test", directory = "test", which.comp = 1:10)
```

Variante 2:

```
evaluation(Gasoline, directory = "plsr",  which.comp = 1:20, last.of.this.method = "plsr.ex")
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. Danach wurden alle gewünschten Datenvorbehandlungen und PCA- und/oder Modellberechnungen durchgeführt. Damit Variante 2 wie dargestellt funktionieren kann, ist es notwendig vorher ein Modell mittels plsr.ex() zu berechnen.
In Variante 1 wird eine gesamte Auswertung für alle angewendeten Methoden für die ersten 10 Komponenten durchgeführt. Diese wird im Verzeichnis „test“ gespeichert (siehe Abbildung 28).
In Variante 2 wird die Auswertung für die ersten 20 Komponenten der letzten durchgeführten Methode von plsr.ex() durchgeführt. Die Auswertung wird im Verzeichnis „plsr“ gespeichert.



Name	Date modified	Type	Size
01_dataprep.plsr1	07/10/2020 10:59	File folder	
02_splitdata.exval1	07/10/2020 10:59	File folder	
03_prediction1	07/10/2020 10:59	File folder	
03_smoothing.sg1	07/10/2020 10:59	File folder	
04_SNV1	07/10/2020 10:59	File folder	
05_plsr.ex1	07/10/2020 10:59	File folder	
06_externalvalidation1	07/10/2020 10:59	File folder	
07_prediction1	07/10/2020 10:59	File folder	
data.RData	07/10/2020 10:59	R Workspace	2,977 KB
infofile.txt	07/10/2020 10:59	Text Document	1 KB

Abbildung 28: Verzeichnis "test" nach Ausührung von evaluation

1.6.2 allgemeine Parameter für PCA.___plot() und PLSR.___plot() Funktionen

Diese Variablen der Plot-Funktionen sind nebensächlich und werden aber bei den meisten Plot-Funktionen genutzt. Daher werden sie hier zusammengefasst, um Platz zu sparen, Abweichung werden in der jeweiligen Funktion beschrieben.

plotcex = 0.8 (numeric):
Größe der Datenpunkte.

plotpch = 20 (numeric):
Designtyp der Datenpunkte. Für jede Nummer ist ein Designtyp hinterlegt.

textcex = 0.5 (numeric):
Die Größe der Beschriftungen der Datenpunkte.

textpos = 3 (numeric):
Die Position der Beschriftung der Datenpunkte (3 → darüber).

saveplots = FALSE (boolean):
Sollen die Plots als .jpg gespeichert werden.

directory = NULL (character):
Wenn saveplots = TRUE, dann werden sie in dieses Verzeichnis gespeichert. Wenn directory = NULL, dann ins working directory.

plotquality = 100 (numeric):
Die Qualität des Plots in Prozent. 100 am besten.

plotwidth = 1000 oder (numeric):
Die Weite des Plots in Pixel.

plotheigth = 500 (numeric):
Die Höhe der Plots in Pixel.

plotfontsize = 17 (numeric):
Die Schriftgröße der Überschriften im Plot.

col = „black“ (character oder character vector):
Die Farbe der Datenpunkte im Plot. Wird durch colourramp überschrieben. Hier kann auch ein Vektor übergeben werden, um Datenpunkte gezielt einzufärben.

which.n.method = NULL (numeric):
Dieser Parameter muss nur beachtet werden, wenn mehrere PCAs oder PLSRs berechnet wurden. Dann nutzt diese Funktion automatisch immer die letzte berechnete PCA oder PLSR. Soll die Vorhersage jedoch mit einem vorherigen Modell durchgeführt werden, dann kann hier die entsprechende Nummer angegeben werden (1 für die erste berechnete PCA/ PLSR, 2 für die Zweite, usw).

1.6.3 PCA.scoreplot() oder PLSR.scoreplot()

Erzeuge einen „normalen“ Scoreplot bestehend aus zwei Komponenten für die PCA oder PLSR, je nachdem welche Funktion genutzt wurde.

Parameter

data = NULL (PLSR.Kronsbein Datensatz):
Für PCA.scoreplot() einen Datensatz, in dem eine PCA durch PCA.calc() berechnet wurde, übergeben. Für PLSR.scoreplot() einen Datensatz, in dem eine PLSR durch plsr.ex() berechnet wurde, übergeben.

which.comp = c(1,2) (numeric vector):

Hier wird angegeben, welche Komponenten dargestellt werden sollen. Wenn mehr als zwei angegeben wurden, dann werden mehrere Scoreplots erzeugt.

colourramp = TRUE (boolean):

Wenn TRUE, dann werden die Datenpunkte entsprechend der Y-Werte eingefärbt. Für welche Y-Variable dies geschehen soll wird mit Ycol festgelegt.

fixedAxes = FALSE (boolean):

Wenn TRUE, dann besitzen alle Achsen die gleiche Skalierung, sodass sie untereinander besser vergleichbar sind.

labels.sample.name = FALSE (boolean):

Wenn TRUE, dann werden die Namen der Proben als Beschriftung an den Datenpunkten angezeigt.

labels.Yvalue = FALSE (boolean):

Wenn TRUE, dann werden die Y-Werte der Proben als Beschriftung an den Datenpunkten angezeigt. Welche Y-Variable genutzt wird, wird durch Ycol festgelegt.

Ycol = 1 (numeric):

Hier wird die Y-Variable festgelegt, welche für colourramp und labels.Yvalue genutzt werden soll.

saveplots = FALSE (boolean):

Wenn TRUE, dann werden die Plots als .jpg abgespeichert.

directory = NULL (character):

Wenn saveplots = TRUE, dann werden die Plots in dieses Verzeichnis gespeichert. Wenn directory = NULL, dann ins working directory.

plotcex, plotpch, textcex, textpos, plotquality, plotwidth, plotheight, plotfontsize, col und which.n.method siehe (1.6.2 allgemeine Parameter für PCA.___plot() und PLSR.___plot() Funktionen).
Abweichungen: plotheigth = 1000

Beispiel:

PCA:

```
PCA.scoreplot(data = Gasoline, which.comp = c(1,2), colourramp = TRUE, fixedAxes = TRUE, Ycol = 1)
```

PLSR:

```
PLSR.scoreplot(data = Gasoline, which.comp = c(1,2), colourramp = TRUE, fixedAxes = TRUE, Ycol = 1)
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. Für PCA.scoreplot() muss vorher PCA.calc() ausgeführt worden sein. Für PLSR.scoreplot() muss vorher plsr.ex() durchgeführt worden sein.
Für die PCA und die PLSR werden jeweils die Scoreplots für die ersten beiden Komponenten erzeugt. Die Datenpunkte werden entsprechend der Y-Werte einer Y-Variable eingefärbt. Hier wurde Ycol = 1 gesetzt, somit wird die erste Y-Variable genutzt. Durch fixedAxes = TRUE besitzen die Achsen der Plots die gleiche Skalierung.
Die erzeugten Plots der Funktionen sind in Abbildung 29 zu sehen.

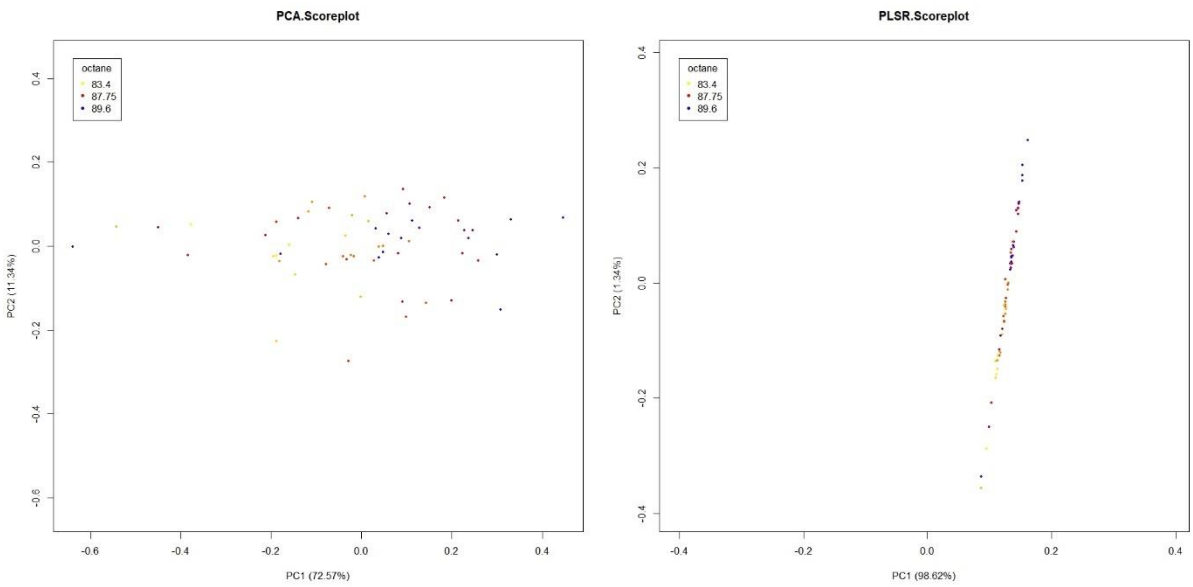


Abbildung 29: Scoreplots der Funktionen PCA.scoreplot()(links) und PLSR.scoreplot()(rechts)

1.6.4 PCA.scoreplot.singlecomp() oder PLSR.scoreplot.singlecomp()

Erzeuge einen Scoreplot, bei dem die Scorewerte gegen den Y-Werte der jeweiligen Proben aufgetragen werden.

Parameter

data = NULL (PLSR.Kronsbein Datensatz):

Für PCA.scoreplot.singlecomp() einen Datensatz, in dem eine PCA durch PCA.calc() berechnet wurde, übergeben. Für PLSR.scoreplot.singlecomp() einen Datensatz, in dem eine PLSR durch plsr.ex() berechnet wurde, übergeben.

which.comp = 1 (numeric vector):

Hier wird angegeben, welche Komponenten dargestellt werden sollen. Wenn mehr als eine angegeben wird, dann werden mehrere Scoreplots erzeugt.

colourramp = FALSE (boolean):

Wenn TRUE, dann werden die Datenpunkte entsprechend der Y-Werte eingefärbt. Für welche Y-Variable dies geschehen soll wird mit Ycol festgelegt.

fixedAxes = FALSE (boolean):

Wenn TRUE, dann besitzen alle Achsen die gleiche Skalierung, sodass sie untereinander besser vergleichbar sind.

labels.sample.name = FALSE (boolean):

Wenn TRUE, dann werden die Namen der Proben als Beschriftung an den Datenpunkten angezeigt.

labels.Yvalue = FALSE (boolean):

Wenn TRUE, dann werden die Y-Werte der Proben als Beschriftung an den Datenpunkten angezeigt. Welche Y-Variable genutzt wird, wird durch Ycol festgelegt.

Ycol = 1 (numeric):

Hier wird die Y-Variable festgelegt, welche für colourramp und labels.Yvalue genutzt werden soll.

saveplots = FALSE (boolean):

Wenn TRUE, dann werden die Plots als .jpg abgespeichert.

directory = NULL (character):

Wenn saveplots = TRUE, dann werden sie in dieses Verzeichnis gespeichert. Wenn directory = NULL, dann ins working directory.

plotcex, plotpch, textcex, textpos, plotquality, plotwidth, plotheight, plotfontsize, col und which.n.method siehe (1.6.2 allgemeine Parameter für PCA.___plot() und PLSR.___plot() Funktionen).

Beispiel:

Beispiel:

PCA:

```
PCA.scoreplot.singlecomp(data = Gasoline, which.comp = 1, fixedAxes = TRUE, Ycol = 1)
```

PLSR:

```
PLSR.scoreplot.singlecomp(data = Gasoline, which.comp = 1, fixedAxes = TRUE, Ycol =1)
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. Für PCA.scoreplot.singlecomp() muss vorher PCA.calc() ausgeführt worden sein. Für PLSR.scoreplot.singlecomp() muss vorher plsr.ex() durchgeführt worden sein.
Für die PCA und die PLSR werden jeweils die Scoreplots für die erste Komponente erzeugt. Diese Komponente wird gegen die Y-Werte einer Y-Variable aufgetragen. Hier wurde Ycol = 1 gesetzt, somit wird die erste Y-Variable genutzt. Durch fixedAxes = TRUE besitzen die Achsen der Plots die gleiche Skalierung.
Die erzeugten Plots der Funktionen sind in Abbildung 30 zu sehen.

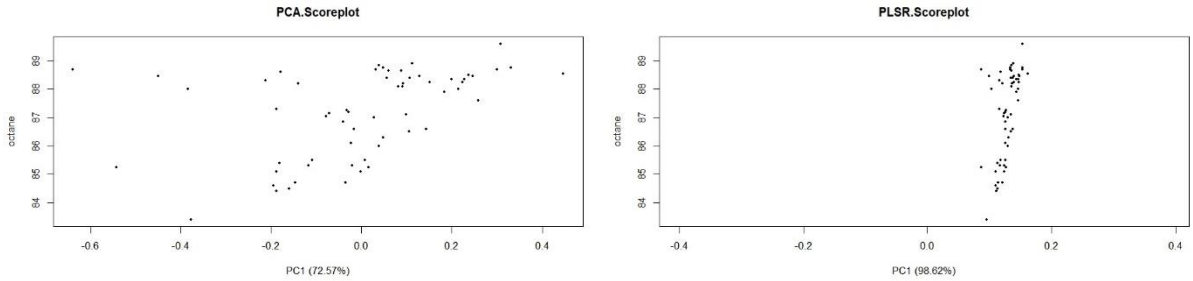


Abbildung 30: Scoreplots der Funktionen PCA.scoreplot.singlecomp()(links) und PLSR.scoreplot.singlecomp()(rechts)

1.6.5 PCA.loadingplot() oder PLSR.loadingplot()

Erzeuge einen „normalen“ Loadingplot bestehend aus zwei Komponenten für die PCA oder PLSR, entsprechend der genutzten Funktion.

Parameter

data = NULL (PLSR.Kronsbein Datensatz):

Für PCA.loadingplot() einen Datensatz, in dem eine PCA durch PCA.calc() berechnet wurde, übergeben. Für PLSR.loadingplot() einen Datensatz, in dem eine PLSR durch plsr.ex() berechnet wurde, übergeben.

which.comp = c(1,2) (numeric vector):

Hier wird angegeben, welche Komponenten dargestellt werden sollen. Wenn mehr als zwei ausgewählt werden, dann werden mehrere Loadingplots erzeugt.

fixedAxes = FALSE (boolean):

Wenn TRUE, dann besitzen alle Achsen die gleiche Skalierung, sodass sie untereinander besser vergleichbar sind.

labels.variable.name = FALSE (boolean):

Wenn TRUE, dann werden die Namen der Variablen als Beschriftung an den Datenpunkten angezeigt.

labels.wavelengths = FALSE (boolean):

Wenn TRUE, dann werden die X-Werte der Spektren (wavelengths-Werte) im Plot als Beschriftung der Datenpunkte angezeigt.

saveplots = FALSE (boolean):

Wenn TRUE, dann werden die Plots als .jpg abgespeichert.

directory = NULL (character):

Wenn saveplots = TRUE, dann werden die Plots in dieses Verzeichnis gespeichert. Wenn directory = NULL, dann ins working directory.

plotcex, plotpch, textcex, textpos, plotquality, plotwidth, plotheight, plotfontsize, col und which.n.method siehe (1.6.2 allgemeine Parameter für PCA.___plot() und PLSR.___plot() Funktionen).
Abweichungen: plotheigth = 1000

Beispiel:

PCA:

```
PCA.loadingplot(data = Gasoline, which.comp = c(1,2), fixedAxes = TRUE, Ycol = 1)
```

PLSR:

```
PLSR.loadingplot (data = Gasoline, which.comp = c(1,2), fixedAxes = TRUE, Ycol =1)
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. Für PCA.loadingplot() muss vorher PCA.calc() ausgeführt worden sein. Für PLSR.loadingplot() muss vorher plsr.ex() durchgeführt worden sein.
Für die PCA und die PLSR werden jeweils die Loadingplots für die ersten beiden Komponenten erzeugt. Durch fixedAxes = TRUE besitzen die Achsen der Plots die gleiche Skalierung.
Die erzeugten Plots der Funktionen sind in Abbildung 31 zu sehen.

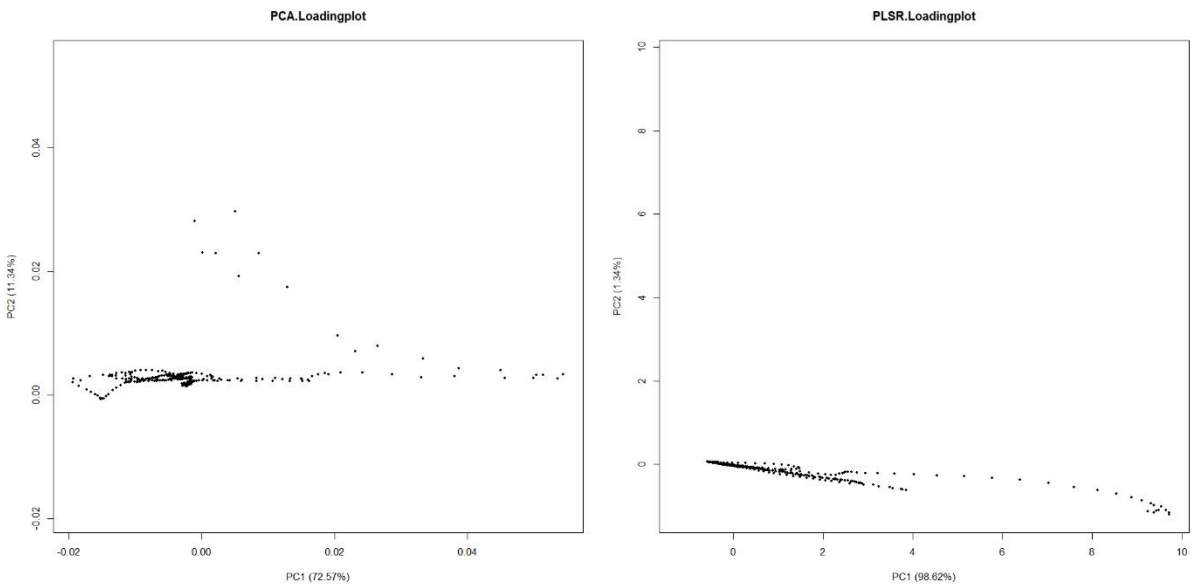


Abbildung 31: Loadingplots der Funktionen PCA.loadingplot()(links) und PLSR.loadingplot()(rechts)

1.6.6 PCA.loadingplot.singlecomp() oder PLSR.loadingplot.singlecomp()

Erzeuge einen loadingplot, bei dem die Loadingwerte gegen den X-Werte der Spektren (Variablen/ wavelengths-Werte) aufgetragen werden.

Parameter

data = NULL (PLSR.Kronsbein Datensatz):

Für PCA.loadingplot.singlecomp() einen Datensatz, in dem eine PCA durch PCA.calc() berechnet wurde, übergeben. Für PLSR.loadingplot.singlecomp() einen Datensatz, in dem eine PLSR durch plsr.ex() berechnet wurde, übergeben.

which.comp = 1 (numeric vector):

Hier wird angegeben, welche Komponenten dargestellt werden sollen. Wenn mehr als eine Komponente ausgewählt wird, dann werden mehrere Loadingplots erzeugt.

fixedAxes = FALSE (boolean):

Wenn TRUE, dann besitzen alle Achsen die gleiche Skalierung, sodass sie untereinander besser vergleichbar sind.

labels.variable.name = FALSE (boolean):

Wenn TRUE, dann werden die Namen der Variablen als Beschriftung an den Datenpunkten angezeigt.

labels.wavelengths = FALSE (boolean):

Wenn TRUE, dann werden die X-Werte der Spektren (wavelengths-Werte) im Plot als Beschriftung der Datenpunkte angezeigt.

saveplots = FALSE (boolean):

Wenn TRUE, dann werden die Plots als .jpg abgespeichert.

directory = NULL (character):

Wenn saveplots = TRUE, dann werden die Plots in dieses Verzeichnis gespeichert. Wenn directory = NULL, dann ins working directory.

plotcex, plotpch, textcex, textpos, plotquality, plotwidth, plotheight, plotfontsize, col und which.n.method siehe (1.6.2 allgemeine Parameter für PCA.___plot() und PLSR.___plot() Funktionen).

Beispiel:

PCA:

PCA.loadingplot.singlecomp(data = Gasoline, which.comp = 1, fixedAxes = TRUE)

PLSR:

PLSR.loadingplot.singlecomp(data = Gasoline, which.comp = 1, fixedAxes = TRUE)

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. Für PCA.loadingplot.singlecomp() muss vorher PCA.calc() ausgeführt worden sein. Für PLSR.loadingplot.singlecomp() muss vorher plsr.ex() durchgeführt worden sein.
Für die PCA und die PLSR werden jeweils die Loadingplots für die erste Komponente erzeugt. Diese Komponente wird gegen die X-Werte der Spektren „wavelengths“ aufgetragen. Durch fixedAxes = TRUE besitzen die Achsen der Plots die gleiche Skalierung.
Die erzeugten Plots der Funktionen sind in Abbildung 32 zu sehen.

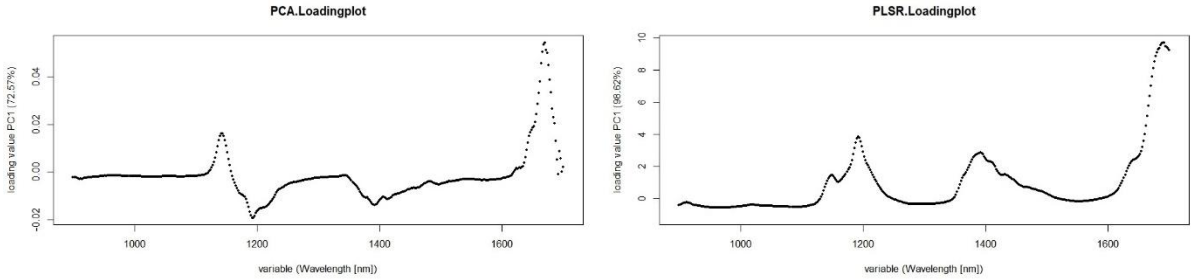


Abbildung 32: Loadingplots der Funktionen PCA.loadingplot.singlecomp()(links) und PLSR.loadingplot.singlecomp()(rechts)

1.6.7 PCA.screeplot() oder PLSR.screeplot()

Dieser Plot dient dazu die erklärte Varianz gegen die Anzahl an Komponenten aufzutragen.

Parameter

data = NULL (PLSR.Kronsbein Datensatz):

Für PCA.screeplot() einen Datensatz, in dem eine PCA durch PCA.calc() berechnet wurde, übergeben. Für PLSR.screeplot() einen Datensatz, in dem eine PLSR durch plsr.ex() berechnet wurde, übergeben.

which.comp = NULL (numeric vector):

Hier wird angegeben, welche Komponenten dargestellt werden sollen.

type = „part.var“ (character):

Dieser Parameter gibt an, wie die erklärte Varianz aufgetragen werden soll.

- „var“ → Die absolute erklärte Varianz pro Komponente wird aufgetragen.
- „part.var“ → Die relative/anteilige erklärte Varianz pro Komponente wird aufgetragen.
- „part.cum.var“ → Die cumulative relative/anteilige erklärte Varianz pro Komponente wird aufgetragen.

saveplots = FALSE (boolean):

Wenn TRUE, dann werden die Plots als .jpg abgespeichert.

directory = NULL (character):

Wenn saveplots = TRUE, dann werden die Plots in dieses Verzeichnis gespeichert. Wenn directory = NULL, dann ins working directory.

plotquality, plotwidth, plotheight, plotfontsize, col und which.n.method siehe (1.6.2 allgemeine Parameter für PCA.___plot() und PLSR.___plot() Funktionen).

Abweichungen: col = „grey“

Beispiel:

PCA:

```
PCA.screeplot(data = Gasoline, which.comp = 1:10, type = "part.var")
```

PLSR:

```
PLSR.screeplot(data = Gasoline, which.comp = 1:10, type = "part.var")
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. Für PCA.screeplot() muss vorher PCA.calc() ausgeführt worden sein. Für PLSR.screeplot() muss vorher plsr.ex() durchgeführt worden sein.
Für die PCA und die PLSR werden jeweils die Screeplots für die ersten 10 Komponenten erzeugt. Da type = „part.var“ gesetzt wurde, wird die relative erklärte Varianz gegen die einzelnen Komponenten aufgetragen.
Die erzeugten Plots der Funktionen sind in Abbildung 33 zu sehen.

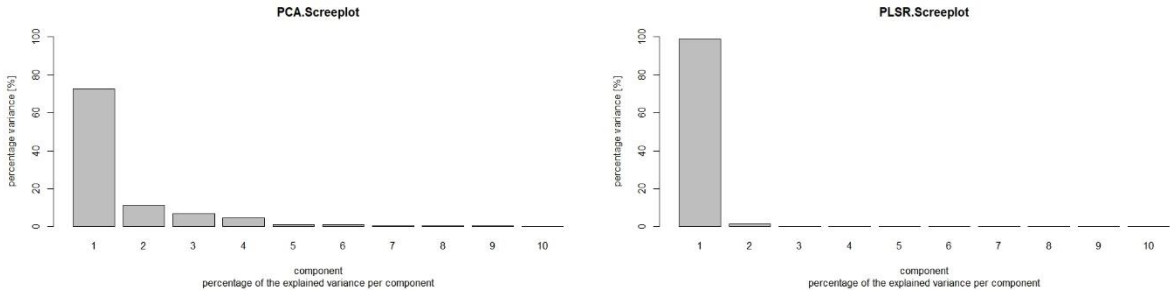


Abbildung 33: Screeplots der Funktionen PCA.screeplot() (links) und PLSR.screeplot()(rechts)

1.6.8 PLSR.predictionplot()

In diesem Plot werden die gemessen Y-Werte gegen die durch das Modell vorhergesagten Werte aufgetragen.

Parameter

data = NULL (PLSR.Kronsbein Datensatz):

Hier einen Datensatz, in dem eine PLSR durch plsr.ex() berechnet wurde, übergeben.

which.comp = NULL (numeric vector):

Hier wird angegeben, welche Komponenten dargestellt werden sollen.

colourramp = FALSE (boolean):

Wenn TRUE, dann werden die Datenpunkte entsprechend der Y-Werte eingefärbt. Für welche Y-Variable dies geschehen soll wird mit Ycol festgelegt.

fixedAxes = TRUE (boolean):

Wenn TRUE, dann besitzen alle Achsen die gleiche Skalierung, sodass sie untereinander besser vergleichbar sind.

labels.sample.name = FALSE (boolean):

Wenn TRUE, dann werden die Namen der Proben als Beschriftung an den Datenpunkten angezeigt.

labels.Yvalue = FALSE (boolean):

Wenn TRUE, dann werden die Y-Werte der Proben als Beschriftung an den Datenpunkten angezeigt. Welche Y-Variable genutzt wird, wird durch Ycol festgelegt.

Ycol = 1 (numeric):

Hier wird die Y-Variable festgelegt, welche für colourramp und labels.Yvalue genutzt werden soll.

saveplots = FALSE (boolean):

Wenn TRUE, dann werden die Plots als .jpg abgespeichert.

directory = NULL (character):

Wenn saveplots = TRUE, dann werden die Plots in dieses Verzeichnis gespeichert. Wenn directory = NULL, dann ins working directory.

plotcex, plotpch, textcex, textpos, plotquality, plotwidth, plotheight, plotfontsize, col und which.n.method siehe (1.6.2 allgemeine Parameter für PCA.___plot() und PLSR.___plot() Funktionen).

Abweichungen: plotheigth = 1000

Beispiel:

PLSR.predictionplot(data = Gasoline, fixedAxes = TRUE, which.comp = 7)

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. Für die Ausführung von PLSR.predictionplot() muss vorher plsr.ex() durchgeführt worden sein. Der die Vorhersage der Werte im predictionplot werden 7 Komponenten verwendet. Durch fixedAxes = TRUE besitzen die Achsen der Plots die gleiche Skalierung.
Der erzeugte Plot der Funktion ist in Abbildung 34 zu sehen.

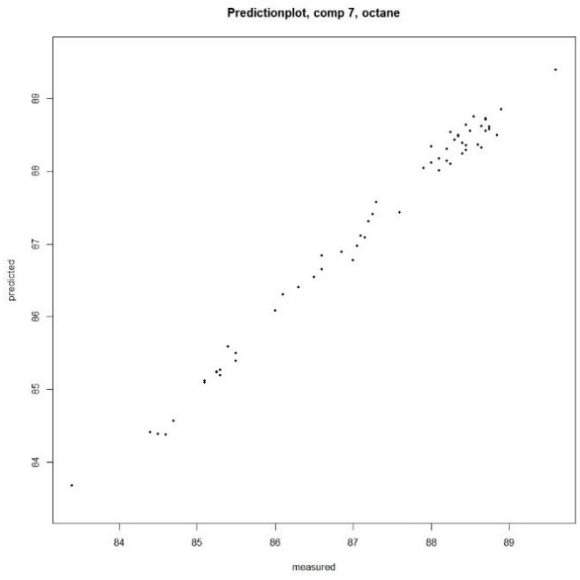


Abbildung 34: Predictionplots der Funktionen PLSR.predictionplot()

1.7 Zusätzliche Funktionen

Unter diese Kategorie fallen die Funktionen, welche nützlich für den Nutzer sind, aber sich keiner anderen Kategorie zuordnen lassen.

Aufgabe	**Funktionsaufruf**	**Quellcodedatei**
Ganz allgemeine Vorbereitungen	gener.prep()	general_functions.R
Einen Datensatz speichern	save.dataset()	general_functions.R
Einen Datensatz laden	load.dataset()	general_functions.R
Die Variablenselektion erzeugt durch variableselection() für später abspeichern	save.variableselection()	variableselection_functions.R
Ein Spektrum anzeigen und dabei Ansicht und Farben auswählen	view.spectra()	spectra_viewer.R
Falsche Variablenzahl nach Spektrenexport aus OPUS korrigieren	correct_number_variables_MPA2()	correct_number_variables.R

Tabelle 5: Übersicht zusätzliche Funktionen

1.7.1 gener.prep()

Diese Funktion dient dazu, ein paar allgemeine Einstellungen im vor hinein zu treffen.
Diese können alternativ auch von Hand gesetzt werden:

```
#Setze die Anzahl der angezeigten Nachkommastellen
options(digits = 4)

 #Setze die maximale Ausgabe länge auf den maximale Wert, sodass alle Ausgaben korrekt getätigt werden.
options(max.print = .Machine$integer.max)
```

Parameter

digits = 4 (numeric):

Die Anzahl der Nachkommastellen festlegen, welche Ausgeben wird.

Beispiel

```
gener.prep(digits = 4)
```

Zusätzliche Informationen:

Diese Funktion sollte am Anfang des Skriptes stehen, nachdem das Package PLSR.Kronsbein aktiviert wurde. Sie bewirkt, dass die Anzahl der ausgegebenen Nachkommastellen auf 4 gesetzt wird und dass die maximale Ausgabelänge der print() Funktion auf den maximalen Wert gesetzt wird.

1.7.2 save.dataset()

Diese Funktion dient dazu, ein Datenset aus R als .RData zu speichern, sodass man dieses zu einem späteren Zeitpunkt wieder mit load.dataset() laden kann.

Parameter

X (PLSR.Kronsbein Datensatz; alle anderen Variablen/Datensätze ebenfalls möglich):

Hier kann der zu speichernde Datensatz übergeben werden. In diesem Fall wird es wahrscheinlich ein PLSR.Kronsbein Datensatz sein. Theoretisch können so aber alle Variablen und Datensätze gespeichert werden.

dir = NULL (character):

Dieser Parameter legt fest, in welchem Verzeichnis der Datensatz gespeichert werden soll. Wenn nichts angegeben wird, dann erfolgt die Speicherung im aktuellen working directory.

ownname = NULL (character):

Hier kann ein eigener Name für die Datei festgelegt werden, ansonsten wird einer automatisch generiert aus dem aktuellen Namen der Variable/Datensatzes.

complete.file = TRUE (character):

Wenn TRUE, dann wird, falls ein Name durch ownname übergeben wurde, an diesen .RData angehängt. Dies geschieht nicht wenn ownname schon die Endung .RData besitzt.

Beispiel:

```
save.dataset(X = Gasoline, dir = "datasets", ownname = "PLSR.1.RData")
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. Danach wurden alle gewünschten Methoden durchgeführt, bevor der aktuelle Datensatz abgespeichert werden soll. Gespeichert wird der Datensatz im Verzeichnis „datasets“, wenn dieses nicht existiert wird es erzeugt. Benannt wird die Datei nach ownname.

1.7.3 load.dataset()

Diese Funktion dient dazu, als .RData abgespeicherte Variablen und Datensätze wieder zu laden. Alternativ kann auch in RStudio eine .RData Datei, über „File → Open File“ geladen werden. Dies ist sinnvoll, wenn später mithilfe eines früher erstellten Modells Vorhersagen getroffen werden sollen.

Parameter

file (character):

Bei diesem Parameter wird der Name der zu ladenden .RData Datei übergeben.

dir = NULL (character):

Dieser Parameter legt fest, aus welchem Verzeichnis der Datensatz geladen werden soll. Wenn nichts angegeben wird, dann erfolgt das Laden aus dem aktuellen working directory.

complete.file = TRUE (boolean):

Wenn TRUE, dann wird an den in file übergebene Dateiname .RData angehängt. Dies geschieht nicht, wenn file schon die Endung .RData besitzt.

Beispiel

Variante 1:

```
load.dataset(file = "PLSR.1.RData", dir = "datasets")
```

Variante 2:

```
Gasoline <- load.dataset(file = "PLSR.1.RData", dir = "datasets")
rm("X")
```

Zusätzliche Informationen:

Für die Funktion load.dataset() muss vorher ein Datensatz über save.dataset() oder evaluation(save.dataset.TF = TRUE) gespeichert worden sein. Der Datensatz wird aus dem Verzeichnis „dataset“ geladen und heißt „PLSR.1.RData“. In Variante 1 wird der Datensatz unter der Variable „X“ geladen. Da dies nicht so schön ist, gibt es noch Variante 2 bei der der Datensatz unter der Variable „Gasoline“ gespeichert wird. Hierbei existiert aber aus programmiertechnischen Gründen noch eine Kopie unter der Variable „X“. Diese Kopie kann über rm(„X“) entfernt werden.

1.7.4 save.variableselection()

Diese Funktion dient dazu eine Variablenauswahl, getroffen durch variableselection(), abzuspeichern, sodass sie später mit load.variableselection() wieder geladen werden kann.

Parameter

data = NULL (PLSR.Kronsbein Datensatz):

Hier einen Datensatz, in dem eine Variablenselektion durchgeführt wurde, übergeben.

selectionalgorithm (character):

Hier muss der zuvor verwendete Selektionsalgorithmus angegeben werden („CARS“, „Procrustes“ oder „PCA.procrustes“).

which.n.method = NULL (numeric):

Dieser Parameter muss nur beachtet werden, wenn mehrere Variablenselektionen mit dem gleichen Algorithmus berechnet wurden. Dann nutzt diese Funktion automatisch immer die letzte berechnete Variablenselektion mit dem in selectionalgorithm angegebene Algorithmus. Soll aber eine vorherige Variablenselektion gespeichert werden, dann kann hier die entsprechende Nummer angegeben werden (1 für die erste berechnete Variablenselektion dieses Algorithmus, 2 für die Zweite, usw.).

directory = NULL (character):

Dieser Parameter legt fest, in welchem Verzeichnis der Datensatz gespeichert werden soll. Wenn nichts angegeben wird, dann erfolgt die Speicherung im aktuellen working directory.

filename = „selectedvariables.RData“ (character):

Hier kann der Name der Datei geändert werden. Sollte er nicht die Endung .RData besitzen wird diese angehängt.

Beispiel

```
Gasoline <- variableselection(data = Gasoline, selectionalgorithm = "CARS", direction = "backwards",
    validation = "CV", break.up = "best.overall", multithread = TRUE, ncomp = 10, segments.CV = 10,
    segments.type.CV = "interleaved")
save.variableselection(data = Gasoline, selectionalgorithm = "CARS", filename = "CARS.selection.RData",
    directory = "test1")
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. Danach wurde eine Variablenselektion durchgeführt, damit diese dann durch save.variableselection gespeichert werden kann. Bei der Speicherung, muss der genutzte Selektionsalgorithmen angegeben werden. Entsprechend der Angaben wird die Variablenselektion unter „CARS.selection.RData“ im Verzeichnis „test1“ gespeichert. Die Variablenselektion kann später durch den Aufruf von load.variableselection geladen und erneut angewandt werden.

1.7.5 view.spectra()

Diese Funktion dient dazu, sich die Spektren im Datensatz näher anzuschauen. Es ist möglich über die Auswahl der Achsenlimitierung in die Spektren hinein zu zoomen. Zusätzlich können die Spektren durch verschiedene Optionen farbig markiert werden. Hierfür gibt es fünf Möglichkeiten. Es ist dabei nicht möglich, mehrere gleichzeitig zu nutzen. Werden mehrere ausgewählt, wird die mit der höheren Priorität genutzt (siehe Parameter). Außerdem können die Spektren auch als .jpg Datei abgespeichert werden. Es ist empfohlen, bei großen Datensätzen auch die Option zu nutzen die Spektren als .jpg abzuspeichern, auch wenn man sich diese nur kurz anschauen möchte. Eine Darstellung im Plots Fenster ist bei großen Datensätzen sehr langsam.

Parameter

data = NULL (PLSR.Kronsbein Datensatz):

Datensatz, welcher durch eine der Einlesefunktionen erzeugt wurde, übergeben.

xlim = c(NA, NA) (numeric vector der Länge 2):

Hier kann der anzuzeigende Bereich der X-Achse ausgewählt werden. Hierfür muss ein Vektor der Länge 2 übergeben werden. Die X-Achse wird auf den Bereich vom ersten bis zum zweiten Wert des Vektors beschränkt. Sollten einer oder beide Werte den Wert NA besitzen, dann werden diese je nachdem auf den minimalen oder maximalen Wert der X-Daten gesetzt.

ylim = c(NA, NA) (numeric vector der Länge 2):

Hier kann der anzuzeigende Bereich der Y-Achse ausgewählt werden. Hierfür muss ein Vektor der Länge 2 übergeben werden. Die Y-Achse wird auf den Bereich vom ersten bis zum zweiten Wert des Vektors beschränkt. Sollten einer oder beide Werte den Wert NA besitzen, dann werden diese je nachdem auf den minimalen oder maximalen Wert der Y-Daten gesetzt.

plottype = „l“ (character):

Hier kann ausgewählt werden, wie die Spektren dargestellt werden.

„l“ → Jedes Spektrum wird als Linie dargestellt

„p“ → Die einzelnen Datenpunkte der Spektren werden dargestellt

„b“ → Es werden die Darstellung von „l“ und „p“ kombiniert

col.l = NULL (character vector):

Hier kann ein Vektor, welcher Farbwerte enthält, übergeben werden. Dieser muss eine Länge entsprechend der Anzahl der Spektren besitzen. Die einzelnen Spektren werden entsprechend der Werte im Vektor eingefärbt. col.l besitzt die viert höchste Priorität der Einfärbungsoptionen.

col.p = NULL (character vector):

Hier kann ein Vektor, welcher Farbwerte enthält, übergeben werden. Dieser muss eine Länge entsprechend der Anzahl an Datenpunkten der einzelnen Spektren besitzen. Diese Option ist nur sinnvoll in Kombination mit plottype = „l“ oder „b“. Die Datenpunkte jedes Spektrums werden entsprechend der Werte im Vektor eingefärbt. col.p besitzt die niedrigste Priorität der Einfärbungsoptionen.

highlight.this.spectra = NULL (numeric vector):

Hier kann ein Vektor übergeben werden, welcher die Reihennummer der Spektren enthält, welche durch eine rote Färbung hervorgehoben werden sollen. highlight.this.spectra besitzt die dritt höchste Priorität der Einfärbungsoptionen.

colourspectra = FALSE (boolean):

Durch diese Option werden alle Spektren eingefärbt. Alle Wiederholungen erhalten die gleiche Farbe. Zu beachten ist aber, wenn zu viele Spektren vorhanden sind, dann erhalten die Spektren irgendwann wieder die gleiche Farbe. Es sind 24 Farben vorhanden. colourspectra besitzt die höchste Priorität der Einfärbungsoptionen.

colourramp = FALSE (boolean):

Durch diese Option werden alle Spektren eingefärbt. Die Färbung erfolgt aufgrund einer Farbskala, welche auf die Werten einer Y-Variable basiert. Die Y-Variable kann über Ycol ausgewählt werden. colourramp besitzt die zweit höchste Priorität der Einfärbungsoptionen.

Ycol = 1 (numeric):

Hier wird die Spalte der Y-Daten festgelegt, welche für colourramp verwendet wird.

p.size = 1 (numeric):

Hier kann die Punktgröße in den dargestellten Spektren verändert werden.

l.size = 1 (numeric):

Hier kann die Linienbreite in den dargestellten Spektren verändert werden.

plotname = „spectra“ (character):

Hier kann die Überschrift des Plots und der Dateiname angepasst werden.

saveplots = FALSE (boolean):

Wenn dieser Wert TRUE ist, dann werden die Plots nicht im Plots-Fenster angezeigt, sondern als .jpg Datei abgespeichert.

directory = NULL (character):

Hier kann das Verzeichnis verändert werden, in das die Spektren gespeichert werden sollen. Wenn der Wert nicht verändert wird, dann wird das Arbeitsverzeichnis genutzt.

resolution = „normal“ (character):

Hier kann die Auflösung der .jpg Datei angepasst werden.

„normal“ → 1000x1000 Pixel
„high“ → 5000x5000 Pixel
„ultrahigh“ → 20000x20000 Pixel

repetitions = data$data.info$read.repetitions() (numeric):

Hier können die Anzahl an Wiederholungen gleicher Datenpunkte angegeben werden. Dies ist aber nicht notwendig, da die Funktion diese automatisch aus dem angegebenen Wert in der Einlesefunktion ausliest.

which.spectra = NULL (numeric vector):

Hier kann ein Vektor übergeben werden, welcher die Reihennummer der Spektren enthält, welche angezeigt werden sollen.

Beispiel

Variante 1:

```
view.spectra(data = Gasoline)
```

Variante 2:

```
view.spectra(data = Gasoline, xlim = c(1350,1500), ylim = c(NA,0.6), saveplots = TRUE,
    plotname = "zoomed in spectra", directory = "spectra")
```

Variante 3:

```
view.spectra(data = Gasoline, xlim = c(1100,1300), ylim = c(0,0.7), l.size = 2, colourramp = TRUE, saveplots = TRUE,
    plotname = "colourramp.spectra", directory = "spectra", resolution = "high")
```

Zusätzliche Informationen:

Bei Gasoline handelt es sich um einen PLSR.Kronsbein Datensatz, erstellt durch eine der Einlesefunktionen. In Variante 1 wird die einfachste Variante der Funktion genutzt, dabei werden alle Spektren im Plots Fenster dargestellt. In Variante 2 wird in das Spektrum über xlim und ylim hereingezoomed. Das erzeugte Spektrum wird im Verzeichnis „spectra“ als .jpg gespeichert.

In Variante 3 wird ebenso in das Spektrum herein gezoomed. Die Spektren werden entsprechend der Y-Variable eingefärbt. Dies geschieht durch colourramp = TRUE. Genutzt wird die erste Y-Variable, dies ist der Standardwert von Ycol. Das erzeugte Spektrum wird im Verzeichnis „spectra“ als .jpg gespeichert. Hierbei wird eine hohe Auflösung genutzt, um besser zwischen den Linien der Spektren unterscheiden zu können. Damit die Linien aber nicht zu fein werden, wurde die Liniendicke noch etwas durch l.size = 2 erhöht.

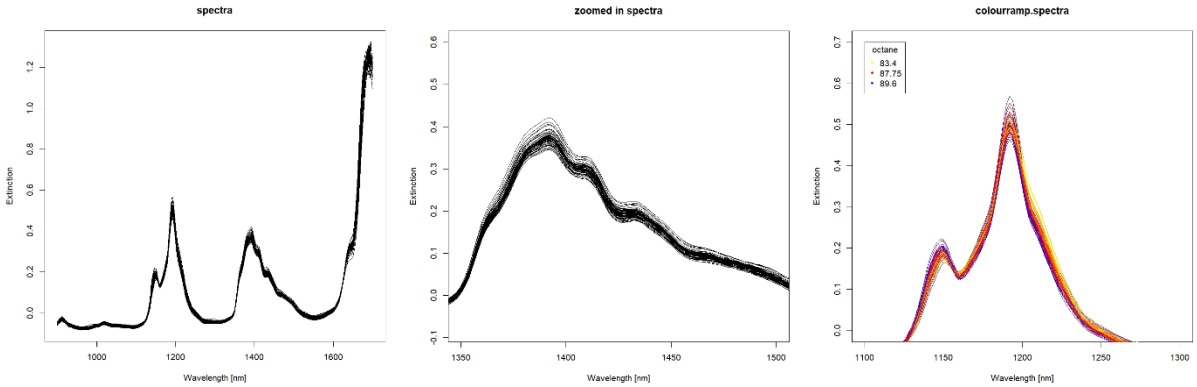


Abbildung 35: Die erzeugten Plots der Spektren durch view.spectra(). Variante 1 ist links dargestellt, Variante 2 in der Mitte und Variante 3 rechts.

1.7.6 correct_number_variables_MPA2()

Diese Funktion dient dazu, einen Bug der OPUS Software zu korrigieren. Manchmal wird bei den Spektren, beim Export als .csv Datei, die Auflösung verändert, obwohl alle Spektren mit gleicher Auflösung aufgenommen wurden. Nun steht man vor dem Problem, dass man nicht mit Spektren unterschiedlicher Variablenzahl in diesem Package arbeiten kann. Um dieses Problem zu beheben, existiert diese Funktion. Sie rechnet die Spektren, welche keine passende Variablenzahl besitzen, auf eine passende Variablenzahl um.

Parameter

dirspectra (character):

Hier wird angegeben, in welchem Verzeichnis die zu korrigierenden Spektren liegen.

wavelengths = NULL(numeric vector):

Wenn keines der Sepktren, welche unter dirspectra liegen, die richtige Auflösung mit den korrekten Wellenlängen besitzt, dann muss hier ein Vektor mit den Wellenlängen übergeben werden.

number_of_variables (numeris):

Hier wird festgelegt, welche Anzahl an Variablen die richtige ist. Wenn Spektren mit einer anderen Anzahl an Variablen vorhanden sind, dann werden sie auf die Anzahl korrigiert.

savefiles = TRUE (boolean):

Wenn dieser Wert TRUE ist, dann werden die Ergebnisse als .csv Datei abgespeichert, sodass sie mit der Funktion fread.dataprep.plsr() wieder eingelesen werden können. Andernfalls wird eine Liste mit den Ergebnissen zurückgegeben.

filename.Xdata = „Xdata.csv“ (character):

Hier kann der Name der Datei für die Spektrenwerte (X-Daten) angepasst werden.

filename.wavelengths = „wavelengths.csv“ (character):

Hier kann der Name der Datei für die Wellenlängen (wavelengths) angepasst werden.

Beispiel:

```
correct_number_variables_MPA2(dirspectra = "Beispiel",number_of_variables = 1899
      ,filename.Xdata = "XdataBeispiel.csv")
```

Zusätzliche Informationen:

Die Spektren in Ordner Beispiel werden auf die Anzahl von 1899 Variablen korrigiert. Hierbei müssen schon richtige Spektren im Ordner vorhanden sein, da wavelengths nicht übergeben wurde. Die Daten werden nach der Korrektur als .csv Dateien gespeichert. Der Name der Datei für die X-Daten wurde geändert.

1.8 Interne Funktionen

Es gibt noch jede Menge interne Funktionen, welche im Regelfall aber nicht vom Nutzer aufgerufen werden, daher werden diese hier nicht näher betrachtet. Für nähere Informationen bitte im Quellcode die Kommentare lesen.

Im Folgenden ist eine grobe Übersicht über die internen Funktionen dargestellt.

Aufgabe	Funktionsaufruf	Quellcodedatei
generelle Zentrierung, Standardisierung und Skalierung der X-Daten	centerX_scaleX_standardizeX()	centerX_scaleX_standardizeX.R
generelle Zentrierung, Standardisierung und Skalierung der Y-Daten	centerY_scaleY_standardizeY()	centerY_scaleY_standardizeY.R
Funktion um in cutspectrum() eine Cut-Matrix in einen logische Vektor zu transformieren	compare.cut()	cut_spectrum.R
Funktion um in cutspectrum() Cuts zu allcuts hinzuzufügen	addcuttoallcuts()	cut_spectrum.R
In Spektrum Variablen auswählen, anhand eines left.variables Vektors. Wird von anderen Funktionen genutzt.	cut.left.variables()	cut_spectrum.R
Für CARS Algorithmus eine Aufteilung für die Daten erstellen	generate.split.CARS()	exvalforCARS.R
Für CARS Algorithmus die Daten für eine externe Validierung aufteilen	splitdata.externalvalidation.CARS()	exvalforCARS.R
Externe Validierung im CARS-Algorithmus durchführen	externalvalidation.CARS()	exvalforCARS.R
Daten einlesen und Fehler abfangen	fread.csv.trycatch()	general_functions.R
Alle Spektren einer Matrix plotten	printplot.allspectrums()	general_functions.R
Berechnung einer inversen Matrix und dabei die Toleranz immer weiter verringern, wenn es nicht funktioniert	solve.try.catch()	general_functions.R
Den Grad der Glättung (degreeofsmoothing) in Glättungs und Ableitungsfunktion überprüfen	false.degreeofsmoothing()	general_functions.R
Das Model in Glättungs und Ableitungsfunktion überprüfen	false.model()	general_functions.R
Überprüfen, ob Daten als Datei oder Matrix an die Funktion übergeben wurden	sourceormatrix()	general_functions.R
apply() durchführen und die Matrix danach drehen	apply.t()	general_functions.R
Die Summe zweier Spektren	add.spectrum()	general_functions.R
Die Differenz zweier Spektren	substract.spectrum()	general_functions.R
Das eine durch das andere Spektrum teilen	divide.spectrum()	general_functions.R
Einen Vector in eine Matrix mit einer Reihe transformieren	as.matrix.byrow()	general_functions.R
Länge eines Vektors oder einer einzeiligen Matrix berechnen, aber nicht einer mehrzeiligen Matrix	length.vector.()	general_functions.R
Funktion, um ein Datenset in Auswertefunktionen zu übergeben oder einzulesen je nach Parametern	check.which.n.method_or_read()	general_functions.R
Gehe in das angegebene Verzeichnis, wenn es nicht existiert erzeuge es	check.set.create.directory()	general_functions.R
Erzeuge eine Farbskala für die Y-Daten, wird in Scoreplots verwendet	Colourramp.sorted.for.Y()	general_functions.R
Überprüfe ob das Datenset grob stimmt	check.data()	general_functions.R
Die maximale Anzahl an Komponenten die durch plsr.ex berechnet werden können ausgeben	calc.max.comp()	general_functions.R
Algorithmus Polynomial baseline Correction	pbc_algo()	polynomial_baseline_correction.R
Savitzky Golay Glättung und Ableitung durchführen	savitzkygolay.smoothing.derivation()	smoothing_derivation_savitzkygolay.R
Algorithmus für apply() Funktion für die Glättung durch den Mittelwert	smo.spect.mean()	smoothing_mean.R
SNV Korrektur für ein einzelnes Spektrum; wird durch apply aufgerufen	SNV.single()	SNV.R
Eine einzelne Wellenzahl [cm^-1] in die Wellenlänge [nm] umrechnen	wn_to_wl()	spect_transformation_UniitspecX.R
Eine einzelne Wellenlänge [nm] in die Wellenzahl [cm^-1] umrechnen	wl_to_wn()	spect_transformation_UniitspecX.R

Die generelle Umrechnung der Wellenzahl und Wellenlänge auf der X-Achse der Spektren	general_change_UnitspecX()	spect_transformation_UnitspecX.R
Einen Einzelnen Extinktionswert in den Transmissionswert umrechnen	E_to_T()	spect_transformation_UnitspecY.R
Einen Einzelnen Transmissionswert in den Extinktionswert umrechnen	T_to_E()	spect_transformation_UnitspecY.R
Die generelle Umrechnung der Extinktion und Transmission auf der Y-Achse der Spektren	general_change_UnitspecY()	spect_transformation_UnitspecY.R
Für einen Datensatz die Vektoren für Trainings und Testdaten generieren	generate.split()	splitdata.R
Den Datensatz anhand der in generate.split() erzeugten Vektoren aufteilen	splitdata.externalvalidation()	splitdata.R
Ein Ausschitt in variableselection() der wiederholt wird, für die plsr in verschiedenen Varianten durch	plsr.RMSEP.variableselection()	variableselection_functions.R
Berechnung von Procrustes	procrustes.own()	variableselection_functions.R
Procrustesvariablenselektionsalgorithmus der durch apply in variableselektion() aufgerufen wird	procrustes.int()	variableselection_functions.R
PCA-Procrustesvariablenselektionsalgorithmus der durch apply in variableselektion() aufgerufen wird	PCA.procrustes.int()	variableselection_functions.R
CARSvariablenselektionsalgorithmus (führt PLSR durch) der durch apply in variableselektion() aufgerufen wird	PLSR.int()	variableselection_functions.R
Jeden einzelnen Wert eines Vectors wiederholen	repeat.each.value.vector	general_functions.R
Erstellen einer ganz allgemeinen Informationsdatei	infofile.eval()	evaluation_generalfunction.R
Alle Einlesefunktionen auswerten	dataprep.plsr.eval	evaluation_subfunctions_dataprep.R
Aufteilung der Daten in Test und Trainingsdatenset auswerten	splitdata.exval.eval()	evaluation_subfunctions_dataprep.R
Ändern der Einheit der Y-Achse im Spektrum auswerten	EtoT_TtoE.eval()	evaluation_subfunctions_dataprep.R
Ändern der Einheit der X-Achse im Spektrum auswerten	wn_to_wl__wl_to_wn.eval()	evaluation_subfunctions_dataprep.R
Beschneiden des Spektrums auswerten	cutspectrum.eval	evaluation_subfunctions_dataprep.R
MSC Korrektur auswerten	MSC.adapted.eval	evaluation_subfunctions_dataprep.R
Polynomial Baseline Correction auswerten	pbc.eval()	evaluation_subfunctions_dataprep.R
Glättungs und Ableitungsfunktionen auswerten	smoothing_derivation.eval()	evaluation_subfunctions_dataprep.R
Zentrieren, Standardisieren und Skalieren der X-Daten auswerten	centerX_scaleX_standardizeX.eval()	evaluation_subfunctions_dataprep.R
Zentrieren, Standardisieren und Skalieren der Y-Daten auswerten	centerY_scaleY_standardizeY.eval()	evaluation_subfunctions_dataprep.R
SNV Korrektur auswerten	SNV.eval()	evaluation_subfunctions_dataprep.R
Variablenselektionsalgorithmen auswerten	variableselection.eval()	evaluation_subfunctions_dataprep.R
Variablenselktion anhand von left.variables Vektor auswerten, wird von anderen Funktionen genutzt	cut.left.variables.eval()	evaluation_subfunctions_dataprep.R
Komplette Auswertung der PCA	PCA.eval()	evaluation_subfunctions_pca.R
Komplette Auswertung der PLSR	PLSR.eval()	evaluation_subfunctions_plsr.R
Zusammenfassung der wichtigsten Ergebnisse der PLSR	summary.eval()	evaluation_subfunctions_plsr.R
Modelkoeffizienten der PLSR als csv	modelcoefficients.eval()	evaluation_subfunctions_plsr.R
Scores und Loadings der PLSR als csv	scoresandloadings.eval()	evaluation_subfunctions_plsr.R
Validationplot der PLSR (RMSEP vs Komponenten)	validationplot.eval()	evaluation_subfunctions_plsr.R
Correlationplot der PLSR	correlationplot.eval()	evaluation_subfunctions_plsr.R
Auswertung der externen Validierung	externalvalidation.eval()	evaluation_subfunctions_plsr.R
Auswertung der Kreuzvalierung (nur teilweise: Segmentauswahl)	crossvalidation.eval()	evaluation_subfunctions_plsr.R
Auswertung der Reduktion an Variablen	reduce_variables_spectra.eval()	evaluation_subfunctions_dataprep.R
Auswertung der Ladefunktion von durchgeführten Variablenselektionen	load.variableselection.eval()	evaluation_subfunctions_dataprep.R
Erzeuge einen Vaktor mit Farben für view.spectra	generate.colourforspectra()	spectra_viewer.R
Auswertung der Funktion zur Berechnung der Mittelwertspektren	calc.meanspectra.eval()	evaluation_subfunctions_plsr.R

Daten für predict.ex bearbeiten, wenn IIR.correction genutzt wurde	IIR.correction.pred()	IIR_correction.R
Auswertung für die IIR Korrektur	IIR.correction.eval()	evaluation_subfunctions_plsr.R
Funktion für apply() in pbc.algo	pbc.apply()	polynomial_baseline_correction.R
Alle Änderungen der Y-Daten rückgängig machen, um die Ursprungsdaten zurück zu erhalten	undo_corrections_Y()	undo_corrections_Y.R
Ein Spektrum mit dem anderen multiplizieren	multiply.spectrum()	general_functions.R
RMSE, BIAS und SE berechnen	parameter.calc()	general_functions.R
Erzeugen einer Kerneltransformationsfunktion	create.kFUN()	kpls.R
Kreuzvalidierungsalgorithmus für KPLS	CV.algo.kpls()	crossvalidation.R
Kreuzvalidierungsalgorithmus	CV.algo()	crossvalidation.R
Vorhersagen mittels KPLS Modell tätigen (interne Funktion von predict.ex())	KPLS.pred()	kpls.R
Auswertung der zusätzlichen Infos für KPLS Modellerstellung	KPLS.eval()	evaluation_subfunctions_plsr.R
Auswertung der Begrenzung des Kalibrierbereichs	reduce.calibrationrange.eval()	evaluation_subfunctions_dataprep.R

Tabelle 6: Übersicht Interne Funktionen

1.9 PLSR.Kronsbein Datensatz

Bei dem PLSR.Kronsbein Datensatz handelt es sich um den Datensatz, mit dem alle Funktionen in diesem Package arbeiten. Dabei handelt es sich um eine Liste (engl. list)(siehe Abbildung 36: Beispiel PLSR.Kronsbein Datensatz), welche verschiedene Objekte enthält. Diese werden im Folgenden näher aufgeschlüsselt. Den Datensatz kann man sich durch folgenden Befehl anzeigen lassen. Der Variablennamen muss an den selbst genutzten Name angepasst werden.

View(Gasoline)

Gasoline	list [8]	List of length 8
prepdata	list [47 x 2] (S3: data.frame)	A data.frame with 47 rows and 2 columns
oriY	list [2]	List of length 2
wavelengths	double [397]	904 906 908 910 912 914 ...
data.info	environment [12] (R6: data.info)	R6 object of class data.info
directorymethoddone	environment [10] (R6: directorym	R6 object of class directorymethoddone
PCA	list [9]	List of length 9
model	list [23] (S3: mvr)	List of length 23
predictions	list [2]	List of length 2

Abbildung 36: Beispiel PLSR.Kronsbein Datensatz

1.9.1 prepdata

Hier werden die aktuellen X und Y Daten gespeichert, diese werden durch die Datenvorbereitungsfunktionen verändert. Mit diesen wird dann durch plsr.ex() das Modell berechnet.

prepdata	list [47 x 2] (S3: data.frame)	A data.frame with 47 rows and 2 columns
Y	double [47 x 1] (S3: AsIs)	-1.877 -1.927 1.273 -3.777 0.723 -1.677 ...
X	double [47 x 397]	-4.18e-02 -3.54e-02 -3.66e-02 -3.85e-02 -4.07e-02

Abbildung 37: Beispiel PLSR.Kronsbein Datensatz prepdata

1.9.2 oriY

In oriY werden die nicht zentrierten Y-Werte und der Mittelwert gespeichert. In prepdata stehen die zentrierten Werte, da sie in den Einlesefunktionen zentriert werden, um Kompatibilität mit der plsr() Funktion aus dem Package pls herzustellen.

oriY	list [2]	List of length 2
Y.values	double [47 x 1]	85.3 85.2 88.5 83.4 87.9 85.5 ...
Y.mean	double [1]	87.18
octane	double [1]	87.18

Abbildung 38: Beispiel PLSR.Kronsbein Datensatz oriY

1.9.3 wavelengths

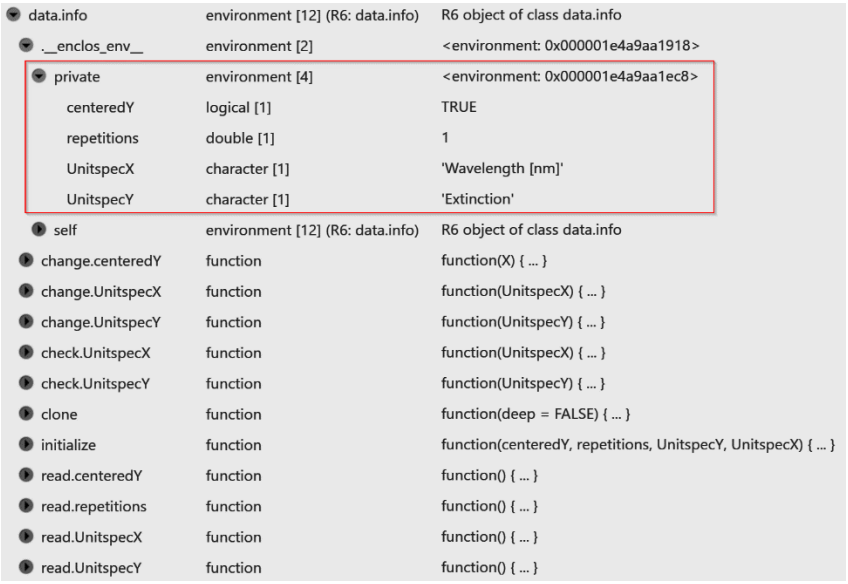
In wavelengths werden die X-Werte der Spektren abgespeichert, welche für die Darstellung der Spektren benötigt werden. Für die Modellberechnung ist wavelengths irrelevant.

wavelengths	double [397]	904 906 908 910 912 914 ...

Abbildung 39: Beispiel PLSR.Kronsbein Datensatz wavelengths

1.9.4 data.info

In data.info wird ein Objekt der Klasse data.info gespeichert. Für nähere Informationen zur Klasse siehe 1.10.2 data.info. Dies dient dazu allgemeine Informationen, wie die Einheit der Achsen, im Spektrum zu speichern. Für den Nutzer ist nur der rot markierte Bereich in Abbildung 40 von Interesse. Der Rest sind interne Funktionen, welche über das Objekt aufgerufen werden können.

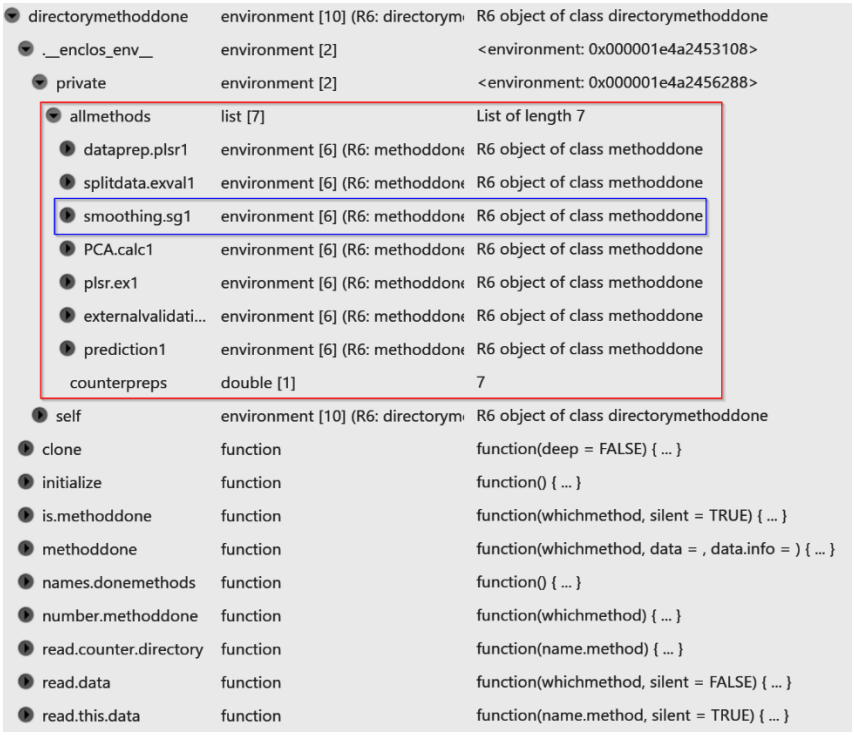


data.info	environment [12] (R6: data.info)	R6 object of class data.info
.__enclos_env__	environment [2]	<environment: 0x000001e4a9aa1918>
private	environment [4]	<environment: 0x000001e4a9aa1ec8>
centeredY	logical [1]	TRUE
repetitions	double [1]	1
UnitspecX	character [1]	'Wavelength [nm]'
UnitspecY	character [1]	'Extinction'
self	environment [12] (R6: data.info)	R6 object of class data.info
change.centeredY	function	function(X) { ... }
change.UnitspecX	function	function(UnitspecX) { ... }
change.UnitspecY	function	function(UnitspecY) { ... }
check.UnitspecX	function	function(UnitspecX) { ... }
check.UnitspecY	function	function(UnitspecY) { ... }
clone	function	function(deep = FALSE) { ... }
initialize	function	function(centeredY, repetitions, UnitspecY, UnitspecX) { ... }
read.centeredY	function	function() { ... }
read.repetitions	function	function() { ... }
read.UnitspecX	function	function() { ... }
read.UnitspecY	function	function() { ... }

Abbildung 40: Beispiel PLSR.Kronsbein Datensatz data.info

1.9.5 directorymethoddone

In directorymethoddone wird ein Objekt der Klasse directorymethoddone gespeichert. Für nähere Informationen zur Klasse siehe 1.10.1 directorymethoddone und methoddone. Dieses Objekt dient dazu, alle durchgeführten Methoden zu verwalten. In Abbildung 41 ist der für den Nutzer interessante Bereich in rot markiert, hier sind alle durchgeführten Methoden aufgelistet. Es handelt sich dabei um Objekte der Klasse methoddone. Für nähere Informationen zur Klasse siehe 1.10.1 directorymethoddone und methoddone. Der Inhalt von smoothing.sg1 ist in Abbildung 42 zu sehen. Hier werden alle relevanten Daten, nach der Durchführung einer Methode gespeichert. Auch eine Kopie von data.info wird angelegt, da sich die Informationen darin verändern können.



directorymethoddone	environment [10] (R6: directorym	R6 object of class directorymethoddone
.__enclos_env__	environment [2]	<environment: 0x000001e4a2453108>
private	environment [2]	<environment: 0x000001e4a2456288>
allmethods	list [7]	List of length 7
dataprep.plsr1	environment [6] (R6: methoddone	R6 object of class methoddone
splitdata.exval1	environment [6] (R6: methoddone	R6 object of class methoddone
smoothing.sg1	environment [6] (R6: methoddone	R6 object of class methoddone
PCA.calc1	environment [6] (R6: methoddone	R6 object of class methoddone
plsr.ex1	environment [6] (R6: methoddone	R6 object of class methoddone
externalvalidati...	environment [6] (R6: methoddone	R6 object of class methoddone
prediction1	environment [6] (R6: methoddone	R6 object of class methoddone
counterpreps	double [1]	7
self	environment [10] (R6: directorym	R6 object of class directorymethoddone
clone	function	function(deep = FALSE) { ... }
initialize	function	function() { ... }
is.methoddone	function	function(whichmethod, silent = TRUE) { ... }
methoddone	function	function(whichmethod, data = , data.info =) { ... }
names.donemethods	function	function() { ... }
number.methoddone	function	function(whichmethod) { ... }
read.counter.directory	function	function(name.method) { ... }
read.data	function	function(whichmethod, silent = FALSE) { ... }
read.this.data	function	function(name.method, silent = TRUE) { ... }

Abbildung 41: Beispiel PLSR.Kronsbein Datensatz directorymethoddone

smoothing.sg1	environment [6] (R6: methoddone)	R6 object of class methoddone
.__enclos_env__	environment [2]	<environment: 0x000001e4a9cbb3a8>
private	environment [3]	<environment: 0x000001e4a9cb7b28>
counterpreps	double [1]	3
data	list [5]	List of length 5
databefor	double [47 x 401] (S3: AsIs)	-0.050193 -0.044227 -0.046867 -0.046705 -0.05085
dataafter	double [47 x 397]	-4.18e-02 -3.54e-02 -3.66e-02 -3.85e-02 -4.07e-02
wavelengthsbefor	double [401]	900 902 904 906 908 910 ...
wavelengthsafter	double [397]	904 906 908 910 912 914 ...
infos	list [4]	List of length 4
degreeofsmoothing	double [1]	5
model	double [1]	2
derivation	double [1]	0
repeatfirstderivation	logical [1]	FALSE
data.info	environment [12] (R6: data.info)	R6 object of class data.info

Abbildung 42: Beispiel PLSR.Kronsbein Datensatz methoddone

1.9.6 PCA

In PCA, werden alle relevanten Informationen der letzten PCA-Berechnung angezeigt. Unter prcomp stehen die Ergebnisse der intern aufgerufenen Funktion prcomp() (stats-Package). Die restlichen Daten wurden durch PCA.calc() berechnet.

PCA	list [9]	List of length 9
prcomp	list [5] (S3: prcomp)	List of length 5
sdev	double [47]	0.2267 0.0815 0.0632 0.0508 0.0229 0.01
rotation	double [397 x 47]	-1.14e-02 -1.24e-02 -1.36e-02 -1.39e-02
center	double [397]	-0.0440 -0.0396 -0.0356 -0.0326 -0.0321
scale	logical [1]	FALSE
x	double [47 x 47]	-2.17e-02 -5.42e-01 -4.48e-01 -3.77e-01
var	double [47]	0.051386 0.006645 0.003990 0.002580 0.
part.var	double [47]	0.77710 0.10049 0.06035 0.03901 0.0079
part.cum.var	double [47]	0.777 0.878 0.938 0.977 0.985 0.989 ...
eigenvectors	double [397 x 47]	-1.14e-02 -1.24e-02 -1.36e-02 -1.39e-02
eigenvalues	double [47]	0.051386 0.006645 0.003990 0.002580 0.
scores	double [47 x 47]	-2.17e-02 -5.42e-01 -4.48e-01 -3.77e-01
loadings	double [397 x 47]	-2.59e-03 -2.82e-03 -3.07e-03 -3.15e-03
wavelengths	double [397]	904 906 908 910 912 914 ...

Abbildung 43: Beispiel PLSR.Kronsbein Datensatz PCA

1.9.7 model

In model werden alle relevanten Informationen der letzten Modellbildung durch plsr.ex() abgespeichert. Dabei werden die meisten Daten durch den internen Aufruf von plsr() (pls Package) erzeugt. Nur var, part.var, part.cum.var, RMSEC, RMSEP.ex (nur bei externer Validierung) werden manuell hinzugefügt.

model	list [23] (S3: mvr)	List of length 23
coefficients	double [397 x 1 x 46]	7.56e-05 6.53e-05 5.49e-05 6.00e-05 5.87e-05 4.92e-05 -6.2
scores	double [47 x 46] (S3: scores)	-0.148674 -0.154360 -0.151190 -0.153458 -0.145589 -0.1509
loadings	double [397 x 46] (S3: loadings)	3.01e-01 2.71e-01 2.44e-01 2.23e-01 2.19e-01 2.32e-01 -2.4
Yscores	double [47 x 46] (S3: scores)	-8.13e-01 -8.35e-01 5.51e-01 -1.64e+00 3.13e-01 -7.27e-01
Yloadings	double [1 x 46] (S3: loadings)	0.433253 6.142457 8.135531 2.024565 0.481232 0.935345 0.3
projection	double [397 x 46]	1.75e-04 1.51e-04 1.27e-04 1.38e-04 1.35e-04 1.13e-04 -1.0
Xmeans	double [397]	0 0 0 0 0 0 ...
Ymeans	double [1]	0
fitted.values	double [47 x 1 x 46]	-0.0644 -0.0669 -0.0655 -0.0665 -0.0631 -0.0654 -0.3499 -2.2
residuals	double [47 x 1 x 46]	-1.81e+00 -1.86e+00 1.34e+00 -3.71e+00 7.86e-01 -1.61e+0
Xvar	double [46]	1.42e+03 3.95e+00 1.91e-01 1.26e-01 1.75e-01 1.28e-02 ...
Xtotvar	double [1]	1427
fit.time	double [1]	0.01
ncomp	double [1]	46
method	character [1]	'simpls'
call	language	plsr(formula = Y ~ X, ncomp = ncomp, data = data$prepdat
terms	formula	Y ~ X
model	list [47 x 2] (S3: data.frame)	A data.frame with 47 rows and 2 columns
var	double [46]	1.42e+03 3.95e+00 1.91e-01 1.26e-01 1.75e-01 1.28e-02 ...
part.var	double [46]	9.97e-01 2.77e-03 1.34e-04 8.83e-05 1.23e-04 8.98e-06 ...
part.cum.var	double [46]	0.997 1.000 1.000 1.000 1.000 1.000 ...
RMSEC	double [46 x 1]	1.534 1.245 0.375 0.231 0.220 0.173 ...
RMSEP.ex	double [46 x 1]	1.459 1.325 0.204 0.223 0.202 0.154 ...

Abbildung 44: Beispiel PLSR.Kronsbein Datensatz model

1.9.8 predictions

In predictions werden die letzten Ergebnisse der predict.ex() Funktion aufgerufen. Unter newdata stehen die neuen X-Daten, wenn welche übergeben werden. Ansonsten werden die X-Daten der Modellbildung genutzt und unter newdata steht NULL. Unter predictions stehen die vorhergesagten Werte.

predictions	list [2]	List of length 2
newdata	NULL	Pairlist of length 0
predictions	double [60 x 1 x 1]	85.3 85.2 88.4 83.4 87.9 85.5 ...

Abbildung 45: Beispiel PLSR.Kronsbein Datensatz predictions

1.10 Klassen und Objekte

In diesem Kapitel werden kurz die in PLSR.Kronsbein genutzten Klassen erläutert. Es handelt sich dabei um R6-Klassen, welche mithilfe des Packages R6 erstellt werden können.

1.10.1 directorymethoddone und methoddone

Bei directorymethoddone handelt es sich um eine Verwaltungklasse von methoddone. Von directorymethoddone wird ein Objekt beim Aufruf einer Einlesefunktion erstellt. Nach Durchführung jeder Methode, bei der eine Auswertung durch evaluation() erfolgen soll, wird in der Liste allmethods (siehe Tabelle 7) ein Objekt von methoddone erstellt. Im Objekt von methoddone werden dann alle Daten und Ergebnisse in data (siehe Tabelle 8) gespeichert. Außerdem wird eine Kopie vom aktuellen Objekt data.info (siehe 1.10.2 data.info) gemacht. counterpreps dient in directorymethoddone dazu, die Gesamtanzahl der Objekte methoddone mitzuzählen und in methoddone dazu, die Methoden durchzunummerieren.

Variablen	Beschreibung
counterpreps (numeric)	hochzählende Variable für alle methoddone Objekte in allmethods
allmethods (list)	Liste aller methoddone Objekte

Tabelle 7: Variablen directorymethoddone

Variablen	Beschreibung
counterpreps (numeric)	Durchnummerierung der erledigten Methoden
data (list)	gespeicherte Daten der Methoden als Liste
data.info (data.info object)	Kopie des aktuellen data.info Objekts

Tabelle 8: Variablen methoddone

1.10.2 data.info

In Objekten von data.info sollen hilfreiche Informationen zum Datensatz verwaltet werden. Welche Information die einzelnen Variablen enthalten lässt sich Tabelle 9 entnehmen.

Variablen	Beschreibung
centeredY (boolean)	Wurde Y in einer Dateneinlesefunktion zentriert
originalY (boolean)	Soll die Auswertung mit den Originalwerten erfolgen?
repetitions (numeric)	Anzahl der Wiederholungen einer Probe im Datenset
UnitspecY (character)	Einheit der Y-Achse der Spektren
UnitspecX (character)	Einheit der X-Achse der Spektren

Tabelle 9: Variablen data.info

1.10.3 Klassenfunktionen

Eine Folgenden ist eine Übersicht über die Funktionen der Klassen dargestellt. Diese werden nicht näher erläutert, da diese nur intern aufgerufen werden und somit für den Nutzer eine untergeordnete Rolle spielen.

1.10.3.1 methoddone

Aufgabe	Funktionsaufruf
Initialisierung eines Objekts	methoddone$new()
Daten (data und data.info) speichern	$set.data()
Alle Variablen auslesen	$get.all()
counterprep auslesen	$get.counterpreps()
Objekt kopieren	$clone

Tabelle 10: Funktionen methoddone

1.10.3.2 directorymethoddone

Aufgabe	Funktionsaufruf
Initialierung eines Objekts	directorymethoddone$new()
Daten eine durchgeführten Methode speichern	$methoddone()
Prüfen ob eine Methode durchgeführt wurde	$is.methoddone()
Wie oft wurde diese Methode durchgeführt?	$number.methoddone()
Daten aller gespeicherten methoddone Objekte eines Typs auslesen	$read.data()
Daten eines gespeicherten methoddonn Objektes auslesen	$read.this.data()
Die counterpreps Nummer eines methoddone Objektes auslesen	$read.counter.directory()
Alle Namen der gespeicherten methoddone Objekte auslesen	$names.donemethods()

Tabelle 11: Funktionen directorymethoddone

1.10.3.3 data.info

Aufgabe	Funktionsaufruf
Initialierung eines Objekts	data.info$new()
Auslesen ob Y zentriert wurde	$read.centeredY()
Ändern ob Y zentriert wurde	$change.centeredY()
Auslesen der Anzahl an Wiederholungen einer Probe	$read.repetitions()
Überprüfen ob Einheit der Y-Achse stimmt	$check.UnitspecY()
Auslesen der Einheit der Y-Achse	$read.UnitspecY()
Ändern der Einheit der Y-Achse	$change.UnitspecY()
Überprüfen ob Einheit der X-Achse stimmt	$check.UnitspecX()
Auslesen der Einheit der X-Achse	$read.UnitspecX()
Ändern der Einheit der X-Achse	$change.UnitspecX()
Ändern der Anzahl an Wiederholungen einer Probe	$change.repetitions
Auslesen ob original Y-Werte genutzt werden sollen Ändern ob original Y-Werte genutzt werden sollen	$read.originalY() $change.originalY()

Tabelle 12: Funktionen data.info

1.11 Beispiele: Vollständige Skripte

Achtung: Die folgenden Beispiele wurden weitestgehend willkürlich ausgewählt und zeigen keine optimale Modellerstellung auf. Sie dienen nur dem Zweck, das prinzipielle Vorgehen, beim Arbeiten mit dem PLSR.Kronsbein Package, aufzuzeigen.

1.11.1 PCA und Auswertung

Dieses Skript dient dazu, eine PCA dieses Datensatzes durchzuführen.

```
setwd("C:/Bachelorarbeit/R/gasoline")
library(PLSR.Kronsbein)
gener.prep()
wavelengths <- seq(from = 900, to = 1700, by = 2)
Gasoline <- dataprep.plsr(X.values = gasoline$NIR, Y.values = matrix(gasoline$octane, ncol = 1),
        wavelengths = wavelengths, UnitspecY = "Extinction", UnitspecX = "Wavelength [nm]",
        colnames.X = paste0("NIR.",wavelengths, "nm"))

Gasoline <- PCA.calc(data = Gasoline, center = TRUE)

evaluation(data = Gasoline, projectname = "test", directory = "test", which.comp = 1:10)
```

Erläuterung:

Dieses Skript liest den Gasoline Datensatz mit all den zugehörigen Angaben ein. Danach wird die Berechnung der PCA durchgeführt. Zum Schluss kommt die Funktion evaluation() zum Einsatz, um eine Auswertung der PCA zu erhalten. Die Auswertung der PCA befindet sich dann im folgenden Verzeichnis "C:/Bachelorarbeit/R/gasoline/test/02_PCA.calc1". In diesem Verzeichnis befinden sich dann Unterordner, um die alle Grafiken und Datensätze zu sortieren. Im Unterordner „Loadingplots“ befinden sich zum Beispiel alle möglichen Kombinationen der angegebenen 10 Komponenten als Loadingplots.

1.11.2 Datenvorbehandlung, PLSR, Auswertung und Speicherung des Datensatzes

Dieses Skript dient dazu, einen Spektrendatensatz mit Datenvorbehandlungsmethoden zu bearbeiten und den Datensatz dann abzuspeichern.

```
setwd("C:/Bachelorarbeit/R/gasoline")library(PLSR.Kronsbein)
gener.prep()
wavelengths <- seq(from = 900, to = 1700, by = 2)
Gasoline <- dataprep.plsr(X.values = gasoline$NIR, Y.values = matrix(gasoline$octane, ncol = 1),
        wavelengths = wavelengths, UnitspecY = "Extinction", UnitspecX = "Wavelength [nm]",
        colnames.X = paste0("NIR.",wavelengths, "nm"))

Gasoline <- polynomial_baseline_correction(data = Gasoline, model = 1)
Gasoline <- SNV(data = Gasoline)
Gasoline <- centerX(data = Gasoline)
Gasoline <- variableselection(data = Gasoline, selectionalgorithm = "PCA.procrustes", direction = "forwards",
        break.up = "interactive", ncomp.PCA.PA = 7 ,centerX = TRUE , n.variables = 200, multithread = TRUE)

evaluation(data = Gasoline, projectname = "test", directory = "test", which.comp = 1:10,
         decision.dir.exists = TRUE, break.up = "interactive")
save.dataset(X = Gasoline, dir = "datasets", ownname = "DatasetforPrediction.RData")
```

Erläuterung:

Dieses Skript liest den Gasoline Datensatz mit all den zugehörigen Angaben ein. Danach erfolgt die Datenvorbehandlung durch eine Untergrundkorrektur, gefolgt von einer SNV-korrektur und der Zentrierung der X-Daten. Außerdem erfolgt eine Variablenselektion durch den PCA.Procrustes Algorithmus. Hierbei wurde eine Vorwärtsselektion bis maximal 200 Variablen gewählt. Abgebrochen wird erst bei 200 ausgewählten Variablen. Danach wird anhand der aufgezeichneten Daten die geschickteste Anzahl an Variablen ausgewählt. In diesem Beispiel wurden 138 Variablen gewählt. Nach der Datenvorbehandlung wird das Modell mithilfe von plsr.ex() erstellt. Hierbei

kommt der SIMPLS Algorithmus zum Einsatz. Zur Validierung wurde eine Kreuzvalidierung mit 15 Segmenten gewählt. Danach wird die evaluation() Funktion aufgerufen, um die Auswertung aller Datenvorbehandlungsfunktionen und der Modellbildung durchzuführen. Zum Schluss wird der Datensatz noch abgespeichert, um ihn später für Vorhersagen nutzen zu können.

1.11.3 Kernel-PLS, Validierung und Auswertung

Dieses Skript dient dazu, eine Kernel-PLS zu berechnen und diese dann über Kreuzvalidierung zu validieren.

```
setwd("D:/OneDrive - bwedu/Uni/08 Bachelorarbeit/R/gasoline")
library(PLSR.Kronsbein)
gener.prep()

wavelengths <- seq(from = 900, to = 1700, by = 2)
Gasoline <- dataprep.plsr(X.values = gasoline$NIR, Y.values = matrix(gasoline$octane, ncol = 1),
        wavelengths = wavelengths, UnitspecY = "Extinction", UnitspecX = "Wavelength [nm]",
        colnames.X = paste0("NIR.",wavelengths, "nm"))
Gasoline <- centerX(data = Gasoline)

Gasoline <- kpls(data = Gasoline, ncomp =10, kfun = "gauss", sigma = 2)

Gasoline <- crossvalidation(model = Gasoline, validation = "LOO")

Gasoline <- evaluation(data = Gasoline, directory = "test")
```

Erläuterung:

Dieses Skript liest den Gasoline Datensatz mit all den zugehörigen Angaben ein. Als Datenvorverarbeitung erfolgt die Zentrierung der X-Daten. Das Modell wird mittels Kernel-PLS erstellt. Hierfür werden die X-Daten in eine Kernelmatrix durch eine Gauß'sche Kernelfunktion transformiert. Als Parameter für die Kernelfunktion wird der Parameter sigma genutzt. Dieser wurde hier auf 2 gesetzt. Anschließend erfolgt die Kreuzvalidierung nach dem leave-one-out Prinzip. Zum Schluss werden noch alle relevanten Daten durch die Funktion evaluation() im Ordner test gespeichert.

1.11.4 Laden eines Datensatzes und Vorhersagen treffen

Dieses Skript dient dazu, mit einem vorher gespeicherten Datensatz mithilfe neuer Spektrendaten Vorhersagen zu treffen.

```
setwd("D:/OneDrive - bwedu/Uni/08 Bachelorarbeit/R/gasoline")
library(PLSR.Kronsbein)
gener.prep()

Gasoline <- load.dataset(file = "DatasetforPrediction.RData", dir = "datasets")
rm("X")

Gasoline <- predict.ex(model = Gasoline, newdata = gasoline$NIR[1:10,])

evaluation(data = Gasoline, last.of.this.method = "prediction", directory = "test.pred", which.comp = 1:10)
```

Erläuterung:

Dieses Skript lädt zuerst den Datensatz, welcher in 1.11.2 Datenvorbehandlung, PLSR, Auswertung und Speicherung des Datensatzes gespeichert wurde. rm(„X") dient dazu, die nicht benötigte Kopie von „Gasoline" zu entfernen. Danach wird eine Vorhersage mithilfe des geladenen Modells getätigt. Als X-Daten, für die die Vorhersage getätigt werden soll, wird hier ein Teil des ursprünglichen Gasoline Datensatzes genutzt, da hier keine anderen Daten zu Verfügung standen. Natürlich können hier auch vollkommen neue Daten eingefügt werden. Zum Schluss wird die evaluation() Funktion aufgerufen, um die Vorhersagen zu speichern. Da hier nur die Ergebnisse der predict.ex() Funktion von Interesse sind, wird evaluation() mit dem Parameter last.of.this.method = „prediction" aufgerufen. Somit werden nur die Daten der Vorhersagefunktion gespeichert. Die anderen Daten, welche im Datensatz stecken, wurden schon bei der Modellerstellung gespeichert.

1.11.5 Auswirkungen der Datenvorbehandlung testen

Dieses Skript dient dazu, die Möglichkeiten des PLSR.Kronsbein Datensatzes aufzuzeigen und wie Verzweigungen mit verschiedenen Variablennamen erzeugt werden können. So ist es möglich, verschiedene Datenvorbehandlungfunktionen zu testen und zu vergleichen.

```
setwd("D:/OneDrive - bwedu/Uni/08 Bachelorarbeit/R/gasoline")
library(PLSR.Kronsbein)
gener.prep()
wavelengths <- seq(from = 900, to = 1700, by = 2)
Gasoline <- dataprep.plsr(X.values = gasoline$NIR, Y.values = matrix(gasoline$octane, ncol = 1),
        wavelengths = wavelengths, UnitspecY = "Extinction", UnitspecX = "Wavelength [nm]",
        colnames.X = paste0("NIR.",wavelengths, "nm"))

Gasoline <- plsr.ex(data = Gasoline, validation = "CV", ncomp = 10, segments.CV = 15)
Gasoline <- smoothspectrums.polynomial(data = Gasoline, degreeofsmoothing = 17, model = 4)
Gasoline <- plsr.ex(data = Gasoline, validation = "CV", ncomp = 10, segments.CV = 15)

Gasoline1 <- variableselection(data = Gasoline, selectionalgorithm = "PCA.procrustes", break.up = "best.overall",
        ncomp.PCA.PA = 7, ncomp = 10, centerX = TRUE)
Gasoline1 <- plsr.ex(data = Gasoline1, validation = "CV", ncomp = 10, segments.CV = 15)

Gasoline2 <- reduce_variables_spectra(data = Gasoline, degreeofreduction = 3)
Gasoline2 <- variableselection(data = Gasoline2, selectionalgorithm = "PCA.procrustes", break.up = "best.overall",
        ncomp.PCA.PA = 7, ncomp = 10, centerX = TRUE)
Gasoline2 <- plsr.ex(data = Gasoline2, validation = "CV", ncomp = 10, segments.CV = 15)

evaluation(data = Gasoline1, projectname = "test1", directory = "test1", which.comp = 1:10, decision.dir.exists =
TRUE)
evaluation(data = Gasoline2, projectname = "test2", directory = "test2", which.comp = 1:10, decision.dir.exists =
TRUE)
```

Erläuterung:

Dieses Skript liest ebenfalls zuerst den Gasoline Datensatz mit all den zugehörigen Angaben ein. Dann wird gleich ein PLS-Modell berechnet, ohne dass die Daten behandelt wurden. Danach werden die Spektren geglättet und wieder ein Modell berechnet. Dies wird alles in der Variable „Gasoline“ gespeichert. So lässt sich zeigen, welche Auswirkung die Glättung des Spektrums auf das Modell hat. Nun wird das Skript verzweigt, indem die Ergebnisse der nächsten Funktionen in zwei verschiedenen Variablen gespeichert werden: „Gasoline1“ und „Gasoline2“. Für „Gasoline1“ wird eine Variablenselektion und die Berechnung des PLS-Modells durchgeführt. Für „Gasoline2“ wird zuerst die Anzahl an Variablen reduziert und dann die Variablenselektion und Modellberechnung durchgeführt. Die Variablenreduktion dient dem Zweck die Variablenselektion deutlich zu beschleunigen. Zum Schluss werden „Gasoline1“ und „Gasoline2“ ausgewertet. So lässt sich der Einfluss der Variablenreduktion auf die Variablenselektion und Modellberechnung zeigen.

1.12 Fehlerbehebung

1.12.1 working directory

Häufig kommt es vor, wenn es zu einem Abbruch innerhalb einer Funktion kommt, dass das working directory innerhalb der Funktion verstellt wurde und durch den Abbruch nicht mehr zurückgestellt werden konnte. Hierfür hilft es das aktuelle working directory mit getwd() auszulesen und mit setwd() zu korrigieren. Als Parameter muss in setwd() das Verzeichnis angegeben werden, auf welches das neue working directory gesetzt werden soll. In das übergeordnete Verzeichnis gelangt man durch setwd(„..“).

1.12.2 sink(), Konsolenausgabe in .txt Datei

Viele Auswertefunktionen geben Text in .txt Dateien aus. Hierfür wird die Konsolenausgabe mithilfe der sink() Funktion in die .txt Datei gespeichert und somit nicht mehr in der Konsole ausgegeben. Dies lässt sich durch sink() ohne Parameter wieder rückgängig machen.

1.12.3 R Neustarten

Bei vielen Fehlern ist es hilfreich, R neu zu starten. Hierfür unter „Session“ „Restart R“ auswählen oder die Tastenkombination Ctrl + Shift + F10 nutzen.

1.12.4 Speicher von R leeren

Wenn falsche oder viele unnötige Daten im Speicher liegen, kann dieser über das Besensymbol im Enviromentreiter geleert werden. Alternativ kann kann der Befehl rm(list = ls()) eingegeben werden.

1.12.5 Richtige Sortierung von Dateien (fread.MPA2.dataprep.plsr())

Wichtig ist, dass die Reihenfolge der X-Daten mit der der Y-Daten übereinstimmt. Beim Nutzen der Funktion fread.MPA2.dataprep.plsr() ist es wichtig, dass die Reihenfolge der .csv Dateien stimmt (siehe Abbildung 46).

	A	B	C
1	**Windows**	**R**	**besser Windows und R**
2	Beispiel.1.csv	Beispiel.1.csv	Beispiel.01.csv
3	Beispiel.2.csv	Beispiel.10.csv	Beispiel.02.csv
4	Beispiel.3.csv	Beispiel.11.csv	Beispiel.03.csv
5	Beispiel.4.csv	Beispiel.12.csv	Beispiel.04.csv
6	Beispiel.5.csv	Beispiel.13.csv	Beispiel.05.csv
7	Beispiel.6.csv	Beispiel.14.csv	Beispiel.06.csv
8	Beispiel.7.csv	Beispiel.15.csv	Beispiel.07.csv
9	Beispiel.8.csv	Beispiel.2.csv	Beispiel.08.csv
10	Beispiel.9.csv	Beispiel.3.csv	Beispiel.09.csv
11	Beispiel.10.csv	Beispiel.4.csv	Beispiel.10.csv
12	Beispiel.11.csv	Beispiel.5.csv	Beispiel.11.csv
13	Beispiel.12.csv	Beispiel.6.csv	Beispiel.12.csv
14	Beispiel.13.csv	Beispiel.7.csv	Beispiel.13.csv
15	Beispiel.14.csv	Beispiel.8.csv	Beispiel.14.csv
16	Beispiel.15.csv	Beispiel.9.csv	Beispiel.15.csv

Abbildung 46: Sortierung von Dateien in Windows 10 und R. In Windows 10 werden Dateien in Explorer anders sortiert, als in R (Siehe Spalten A und B). Besser ist die Benennung in Spalte C, diese wird in Windows 10 und R gleich sortiert.

1.12.6 Keine Konstante Variablenanzahl in den Spektren (fread.MPA2.dataprep.plsr())

Wenn folgender Fehler bei der Ausführung der Funktion fread.MPA2.dataprep.plsr() auftaucht, liegt es häufig daran, dass nicht alle Spektrendateien die gleiche Anzahl an Variablen besitzen. Dieses Problem taucht ab und zu nach dem Export von Spektren aus OPUS aufgrund eines Bugs auf. Durch den Bug stimmt die Auflösung der Spektren nicht überein. Hier muss dann die Auflösung der Spektren korrigiert werden, dies lässt sich mit der Funktion correct_number_variables_MPA2()

bewerkstelligen. Danach können die korrigierten Daten mit der Funktion fread.dataprep.plsr() wieder eingelesen werden.

Fehlermeldung:

```
Error in X.values[i, ] <- as.matrix(fread.csv.trycatch(allfiles[i], silent = TRUE))[,  :
number of items to replace is not a multiple of replacement length
```

1.13 Tipps und Tricks

1.13.1 Autovervollständigung und Parameterauswahl

Durch die Tastenkombination Strg + Leertaste kann die Autovervollständigung in R aufgerufen werden. So kann man sich lange Eingaben ersparen. Außerdem kann durch Strg + Leertaste innerhalb einer Funktion aus allen Parametern ein passender ausgewählt werden. Durch die angezeigte Auswahl kann mit den Pfeiltasten navigiert werden und die gewünschte Auswahl mit Tab bestätigt werden. Damit diese hilfreiche Funktion funktioniert muss zuvor die entsprechende library aktiviert worden sein. In diesem Fall durch library(PLSR.Kronsbein).

1.13.2 Quellcode einer Funktion anzeigen lassen

Um sich den Quellcode einer Funktion anzeigen zu lassen, muss man nur mit STRG + Rechtsklicken auf die Funktion klicken.

1.13.3 Daten und Datensätze anzeigen

Mithilfe der Funktion View() kann man sich Daten anzeigen lassen.
Beispiel: View(Gasoline)
Möchte man nur einen Teil der Daten sehen, zum Beispiel den Inhalt eines Objektes einer Liste, lässt sich dieser durch das $ Symbol auswählen.
Beispiel: View(Gasoline$prepdata$X)

2 Theorie

2.1 FT-NIR

2.1.1 Spektroskopie Grundlagen

2.1.1.1 Das elektromagnetische Spektrum

In Abbildung 47 ist das elektromagnetische Spektrum dargestellt. Der bekannteste Teil ist der für das menschliche Auge sichtbare Teil mit einer Wellenlänge von ca. 400 bis ca. 800nm (Gmelch und Reineke 2019, S. 11). Im kurzwelligeren Bereich als das sichtbare Licht befindet sich der ultraviolette Bereich (UV) und dann die Röntgen und Gammastrahlung. Im langwelligeren Bereich als das sichtbare Licht befindet sich die Infrarot- (IR) und die Mikrowellenstrahlung.

Der IR-Bereich lässt sich noch in nahes (NIR), mittleres (MIR) und fernes IR (FIR) unterteilen. Der FIR-Bereich ist in Abbildung 47 im Terahertz-Bereich mit angegeben und befindet sich im Wellenlängenbereich von 25µm bis 1mm. Der Bereich für die MIR-Strahlung wird mit dem Bereich von 2,5 bis 25µm angegeben. Der für diese Thesis interessante NIR-Bereich bezieht sich auf den Wellenlängenbereich von 800nm bis 2,5µm (Gauglitz und Moore 2014, S. 33).

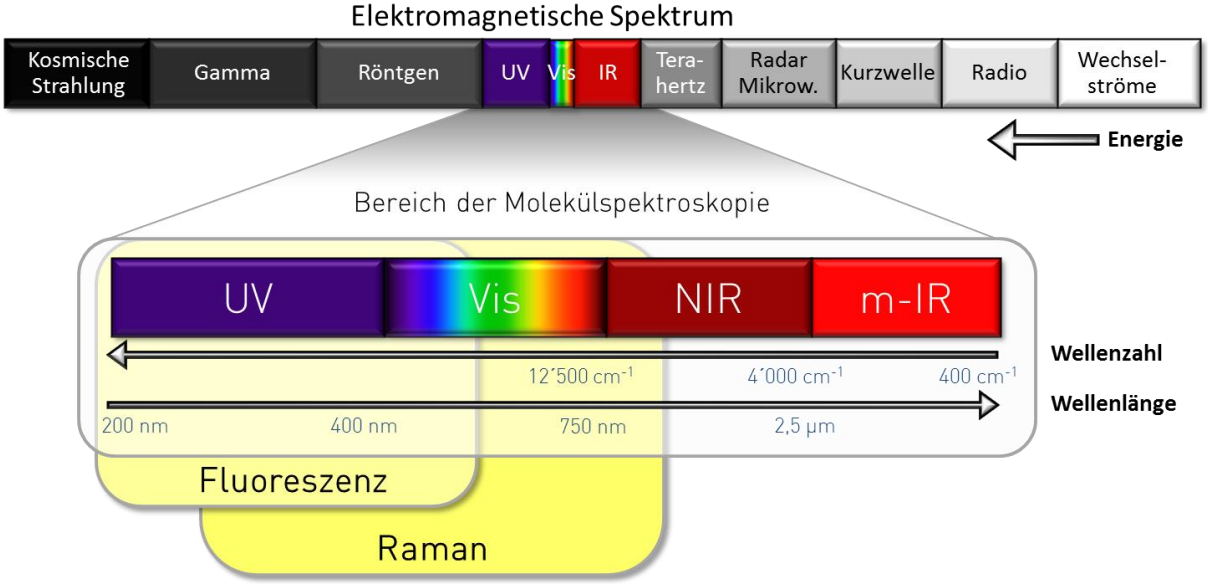


Abbildung 47: Elektromagnetisches Spektrum (Hellma GmbH & Co. KG 2020)

2.1.1.2 Physikalische Zusammenhänge

Wie in Abbildung 47 zu sehen, wird auch die Wellenzahl genutzt, um elektromagnetische Strahlung zu charakterisieren. Die Wellenzahl $\bar{\nu}$ wird meistens in der IR-Spektroskopie und manchmal in der NIR-Spektroskopie genutzt. Die Wellenzahl ist der Kehrwert der Wellenlänge λ und wird in der Regel in
[cm^{-1}] angegeben (Ritgen 2019, S. 233–234).

$$\bar{\nu} = \frac{1}{\lambda} \tag{2.1}$$

Eine weitere Möglichkeit elektromagnetische Strahlung zu charakterisieren ist die Frequenz. Diese berechnet sich, indem die Lichtgeschwindigkeit c durch die Wellenlänge λ geteilt wird und wird in [Hz] oder [1/s] angegeben (Gauglitz und Moore 2014, S. 31).

$$\nu = \frac{c}{\lambda} \tag{2.2}$$

Um die Eigenschaften der Strahlung zu verstehen, ist die Energie E, welche ein Photon besitzt, wichtig. Diese wird berechnet, indem die Frequenz ν der Strahlung mit der Planck Konstante (h = 6,626 *10^{-34}J*s) multipliziert wird (Gauglitz und Moore 2014, S. 31).

$$E = h \cdot \nu \tag{2.3}$$

Das bedeutet, Strahlung mit hoher Frequenz ν ist energiereicher als Strahlung mit niedrigerer Frequenz ν. Dasselbe gilt für die Wellenzahl $\bar{\nu}$, da diese direkt proportional zur Frequenz ν ist. Der Zusammenhang zwischen Wellenlänge λ und Energie E ist umgekehrt proportional und somit ist kurzwellige Strahlung energiereicher als langwellige.

2.1.1.3 Transmissions Spektroskopie

Die meisten spektroskopischen Messungen werden in Transmission gemessen (Gauglitz und Moore 2014, S. 71), diese Technik wird auch in dieser Thesis verwendet, um die Spektren aufzunehmen.

Bei der Transmissions-Spektroskopie wird gemessen, welcher Anteil der Strahlung von einer Substanz oder Lösung absorbiert wird. Hierfür wird, wie in Abbildung 48 gezeigt, Strahlung durch die Probe geschickt I_0, welche dann zu gewissen Teilen absorbiert I_A, durchgelassen I_T, reflektiert $I_R$und gestreut I_S wird. Aufgrund der Energieerhaltung müssen absorbierte I_A, durchgelassene I_T, reflektierte $I_R$und gestreute I_S Strahlung zusammen die gleiche Intensität wie die eingestrahlte Strahlung besitzen I_0 (Gauglitz und Moore 2014, S. 71).

$$I_0 = I_A + I_T + I_R + I_S \tag{2.4}$$

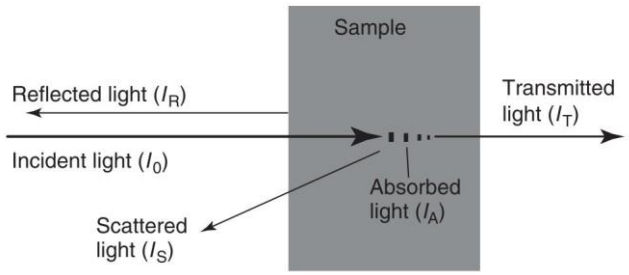


Abbildung 48: Intensitätsbilanz von einstrahlender Strahlung bei der Interaktion mit einer Probe (Gauglitz und Moore 2014, S. 72)

Da es nicht möglich ist den, absorbierten Anteil der Strahlung direkt zu messen, wird die durch die Probe gelassene Strahlung I_T und die eingetragene Strahlung I_0 bestimmt. Messtechnisch ist es nicht sinnvoll, die eingestrahlte Strahlung I_0 direkt zu messen, deshalb misst man zusätzlich eine Leerprobe (Blank) und setzt man I_T der Leerprobe der eingestrahlten Strahlung I_0 gleich (Ritgen 2019, S. 112). So werden außerdem Intensitätsverluste, welche nicht durch die Probe entstehen, beachtet.

Es gibt 2 Messgrößen, welche in der Spektroskopie bestimmt werden. Die Transmission und die Extinktion.
Die Transmission T ist definiert als Anteil der Strahlung, welcher nicht von der Probe absorbiert wird und wird bestimmt, indem die durch die Probe transmittierte Strahlung I_T durch die eingestrahlte Strahlung I_0 geteilt wird (Ritgen 2019, S. 120).

$$T = \frac{I_T}{I_0} \tag{2.5}$$

Die Extinktion E ist definiert als der dekadische Logarithmus der eingestrahlten Strahlung I_0 geteilt durch die durchgelassene Strahlung I_T (Ritgen 2019, S. 120).

$$E = \lg\left(\frac{I_0}{I_T}\right) \tag{2.6}$$

Die Definition der Extinktion ist nicht so anschaulich wie die Transmission, bietet aber den großen Vorteil, dass sich mithilfe des Lambert-Beerschen Gesetzes ein linearer Zusammenhang zwischen Extinktion bei einer Wellenlänge und Probenkonzentration herstellen lässt. Die Extinktion E ist gleich dem Extinktionskoeffizienten ε_λ multipliziert mit der Schichtdicke d, der gemessenen Probe, und der molaren Konzentration c. Der Exktinktionskoeffizient ε_λ ist eine wellenlängenabhängige Stoffkonstante (Ritgen 2019, S. 120). Zu beachten ist aber, dass das Lambert-Beersche Gesetz nur für einen begrenzten Konzentrationsbereich gilt (Gauglitz und Moore 2014, S. 504). Bei einer zu hohen Extinktion/Konzentration ist der Zusammenhang zwischen Konzentration c und Extinktion E nicht mehr linear.

$$E = \varepsilon_\lambda \cdot c \cdot d \tag{2.7}$$

2.1.2 Absorption von MIR-/NIR-Strahlung

Die Photonen der MIR- und NIR-Strahlung liegen in dem Energiebereich, sodass Molekülbindungen zum Schwingen oder Rotieren angeregt werden können (Gauglitz und Moore 2014, S. 32).
Damit es zu einer Absorption kommt, müssen zwei Bedingungen erfüllt sein. Das Photon muss genau die Menge an Energie (siehe Formel (2.3)) besitzen, um eine Schwingung anregen zu können (Ritgen 2019, S. 194) und es muss zu einer Dipolmomentsänderung aufgrund der Schwingung kommen(Hecht 2019, S. 17).

2.1.2.1 Dipolmomentsänderung

Ein Dipol beschreibt den Zustand einer Ladungstrennung, das heißt, dass die Ladungsschwerpunkte von positiver und negativer Ladung nicht zusammenfallen. Die Ladungsschwerpunkte entstehen aufgrund von unterschiedlicher Elektronegativität der Bindungspartner (Riedel 2010, S. 114).
Dies ist zum Beispiel bei Fluorwasserstoff (HF) der Fall. Durch eine Schwingung verändern sich die Ladungsschwerpunkte und damit auch das Dipolmoment.

CO_2 ist ebenfalls IR aktiv, obwohl aufgrund seiner Symmetrie keinen permanenten Dipol besitzt (Riedel 2010, S. 115). Dies liegt daran, dass einige Schwingungen einen Dipol induzieren, da sich dann Ladungsschwerpunkte verschieben und somit kommt es ebenfalls zu einer Dipolmomentsänderung (Hecht 2019, S. 18).

2.1.2.2 Schwingungsarten und Bindungspartner

In Abbildung 49 sind die zwei Arten von Schwingungen dargestellt. Bei Valenzschwingungen kommt es zu einer Änderung der Bindungswinkel und bei Deformationschwingungen zu einer Änderung der Bindungslänge (Hecht 2019, S. 16). Die Schwingungen lassen sich noch weiter unterteilen, was für diese Arbeit jedoch nicht von Bedeutung ist. Die unterschiedlichen Schwingungen benötigen eine unterschiedliche Menge an Energie, um angeregt zu werden, es sei denn, es handelt sich um eine entartete Schwingung.

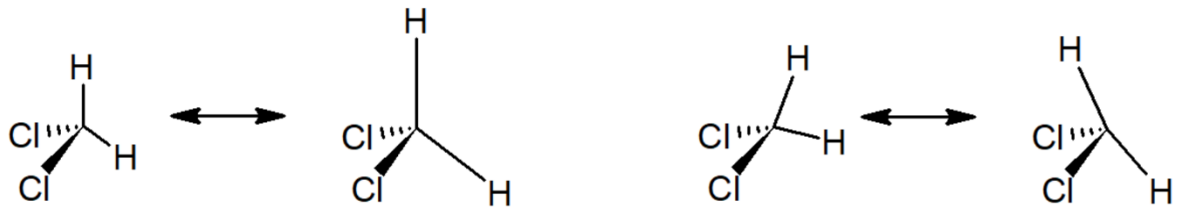


Abbildung 49: links symmetrische Valenzschwingung und rechts „in plane“ Deformationsschwingung

Die Masse der Bindungspartner spielt ebenfalls eine entscheidende Rolle für die benötigte Energie. Dies lässt sich stark vereinfacht veranschaulichen, indem man die Bindung als Feder betrachtet. So gilt, dass die Frequenz ν und damit die Energie (siehe Formel (2.3)) proportional zur Wurzel vom Kehrwert der Masse m ist (siehe Formel (2.8)). Bei D handelt es sich um die Federkonstante. Um mit dieser Gleichung rechnen zu können müsste, man die reduzierte Masse nutzen (Hecht 2019, S. 9), dies wird für die prinzipielle Betrachtung jedoch nicht benötigt.

$$\nu = \frac{1}{2\pi} \cdot \sqrt{\frac{D}{m}} \tag{2.8}$$

2.1.2.3 Anharmonischer Oszillator

Die benötigte Energie hängt ebenfalls davon ab, welcher Schwingungsübergang vollzogen wird. Dies lässt gut am Model des anharmonschen Oszillators erklären (siehe Abbildung 50). Dargestellt sind die verschiedenen Schwingungsenergieniveaus. Ein Übergang vom Schwingungsgrundzustand (n = 0) in den ersten angeregten Zustand (n = 1) wird als Grundschwingung bezeichnet, dies erfolgt üblicherweise durch die Anregung mit IR-Strahlung. Bei ausreichender Energie der Strahlung kommt es zu der Anregung der sogenannten Oberschwingungen, dabei handelt es sich um die Anhebung des Schwingungszustandes, um mehr als ein Niveau (Hecht 2019, S. 13). Im NIR-Spektrum sind hauptsächlich Oberschwingungen und Kombinationsschwingungen zu finden. Diese sind immer schwächer als Grundschwingungen. Normalerweise ist dies ein Nachteil, kann aber bei stark absorbierenden oder dicken Schichtdicken in der Prozessanalytik oder Qualitätskontrolle von Vorteil sein (Gauglitz und Moore 2014, S. 34). Bei Kombinationsschwingungen handelt es sich um die gleichzeitige Anregung zweier Normalschwingungen (Hecht 2019, S. 19).

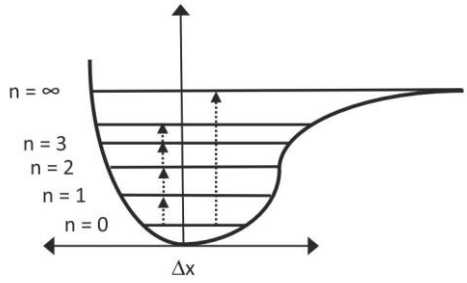


Abbildung 50: Anharmonischer Oszillator (Hecht 2019, S. 13)

Zusammengefasst hängt die benötigte Energie von der Art der Schwingung, der Bindungspartner und ob es sich um eine Grund-, Ober oder Kombinationsschwingung handelt, ab. So kommen die vielen verschiedenen Banden zustande.

2.1.3 FT-NIR Technik

Bei der Fourier Transformation Near Infrared Radiation (FT-NIR) Spektroskopie wird ein Einstrahlphotometer, wie in Abbildung 51 dargestellt, genutzt. Als Strahlungsquelle dient meistens eine Wolfram-Halogenlampe (Gauglitz und Moore 2014, S. 46). Wichtig für die FT-Technik ist das Interferometer, in Abbildung 51 ist ein Michelson Interferometer dargestellt. Dieses besteht aus einem Strahlteiler (CaF_2) und zwei flachen Spiegeln (Gauglitz und Moore 2014, S. 41). Das Interferometer erzeugt das benötigte Interferogram durch Aufteilen und wieder Zusammenführen der Strahlung. Aufgrund der unterschiedlichen optischen Weglänge der zwei Strahlengänge kommt es bei der Zusammenführung zur Interferenz. In Abbildung 51 ist die Messung in Transmission dargestellt, hierfür geht der Strahlengang nach dem Interferometer durch die Probe oder die Reference hindurch und führt dann zum Detektor. Der Detektor detektiert die nicht absorbierte Strahlung. Der Vorteil der FT-Technik besteht darin, dass nicht das Spektrum durchgescannt werden

muss, sondern das gesamte Spektrum auf einmal gemessen werden kann. Dies ist möglich, da aus dem aufgenommenen Interferogram durch Fourier Transformation ein Spektrum, wie man es aus der „klassischen“ NIR-Technik kennt, errechnet werden kann (Gauglitz und Moore 2014, S. 42–43).

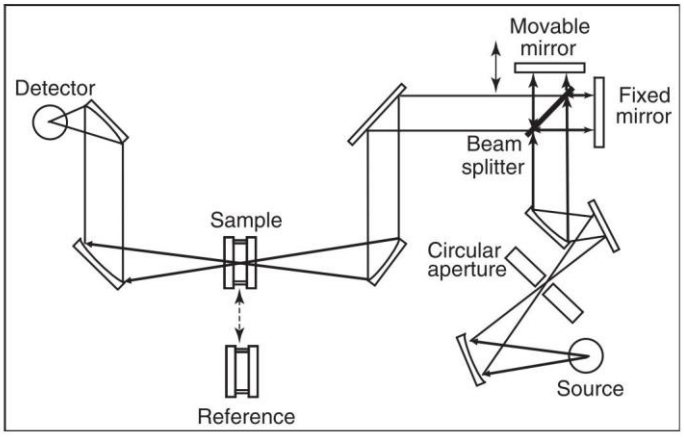


Abbildung 51: FT-NIR Spektrometer (schematische Darstellung) (Gauglitz und Moore 2014, S. 41)

2.2 multivariate Datenanalyse

2.2.1 PCA

Eine bekannte multivariate Analysenmethode ist die Hauptkomponentenanalyse (PCA), welche dazu dient, viele Variablen zu wenigen Hauptkomponenten (PCs) zusammenzufassen, somit werden die Daten „übersichtlicher“ und es ist leichter, Zusammenhänge zu erkennen (Kessler 2008, S. 24).

Die grundlegenden Zusammenhänge der PCA werden in Abbildung 52 dargestellt. Aus den Ausgangsdaten X werden eine Scorematrix T, die Loadings P und die Restvarianz E berechnet.

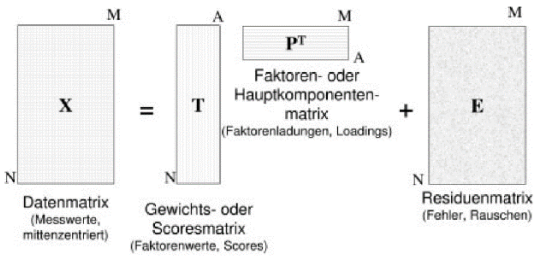


Abbildung 52: Matrizen der Hauptkomponentenanalyse (Kessler 2008, S. 36)

$$X = TP^T + E \tag{2.9}$$

Dies geschieht, indem zuerst eine Kovarianzmatrix aus den Ausgangsdaten X berechnet wird. Aus dieser Kovarianzmatrix lassen sich dann die Eigenwerte und Eigenvektoren berechnen. Aus den Eigenwerten und Eigenvektoren ergeben sich die Faktoren. Alle Faktoren zusammen ergeben die Faktorenmatrix P. In dieser werden die einzelnen Elemente engl. loadings genannt (Kessler 2008, S. 22–23). In Abbildung 53 sind die loadings für die 1. und 2. Hauptkomponente eines Beispieldatensatzes (Rohdaten siehe Anhang 1) dargestellt.

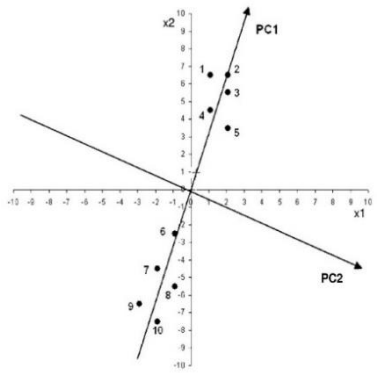


Abbildung 53: Hauptkomponenten im Koordinatensystem (Kessler 2008, S. 32)

Zu sehen ist, dass durch die 1. Hauptkomponente ein Großteil der Gesamtvarianz der Datenpunkte erklärt wird. Die Hauptkomponenten sind Linearkombinationen der Variablen, welche die Variation in den Daten maximieren. Die 2. Hauptkomponente erklärt den Rest der Varianz, sie steht orthogonal zur 1. Hauptkomponente und erklärt am meisten der restlichen Varianz (Siswadi et al. 2012, S. 858). Im Beispiel gibt es maximal zwei Hauptkomponenten, da der Ausgangsdatensatz nur aus zwei Variablen besteht.

Die berechneten Hauptkomponenten stellen nun ein neues Koordinatensystem dar. Die ursprünglichen Daten müssen nun auf dieses neue Koordinatensystem projiziert (umgerechnet) werden (Kessler 2008, S. 23). Die auf Hauptkomponenten umgerechneten Daten nennt man Scores

und stehen in der Scorematrix T (Kessler 2008, S. 23). In Abbildung 54 ist der Scoreplot für die Bespieldaten dargestellt.

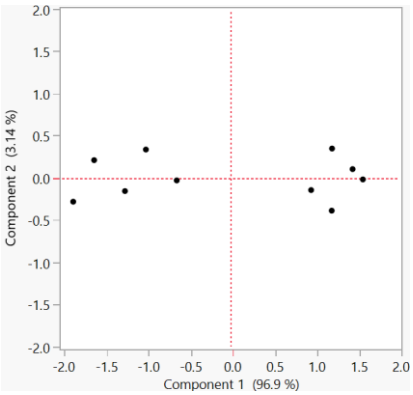


Abbildung 54: Scoreplot

Die Restvarianz E enthält den Teil der Varianz, welcher nicht durch die Hauptkomponenten erklärt wird, genau gesagt die Differenz der Ausgangsdaten X und der durch Faktoren T und Scores P reproduzierten X'-Datenmatrix (siehe (2.10)). Der Term E verschwindet, wenn alle möglichen Hauptkomponenten berechnet werden. Die gesamte Varianz wird in diesem Fall erklärt.

$$E = X - X' = X - TP^T \tag{2.10}$$

Da das Ziel der PCA die Dimensionsreduktion ist, werden alle höheren Hauptkomponenten weggelassen, sobald ein ausreichender Teil der Varianz durch die ersten Hauptkomponenten erklärt wird (Siswadi et al. 2012, S. 858). Die ersten Hauptkomponenten sollten alle relevanten Informationen enthalten, in den höheren Komponenten ist normalerweise nur noch Rauschen enthalten. Welcher erklärte Anteil der Varianz als ausreichend angenommen wird, hängt vom Anwendungsfall ab und liegt häufig zwischen 70 und 90% (Siswadi et al. 2012, S. 858).

2.2.2 PCR

Mithilfe der PCA lassen sich qualitative Aussagen treffen, es ist aber nicht möglich, quantitative Vorhersagen zu tätigen. Hierfür wird die Principle Component Regression benötigt (PCR), welche die PCA mit der multiplen linearen Regression (MLR) kombiniert (Kessler 2008, S. 103).

Die MLR wird verwendet, um einen linearen Zusammenhang zwischen mehreren X-Variablen und einer Y-Variable herzustellen, um das Modell (2.11) zu erstellen (Kessler 2008, S. 99). Die Formel (2.12) beschreibt den gleichen Zusammenhang, wie die Formel (2.11), dargestellt als Matrixschreibweise.

$$y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n + e \tag{2.11}$$

$$y = bX + e \tag{2.12}$$

Ziel der MLR ist es, die Regressionskoeffizienten b_0 bis b_n so zu bestimmen, dass die quadrierten Residuen e (der Fehlerterm) minimal werden. Hierfür wird das Least Square-Verfahren angewendet (Kessler 2008, S. 99–100).

$$b = (X^TX)^{-1}X^Ty \tag{2.13}$$

Die MLR ist für kollineare Datensätze wie Spektren ungeeignet, außerdem muss die Anzahl der Spektren größer als die Anzahl der Variablen (Datenpunkte eines Spektrums) sein, was häufig unpraktikabel ist (Kessler 2008, S. 103).

Bei der PCR werden jetzt anstatt der Originalvariablen die Scores der ersten Hauptkomponenten genutzt. Die Scores der höheren Hauptkomponenten werden weggelassen, da diese meist nur Rauschen enthalten und nicht viel zur Gesamtvarianz beitragen (Kessler 2008, S. 103). Beim Vergleich der Formeln (2.12) und (2.14) fällt auf, dass bei der PCR die Ursprungsdatenmatrix X gegen die Scorematrix T getauscht wurde. Die Regressionskoeffizienten besitzen in Formel (2.14) ein anderes Formelzeichen q. T lässt sich durch XP ersetzen, wodurch der Zusammenhang mit den Ursprungsdaten X über die Loadings P gezeigt werden kann (2.15) (Kessler 2008, S. 104). Und indem man die Loadings P und die Regressionskoeffizienten q miteinander multipliziert, erhält man die Regressionskoeffizienten b für die Ursprungsdaten X (2.16). Das berechnete Modell ähnelt dem Modell der MLR in der Form.

$$y = Tq + e \quad (2.14)$$
$$y = XPq + e \quad (2.15)$$
$$y = Xb + e \quad (2.16)$$

Obwohl das berechnete Model die gleiche Form hat, können die Probleme der MLR mit der PCR umgangen werden (Kessler 2008, S. 103). Kollineare Variablen werden durch die PCA zu einer Hauptkomponente zusammengefasst und es müssen nicht mehr so viele Variablen regressiert werden, da nur noch die Hauptkomponenten gegen Y regressiert werden.

2.2.3 PLSR

Die Partial Least Square Regression (PLSR) ist der PCR sehr ähnlich. Bei der PCR werden die Hauptkomponenten so berechnet, dass ein möglichst großer Teil der X-Varianz erklärt wird. Bei der PLSR wird zusätzlich die Korrelation zwischen X- und Y-Daten beachtet (Wang et al. 2015, S. 390). Somit erfolgt die Berechnung der Komponenten nicht nur auf Grundlage der X-Daten, sondern auch die Y-Daten werden mit einbezogen. Daraus ergibt sich der Vorteil der PLSR, dass weniger Komponenten benötigt werden, um ein vergleichbar gutes Model zu berechnen (Kessler 2008, S. 113).

Unterschieden wird bei der PLSR zwischen der PLS1 und PLS2. Bei der PLS1 gibt es nur eine Responsevariable. Bei der PLS2 sind mehrere möglich. Die PLS1 kann immer als Sonderfall der PLS2 betrachtet werden (Kessler 2008, S. 113), deshalb werden im folgendem die PLS2 Algorithmen dargestellt. Es ist auch möglich, mehrere PLS1 Modelle für mehrere Responsevariablen, anstatt einem PLS2 Modell, zu berechnen (Kessler 2008, S. 129–130). Ein PLS2 Modell kann von Vorteil sein, wenn die Responsevariablen stark miteinander korrelieren. In diesem Fall lässt sich durch ein PLS2 Modell Zeit sparen und die PLS2 kann besser mit großen Varianzen in der Responsevariable umgehen (Kessler 2008, S. 130). Mehrere PLS1 Modelle sind vorteilhaft bei unkorrelierten Responsevariablen, hier kann das Modell individueller auf die einzelnen Responsevariablen eingehen (Kessler 2008, S. 130).

2.2.3.1 NIPALS-Algorithmus

Es gibt verschiedene Algorithmen, um eine PLS-Regression durchzuführen. Der erste bekannte Algorithmus ist der „Nonlinear Iterative Partial Least Square“ (NIPALS) Algorithmus, welcher 1974 von Herman Wold veröffentlicht wurde (Kessler 2008, S. 112). Der NIPALS Algorithmus wird auch orthogonal scores Algorithmus genannt (Bjern-Helge Mevik 2020).

Im folgendem wird ein NIPALS-PLS2 Algorithmus dargestellt und zum Großteil von hier (Kessler 2008, S. 127–129) übernommen. Dieser wurde etwas in der Darstellung angepasst und zwei Stellen korrigiert.

Die X und Y-Datenmatrix werden als erstes mittenzentriert und die Variablen festgelegt. a ist eine Laufvariable für die einzelnen Komponenten. u_a ist am Anfang eine einzelne Y-Variable. Man kann hier irgendeine Variable nutzen. Besser ist es jedoch, diese zu nutzen, bei der der Vektor den größten Betrag aufweist.

$$a = 1 \quad (2.17)$$
$$X_a = X \quad (2.18)$$
$$Y_a = Y \quad (2.19)$$
$$u_a = max|Y_i| \text{ oder } u_a = Y_1 \quad (2.20)$$

Nun werden die gewichteten Loadings w_a berechnet und auf den Betrag 1 normiert. Die Formel für die Normierung wurde im Vergleich zur Quelle geändert (Kessler 2008, S. 127–129). Die gewichteten Loadings drücken den Zusammenhang zwischen den X und Y-Daten aus und beschreiben das „lokale PLS-Modell“ (Kessler 2008, S. 115).

$$w_a = {X_a}^T \cdot u_a \quad (2.21)$$
$$w_a = \frac{w_a}{\sqrt{{w_a}^T \cdot w_a}} \text{ (in Quelle: } w_a = \frac{w_a}{\sqrt{w_a \cdot {w_a}^T}}) \quad (2.22)$$

Für das „lokale Modell“ werden nun die Scores t_a berechnet (Kessler 2008, S. 115)

$$t_a = X_a \cdot w_a \quad (2.23)$$

Im nächsten Schritt werden die p-Loadings p_a berechnet. In diesen stecken die Informationen der X-Daten (Kessler 2008, S. 115).

$$p_a = \frac{{X_a}^T \cdot t_a}{{t_a}^T \cdot t_a} \quad (2.24)$$

Danach folgt die Berechnung der q-Loadings q_a. In diesen stecken die Informationen der Y-Daten (Kessler 2008, S. 115). Die Formel wurde im Vergleich zur Quelle verändert, man erhält so die transponierte q_a Matrix, so konnte der Algorithmus vereinfacht werden.

$$q_a = \frac{{Y_a}^T \cdot t_a}{{t_a}^T \cdot t_a} \text{(in Quelle: } q_a = \frac{{t_a}^T \cdot Y_a}{{t_a}^T \cdot t_a}) \quad (2.25)$$

Nun folgt der Konvergenztest, von diesem hängt das weitere Vorgehen ab. Bei der ersten Berechnung von $conv$ für jede Komponente wird die Formel (2.26) verwendet. Ist die Summe von $conv$ kleiner als ein Referenzwert (meist 10^{-6}) dann ist die Konvergenz erreicht und man fährt mit Formel (2.29) fort. Ist die Konvergenz nicht erreicht, dann wird u_a mit der Formel (2.28) neu berechnet und die in den Formeln (2.21) bis (2.25) beschriebenen Schritte werden wiederholt. Ist dies geschehen, wird der Konvergenztest erneut durchgeführt. Ab der zweiten Iteration wird der $conv$ immer mit Formel (2.27) berechnet. Dieses Vorgehen wird so oft wiederholt bis die Konvergenz erreicht wird.

$$conv = |t_a - u_a| \quad (2.26)$$
$$conv = |u_a - u_{a-1}| \quad (2.27)$$
$$u_a = \frac{{Y_a}^T \cdot q_a}{{q_a}^T \cdot q_a} \quad (2.28)$$

Jetzt wurde die a-te Komponente berechnet. Die in dieser Komponente enthaltenen Informationen werden nun aus der X und Y-Datenmatrix entfernt und die die Laufvariable a wird um 1 erhöht. Nun

kann die nächste Komponente berechnet werden, hierfür springt man wieder zu Formel (2.20) zurück.

$$X_{a+1} = X_a - t_a \cdot {p_a}^T \quad (2.29)$$
$$Y_{a+1} = Y_a - t_a \cdot {q_a}^T \quad (2.30)$$
$$a = a + 1 \quad (2.31)$$

Wurde die maximal mögliche oder die gewünschte Anzahl an Komponenten berechnet, bleiben die Fehlermatrizen E und F übrig und die Berechnung der Komponenten ist abgeschlossen.

$$E = X_{\max(a)+1} \quad (2.32)$$
$$F = Y_{\max(a)+1} \quad (2.33)$$

Zum Schluss müssen noch die Regressionskoeffizienten berechnet werden. In den Formeln (2.21) bis (2.24) wurden die Vektoren w_a, p_a, q_a mit den Loadings für die einzelnen Komponenten berechnet. Für die Berechnung der Regressionskoeffizienten B_k werden die entsprechenden Matrizen W_k, P_k, Q_k benötigt. Diese Matrizen W_k, P_k, Q_k besitzen genau die Anzahl an Spalten, wie Komponenten k genutzt werden, um die Regressionskoeffizienten zu berechnen. Wie man die richtige Anzahl an Komponenten auswählt, wird in 2.2.6.4 Bestimmung der Anzahl an Komponenten beschrieben. Jede Spalte der Matrizen W_k, P_k, Q_k enthalten die entsprechenden Vektoren w_a, p_a, q_a. Zum Beispiel in der 1. Spalte von W_k den Vektor w_1 und so weiter. Der Y-Achsenabschnitt b_0 wird mit Hilfe der Mittelwert der Regressionskoeffizienten B_k und der ursprünglichen X- und Y-Datenmatrix berechnet.

$$B_k = W_k \cdot \left({P_k}^T W_k\right)^{-1} \cdot {Q_k}^T \quad (2.34)$$
$$b_{0,k} = \bar{y}^T - \bar{x}^T B_k \quad (2.35)$$

Mit dem fertig berechneten Modell lassen sich Vorhersagen treffen. Hierfür werden aus einem neue X-Datensatz x_i und den Regressionskoeffizienten B_k und $b_{0,k}$ die Y-Daten $y_{i,k}$ berechnet.

$$y_{i,k} = b_{0,k} + {x_i}^T \cdot B_k \quad (2.36)$$

2.2.3.2 SIMPLS-Algorithmus

Ein weiterer bekannter Algorithmus ist der „Straightforward implementation of a statistically inspired modification of the partial least squares" (SIMPLS) Algorithmus (Alin und Agostinelli 2017, S. 1). Der SIMPLS Algorithmus ist effizienter als der NIPALS Algorithmus, kurz gesagt er benötigt weniger Rechenzeit. SIMPLS und NIPALS liefern beide als PLS1 gleiche Ergebnisse und als PLS2 ähnliche Ergebnisse (TER BRAAK und DE JONG. 1998, S. 42).

2.2.4 Kernel-PLS

Die PLS ist eine lineare Regressionsmethode. Es kann aber vorkommen, dass nicht lineare Zusammenhänge modelliert werden sollen. Eine mögliche Lösung im multivariaten Raum hierfür ist die Kernel-PLS. Bei der Kernel-PLS handelt es sich um eine Abwandlung der „normalen" PLS. Dabei wird die Berechnung der PLS ausgehend von einem sogenannten Kernel und der Y Daten ausgeführt. Dabei ist es möglich, den Kernel zu transformieren, so werden die X-Daten in einen nichtlinearen Raum transferiert. Durch die Transformation in den nichtlinearen Raum steigt die Wahrscheinlichkeit, dass sich diese durch eine lineare PLS modellieren lassen (Botre et al. 2016, S. 214).

2.2.4.1 Kernelfunktionen

Die Transformation in den nicht linearen Raum wird Kernel-Trick genannt. Es gibt verschiedene Kernelfunktionen, durch welche die Transformation erreicht werden kann. Die folgenden Funktionen

wurden aus dem R-Package „kernlab“ übernommen (Karatzoglou et al. 2019, S. 9) und von folgender Webseite (César Souza 2010).

X_1 und X_2 sind die Ausgangsmatrizen für den Kernel. X_1 besitzt i Zeilen und v Spalten. Jede Zeile ist eine Probe und jede Spalte eine Variable. X_2 besitzt j Zeilen und ebenso v Spalten. Wieder ist jede Zeile eine Probe und jede Spalte eine Variable. Es kann sich je nach Fall bei X_1 und X_2 um die gleiche Matrix oder um verschiedene Matrizen handeln. $K(X_1, X_2)$ ist die Kernelmatrix, welche man aus den Kernelfunktionen erhält. Die Kernelmatrix besitzt i Reihen und j Spalten. Bei wenigen Kernelfunktionen ist es möglich, die Kernelmatrix direkt durch Matrizenrechnung zu berechnen (siehe Tabelle 13: Kernelfunktionen (Matrizenrechnung)). Zusätzlich ist es bei allen Kernelfunktionen möglich, die einzelnen Werte der Kernelmatrix durch Vektorrechnung zu berechnen (siehe Tabelle 14: Kernelfunktion (Vektorrechnung)). Hierbei wird immer ein Element der Kernelmatrix in der i-ten Zeile und der j-ten Spalte berechnet. Hierfür wird der Vektor der i-ten Zeile aus der Matrix X_1 und der Vektor der j-ten Zeile aus der Matrix X_2 verwendet. Die Formeln in den beiden Tabellen enthalten zusätzlich noch einige Parameter, welche in Tabelle 15 beschrieben sind.

lineare Kernelfunktion	$K(X_1, X_2) = X_1 \cdot {X_2}^T$	(2.37)
polynomische Kernelfunktion	$K(X_1, X_2) = \left(\alpha \cdot \left(X_1 \cdot {X_2}^T\right) + c\right)^d$	(2.38)
hyperbolische Tangens Kernelfunktion	$K(X_1, X_2) = tanh\left(\alpha \cdot \left(X_1 \cdot {X_2}^T\right) + c\right)$	(2.39)

Tabelle 13: Kernelfunktionen (Matrizenrechnung)

lineare Kernelfunktion	$K(X_1, X_2)_{i,j} = X_{1,i} \cdot {X_{2,j}}^T$	(2.40)
polynomische Kernelfunktion	$K(X_1, X_2)_{i,j} = \left(\alpha \cdot \left(X_{1,i} \cdot {X_{2,j}}^T\right) + c\right)^d$	(2.41)
hyperbolische Tangens Kernelfunktion	$K(X_1, X_2)_{i,j} = tanh\left(\alpha \cdot \left(X_{1,i} \cdot {X_{2,j}}^T\right) + c\right)$	(2.42)
radiale Kernelfunktion, Gaußfunktion	$K(X_1, X_2)_{i,j} = e^{\left(-\sigma \left\|X_{1,i} - X_{2,j}\right\|^2\right)}$	(2.43)
Laplace Kernelfunktion	$K(X_1, X_2)_{i,j} = e^{\left(-\sigma \left\|X_{1,i} - X_{2,j}\right\|\right)}$	(2.44)
Bessel Kernelfunktion	$K(X_1, X_2)_{i,j} = \left(-Bessel^d_{(v+1)}\, \sigma \left\|X_{1,i} - X_{2,j}\right\|^2\right)$	(2.45)
ANOVA Kernelfunktion	$K(X_1, X_2)_{i,j} = \sum_{k=1}^{n} e^{\left(-\sigma \left({X_{1,i}}^k - {X_{2,j}}^k\right)^2\right)^d}$	(2.46)
Spline Kerne	$K(X_1, X_2)_{i,j} = 1 + X_{1,i} \cdot X_{2,j} + X_{1,i} \cdot X_{2,j}\, min\left(X_{1,i}, X_{2,j}\right) - \frac{X_{1,i} \cdot X_{2,j}}{2} \left(min\left(X_{1,i}, X_{2,j}\right)\right)^2 + \frac{\left(min\left(X_{1,i}, X_{2,j}\right)\right)^3}{3}$	(2.47)

Tabelle 14: Kernelfunktion (Vektorrechnung)

α	Skalierung (scale)
c	Versatz (offset)

d	Grad der Funktion (degree)
σ	Kernweite (sigma)
υ	Die Ordnung der Besselfunktion (order)

Tabelle 15: Parameter Kernelfunktion

2.2.4.2 Modellbildung

Die folgende Modellbildung wurde hauptsächlich aus folgendem Paper (Wang et al. 2015) entnommen und so dargestellt, wie auch im R-Package PLSR.Kronsbein (siehe 1.5.3 kpls()) gerechnet wird.

Zuerst muss aus der X-Datenmatrix mit n Reihen und m Spalten eine Kernelmatrix berechnet werden. Dies geschieht mithilfe einer der Kernelfunktion (siehe 2.2.4.1 Kernelfunktionen). Dabei sind X_1 und X_2 jeweils die X-Datenmatrix. Die erhaltene Kernelmatrix besitzt dann n Reihen und Spalten.

Eine Kernel-PLS berechnet aus einem Kernel der linearen Kernelfunktion, verhält sich wie eine normale PLS (César Souza 2010), da hier auch keine Transformation stattfindet. Bei den anderen Kernelfunktion, welche in Tabelle 13 und Tabelle 14 dargestellt sind, findet eine Transformation statt, sodass man ein anderes Modell erhält. Die richtige Transformation für ein gutes Modell muss durch Austesten herausgefunden werden.

Nach der Berechnung der Kernelmatrix K sollte die Zentrierung erfolgen. Hierfür wird eine Matrix N benötigt, welche dieselben Dimension wie K aufweist und überall den Wert $1/n$ enthält. Die Berechnung der zentrierten Kernelmatrix erfolgt dann nach Formel (2.48).

$$K.cent = K - (N \cdot K) - (K \cdot N) + (N \cdot K \cdot N) \tag{2.48}$$

Zuerst müssen die Variablen für den folgenden Algorithmus festgelegt werden. a ist eine Laufvariable für die einzelnen Komponenten. u_a ist am Anfang eine einzelne Y-Variable. Man kann hier irgendeine Y-Variable nutzen. Besser ist jedoch diese Y-Variable zu nutzen, bei der der Vektor den größten Betrag aufweist.

$$a = 1 \tag{2.49}$$
$$K_a = K.cent \tag{2.50}$$
$$Y_a = Y \tag{2.51}$$
$$u_{old} = max|Y_i| \text{ oder } u_{old} = Y_1 \tag{2.52}$$

Bei den folgenden Berechnungen wird iterativ vorgegangen.
Die Scores t_a werden berechnet und dann normiert. Die doppelten Betragsstriche stellen die Länge des Vektors dar.

$$t_a = K_a \cdot u_{old} \tag{2.53}$$
$$t_a = \frac{t_a}{\|t_a\|} \tag{2.54}$$

Nun können die Y-Loadings berechnet werden.

$$q_a = Y_A \cdot {t_a}^T \tag{2.55}$$

Jetzt müssen noch die Y-Scores berechnet und normiert werden.

$$u_a = Y_A \cdot {q_a}^T \tag{2.56}$$
$$u_a = \frac{u_a}{\|u_a\|} \tag{2.57}$$

Nun kann der Konvergenztest durchgeführt werden.

$$crit = \frac{\|u_a - u_{old}\|}{\|u_{old}\|} \tag{2.58}$$

Ist der $crit$ Wert größer als 10^{-9}, dann muss noch eine weitere Iteration durchgeführt werden. u_a wird zu u_{old} und die Berechnung beginnt wieder bei Formel (2.53).

Ist der $crit$ Wert kleiner gleich 10^{-9} dann ist der Konvergenztest bestanden und die a-te Komponente wurde berechnet. a kann um 1 erhöht werden. Nun muss die in der Komponente enthaltene Information aus K_a und Y_a entfernt werden. Hierfür wird die Einheitsmatrix I_n der Größe n benötigt. Eine Einheitsmatrix ist eine quadratische Matrix, bei der in der Diagonalen nur Einsen und ansonsten Nullen stehen. Mit den geänderten Matrizen kann die nächste Komponente beginnend bei Formel (2.52) berechnet werden.

$$K_{a+1} = (I_n - t_a \cdot {t_a}^T) \cdot K_a \cdot (I_n - t_a \cdot {t_a}^T) \tag{2.59}$$
$$Y_{a+1} = Y_a - t_a \cdot {t_a}^T \cdot Y_a \tag{2.60}$$
$$a = a + 1 \tag{2.61}$$

Wurde die maximal mögliche oder gewünschte Anzahl an Komponenten berechnet, dann bleiben die Fehlermatrizen E und F übrig und die Berechnung der Komponenten ist abgeschlossen.

$$E = K_{\max(a)+1} \tag{2.62}$$
$$F = Y_{\max(a)+1} \tag{2.63}$$

Zum Schluss müssen noch die Regressionskoeffizienten berechnet werden. Im oben beschriebenen Algorithmus wurden die Vektoren t_a, u_a, q_a für die einzelnen Komponenten berechnet. Für die Berechnung der Regressionskoeffizienten B_k werden die entsprechenden Matrizen T_k, U_k, Q_k benötigt. Diese Matrizen T_k, U_k, Q_k besitzen genau die Anzahl an Spalten, wie Komponenten k genutzt werden, um die Regressionskoeffizienten zu berechnen. Wie man die richtige Anzahl an Komponenten auswählt, wird in 2.2.6.4 Bestimmung der Anzahl an Komponenten beschrieben. Jede Spalte der Matrizen T_k, U_k, Q_k enthalten die entsprechenden Vektoren t_a, u_a, q_a. Zum Beispiel in der 1. Spalte von T_k den Vektor t_1 und so weiter.

$$B_k = U_k \cdot \left({T_k}^T \cdot K.cent \cdot U_k\right)^{-1} \cdot \left({T_k}^T \cdot Y\right) \tag{2.64}$$

2.2.4.3 Vorhersagen treffen

Eine Vorhersage wird für eine neue X-Datenmatrix $X.new$ getroffen. $X.new$ besitzt $n.new$ Reihen und m Spalten. Nun muss aus $X.new$ eine Kernelmatrix $K.new$ berechnet werden. Hierfür wird ebenfalls die originale X-Datenmatrix X benötigt. X_1 in der Kernelfunktion ist jetzt $X.new$ und X_2 ist X.

Die neue Kernelmatrix sollte ebenfalls zentriert werden. Hierfür werden die Matrizen K und N aus der Kalibrierung und eine neue Matrix $N.new$ benötigt. $N.new$ besitzt die gleichen Werte wie N, also $1/n$, aber $n.new$ Reihen und n Spalten. Die zentrierte Kernelmatrix $K.new.cent$ kann nach folgender Formel berechnet werden.

$$K.new.cent = K.new - (N.new \cdot K) - (K.new \cdot N) + (N.new \cdot K \cdot N) \tag{2.65}$$

Die Vorhersagen $Y.pred$ werden mit folgender Formel berechnet.

$$Y.pred = K.new.cent \cdot B_k \tag{2.66}$$

2.2.5 PLS-DA

Die Partial Least Square Discriminants Analyse (PLS-DA) ist ein Klassifizierungsalgorithmus auf Basis der „klassischen“ PLS Regressionsmethode. Anstatt der Referenzwerte enthält die Y-Matrix in der PLS-DA Klassifizierungsvariablen. Die Regression erfolgt dann genauso wie in der PLS-Regression (siehe 2.2.3 PLSR) (Ballabio und Consonni 2013, S. 3790–3791).

2.2.6 Validierung

Die Validierung ist notwendig, um die Güte eines Modells für zukünftige Vorhersagen zu bestimmen (Kessler 2008, S. 153). Hierfür gibt es zwei verschiedene Möglichkeiten, die externe Validierung und die Kreuzvalidierung.

2.2.6.1 Externe Validierung

Für die externe Validierung wird ein Kalibier- und ein Validierungsdatensatz benötigt. Mit dem Kalibrierdatensatz wird das Modell erstellt und der Validierungsdatensatz dient zur Validierung. Hierbei ist wichtig darauf zu achten, dass Kalibrier- und Validierungsdatensatz repräsentativ für die später vorherzusagenden Daten sind. Ist dies der Fall, dann ist die externe Validierung die beste Validierungsmethode. Für die Validierung werden die durch das Modell vorhergesagten Werten mit den tatsächlich gemessenen Werten verglichen (Kessler 2008, S. 159). Die Ergebnisse werden in den folgenden Güteparametern dargestellt.

2.2.6.2 Kreuzvalidierung

Der Vorteil der Kreuzvalidierung liegt darin, dass kein zusätzlicher Datensatz für die Validierung benötigt wird. Der Nachteil liegt im deutlich erhöhten Rechenaufwand und der Tendenz zum Overfitting. Für die Kreuzvalidierung wird ein Teil der Proben aus dem Datensatz weggelassen und dann aus dem restlichen Datensatz ein Modell berechnet. Für die weggelassenen Daten wird dann mit diesem Modell eine Vorhersage getroffen. Dieses Prozedere wird so oft durchgeführt, bis alle Proben einmal weggelassen und Vorhersagen für diese getroffen wurden. Es gibt verschiedene Möglichkeiten, eine Kreuzvalidierung durchzuführen. Die einfachste ist die vollständige Kreuzvalidierung (Full Cross Validation oder Leave-One-Out) bei der immer nur eine Probe weggelassen wird. Die Alternative ist, den Datensatz in Segmente aufzuteilen und in jedem Schritt ein gesamtes Segment auszulassen. Wichtig ist, dass bei einer vollständigen Kreuzvalidierung keine Wiederholmessungen enthalten sind, dies würde zu einer zu optimistischen Validierung führen (Kessler 2008, S. 156).

2.2.6.3 Güteparameter

Die Güteparameter dienen dazu, eine Kalibration oder auch eine Validierung zu bewerten.

2.2.6.3.1 Standardfehler der Kalibration

Der Standardfehler der Kalibration gibt die Restvarianz nach der Kalibration an. Die Berechnung erfolgt durch die Aufsummierung aller quadrierten Differenzen der tatsächlichen Werte y_i und der Schätzwerte $\hat{y}_i$, welche durch die Anzahl an Freiheitsgrad df geteilt werden muss. Aus dem Ergebnis muss noch die Wurzel gezogen werden, um den Standardfehler der Kalibration zu erhalten (siehe Formel(2.67)) ((Kessler 2008, S. 94)).

$$S = \sqrt{\sum_{i=1}^{n} \frac{(y_i - \hat{y}_i)^2}{df}} \tag{2.67}$$

Dieser Wert ist jedoch ungeeignet für die Beurteilung eines PCR oder PLS-Modells, da bei diesem die Anzahl der Freiheitsgerade nicht bekannt ist.

2.2.6.3.2 Mittlerer Fehler RMSE

Als Alternative zum Standardfehler dient der Mittlere Fehler RMSE (Root Mean Square Error).

Der mittlere Fehler ist definiert als die Wurzel der durch die Anzahl der Proben n geteilten Fehlerquadratsumme. Die Fehlerquadratsumme wird im englischen Predicted Residual Sum of Squares (PRESS) genannt (siehe Formel (2.68)). Die Fehlerquadratsumme ist die aufsummierte Differenz zwischen den Schätzwerten $\hat{y}_i$ und den tatsächlichen Werten y_i (siehe Formel (2.69))(Kessler 2008, S. 94).

$$PRESS = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2.68}$$

$$RMSE = \sqrt{\frac{PRESS}{n}} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}} \tag{2.69}$$

Der RMSE lässt sich als Güteparameter für den Kalibrierfehler Root Mean Square Error of Calibration (RMSEC) und als Güteparameter für den Vorhersagefehler Root Mean Square Error of Prediction (RMSEP) nutzen. Der Unterschied liegt darin, wie die gemessenen Werte und die Schätzwerte erhalten werden. Für den RMSEC werden als Schätzwerte, die durch das Modell vorhergesagten Werte, welche auch schon für die Modellberechnung genutzt wurden, genutzt. Für den RMSEP werden als Schätzwerte die vorhergesagten Werte, entsprechend der Validiermethode (siehe 2.2.6.1 Externe Validierung und 2.2.6.2 Kreuzvalidierung) genutzt.

2.2.6.3.3 Systematischer Fehler (BIAS)

Der systematische Fehler BIAS kann bei der Kalibrierung sowie bei der Validierung angegeben werden. Bei der Kalibrierung beschreibt der BIAS, ob in dieser ein systematischer Fehler vorliegt. Bei einer guten Kalibrierung liegt der BIAS nahe bei null. Wenn der BIAS der Validierung sich stark von null unterscheidet, dann liegt hier ein systematischer Fehler vor.
Der BIAS berechnet sich, indem Differenzen der tatsächlichen Werte y_i und der Schätzwerte $\hat{y}_i$ aufsummiert und durch die Anzahl der Proben n geteilt werden (siehe Formel (2.70)) (Kessler 2008, S. 96).

$$BIAS = \sum_{i=1}^{n} \frac{(y_i - \hat{y}_i)}{n} \tag{2.70}$$

2.2.6.3.4 Standardfehler (SE)

Eine weitere Fehlerangabe ist der Standardfehler (SE). Ähnlich wie beim RMSE unterscheidet man beim Standardfehler den SEC (Standard Error Calibration) bei der Kalibration und den SEP (Standard Error Prediction) bei der Validierung. Die Berechnung erfolgt ähnlich zum RMSE, außer das zusätzlich noch der BIAS abgezogen wird (Kessler 2008, S. 96).

$$SE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i - BIAS)^2}{n-1}} \tag{2.71}$$

Unterscheiden sich RMSE und SE stark, dann weißt das Modell oder die Vorhersage einen systematischen Fehler auf (Kessler 2008, S. 96).

2.2.6.4 Bestimmung der Anzahl an Komponenten

Bei der PCR und auch bei der PLSR muss die passende Anzahl an Komponenten gefunden werden, um das optimale Modell zu erhalten. Wenn zu wenige Komponenten genutzt werden, kommt es zum sogenannte „Underfitting“, dabei beschreibt das Modell die Daten nicht ausreichend. Es kommt zu einem Kalibrierfehler. Werden zu viele Komponenten genutzt, kommt es zum „Overfitting“. Das Modell modelliert neben den tatsächlich existierenden Effekten auch das zufällige Rauschen. Der Schätzfehler durch das Modell steigt. Zusammen ergeben der Kalibierfehler und der Schätzfehler den Vorhersagefehler (siehe Abbildung 55). Dieser besitzt ein Optimum bei einer bestimmten Anzahl von Komponenten und dieses Optimum gilt es zu finden (Kessler 2008, S. 153).

Der RMSEC beschreibt nur den Modellfehler und sinkt deshalb mit jeder zusätzlichen Komponente (siehe Abbildung 56). Er ist deshalb nicht geeignet, um die passende Anzahl an Komponenten zu finden. Er kann jedoch zeigen, ob das Modell generell zu den Daten passt.

Um die optimale Anzahl an Komponenten zu bestimmen, wird der RMSEP benötigt. Hierbei ist die externe Validierung der Kreuzvalidierung vorzuziehen, wenn ein repräsentatives Testdatenset zur Verfügung steht (Kessler 2008, S. 157). Beim Beispiel aus Abbildung 56 („separates Datenset“) liegt die optimale Anzahl an Komponenten bei 4.
Die Kreuzvalidierung neigt tendenziell zu Overfitting (Kessler 2008, S. 163). Dies zeigt sich auch in Abbildung 56, das Optimum liegt bei 5 Komponenten, eine Komponente mehr als bei der externen Validierung. Da immer nur eine oder wenige Proben weggelassen werden, wird der Schätzfehler bei der Kreuzvalidierung unterschätzt.

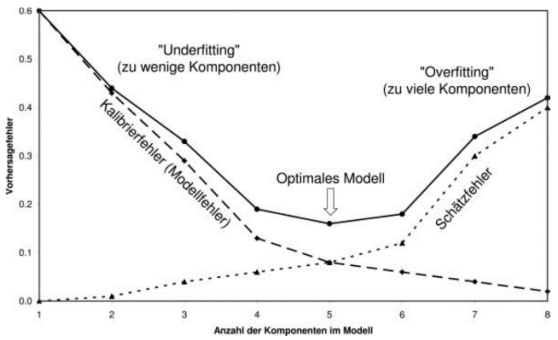


Abbildung 55: Abhängigkeit der Vorhersagefehlers von der Anzahl der Komponenten (Kessler 2008, S. 154)

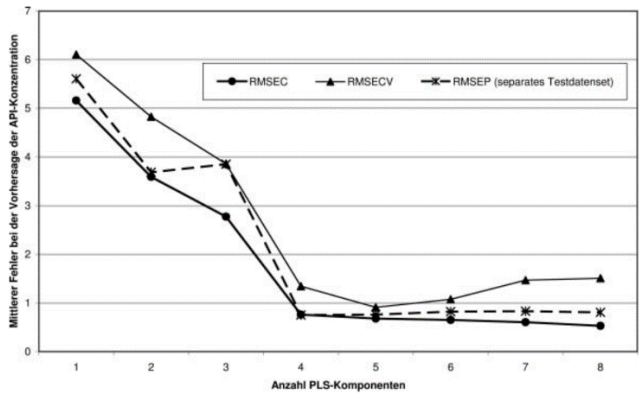


Abbildung 56: Validationplot: Beispiel für die verschiedenen RMSEs im Vergleich zur Anzahl der Komponenten (Kessler 2008, S. 161)

2.2.7 Bedeutung der wichtigen Plots in der PLSR

2.2.7.1 Validationplot

Der Validationplot wurde schon in Abbildung 56 dargestellt. Im Validationplot wird der Vorhersagefehler (RMSEP) gegen die Anzahl an Komponenten dargestellt. So lässt sich auch grafisch das Minimum ermitteln und so die optimale Anzahl an Komponenten finden.

2.2.7.2 Predictionplot

Im Predictionplot werden die gemessenen gegen die vorhergesagten Y-Werte aufgetragen. So lässt sich grafisch darstellen, ob das gewählte Modell zu den Daten passt.

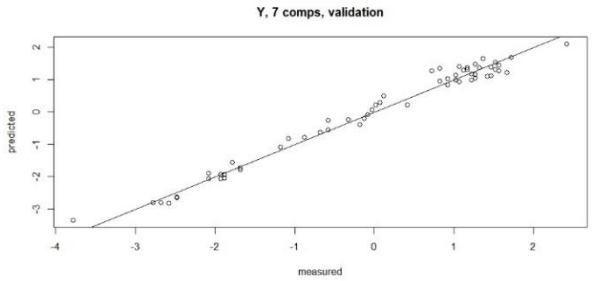


Abbildung 57: Predictionplot

2.2.7.3 Loadingplot

Der Loadingplot stellt dar, welche Variablen in einer Komponente wichtig sind (Kessler 2008, S. 29). In Beispiel Abbildung 58 lässt sich erkennen, dass die Variablen 110 bis 170 und die Variablen 230 bis 310 eine große Rolle für die erste Komponente spielen.

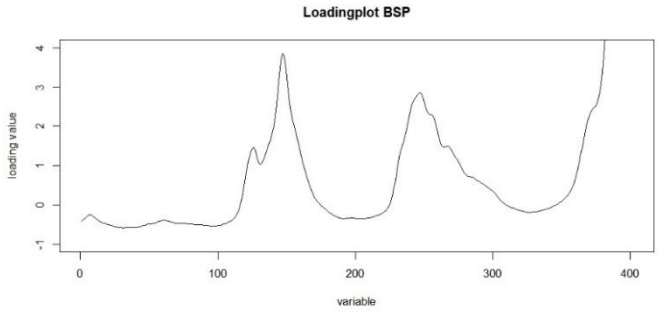


Abbildung 58: p-Loadings (X-Werte) für die erste Komponente eines Beispieldatensatzes

2.2.7.4 Scoreplot

Im Scoreplot lässt sich erkennen, ob sich Proben anhand der verschiedenen Hauptkomponenten unterscheiden lassen. In Abbildung 59 lassen sich anhand der 2. Komponente der Gruppen von Proben mit einer unterschiedlichen Konzentration an C unterscheiden.

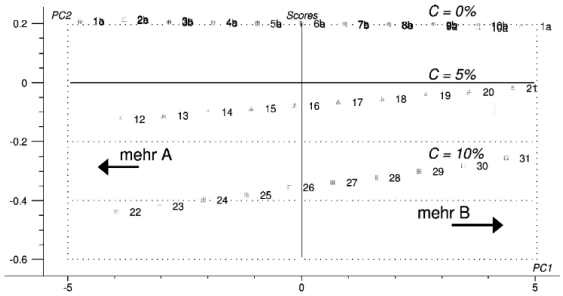


Abbildung 59: Beispiel Scoreplot: Drei Gruppen von Proben mit unterschiedlicher Konzentration an C (Kessler 2008, S. 133)

2.3 Datenvorbehandlung

Eine passende Datenvorbehandlung kann dazu genutzt werden, um die Vorhersageergebnisse einer Regressionsmethode zu verbessern. Aber auch eine Veränderung zum Schlechteren ist möglich (Kessler 2008, S. 185).

2.3.1 Zentrierung, Standardisierung & Skalierung

Bei der Zentrierung wird der Mittelwert aller Daten einer Variablen (eine Spalte j der X-Datenmatrix) $\bar{x}_j$ von dem einzelnen Wert der jeweiligen Variable subtrahiert $x_{i,j}$. So erhält man eine neue X-Datenmatrix mit den Werten $x(zent)_{i,j}$ (Kessler 2008, S. 37). Durch die Mittenzentrierung kann erreicht werden, dass die Scores und Loadings der PCA oder PLS einfacher zu interpretieren sind (Kessler 2008, S. 183).

$$x(zent)_{i,j} = x_{i,j} - \bar{x}_j \tag{2.72}$$

Bei der Standardisierung (auch z-Transformation genannt) werden die einzelnen Werte der zentrierten X-Datenmatrix $x(zent)_{i,j}$ durch die Standardabweichung s_j innerhalb einer Variable (eine Spalte j der X-Datenmatrix) dividiert. So erhält man die standardisierte X-Datenmatrix mit den Werten $x(stand)_{i,j}$ (Kessler 2008, S. 65). Eine Standardisierung wird angewendet, wenn alle Variablen die gleiche Gewichtung erhalten sollen. Ohne Standardisierung spielen Variablen mit höheren Werten eine größere Rolle im Modell. Die Standardisierung wird normalerweise nicht auf Spektren angewendet. Ansonsten würden Variablen, welche keine Absorption aufweisen und damit nur Rauschen enthalten, genauso stark gewichtet, wie die Variablen mit größerer Absorption (Kessler 2008, S. 183).

$$x(stand)_{i,j} = \frac{x(zent)_{i,j}}{s_j} = \frac{x_{i,j} - \bar{x}_j}{s_j} \tag{2.73}$$

Es ist auch möglich, Variablen ganz gezielt zu gewichten, hierfür werden die Spalten der X-Datenmatrix einfach mit Skalierungsfaktoren gewichtet (Kessler 2008, S. 183). Ähnlich wie bei der Standardisierung, nur dass hier die Skalierungsfaktoren frei gewählt werden und nicht als die Standardabweichung festgelegt sind. Hierbei ist es möglich, sie mit den Faktoren zu multiplizieren oder zu dividieren.

Die genannten Verfahren lassen sich ebenso auch auf die Y-Datenmatrix anwenden.

2.3.2 SNV

Bei der Standard Normal Variate (SNV) Transformation werden die Spektren standardisiert. Sie dient dazu, Streueffekte zu korrigieren. Dabei wird der Mittelwert $\bar{x}_i$ eines Spektrums (eine Zeile i der X-Datenmatrix) von jedem Datenpunkt im Spektrum $x_{i,j}$ subtrahiert. Danach werden die Datenpunkte durch die Standardabweichung s_i innerhalb des Spektrums (eine Zeile i der X-Datenmatrix) dividiert. Wichtig ist die SNV nicht mit der „normalen“ Zentrierung und Standardisierung zu verwechseln. Bei der SNV erfolgt die Zentrierung und Standardisierung zeilenweise anhand der X-Datenmatrix und nicht spaltenweise (vgl. Formel (2.73) und (2.74)) (Kessler 2008, S. 202).

$$x(SNV)_{i,j} = \frac{x_{i,j} - \bar{x}_i}{s_i} \tag{2.74}$$

Der Vorteil der SNV liegt darin, dass jedes Spektrum einzeln korrigiert wird und kein Referenzspektrum benötigt wird (vgl. mit 2.3.3 MSC), dies wirkt sich positiv aus, wenn die Variabilität zwischen den Spektren groß ist (Kessler 2008, S. 202).

2.3.3 MSC

Die Multiplicative Scatter Correction korrigiert Streueffekte und den Untergrund (Windig et al. 2008, S. 1153). Hierbei werden die Spektren mit einem Referenzspektrum korrigiert, welches die mittlere Streuung und die mittlere Basislinie repräsentiert. Als Referenzspektrum wird normalerweise das Mittelwertspektrum $\bar{X}$ genutzt. Nun werden mittels Least Square Verfahren (siehe 2.2.2 PCR Formel (2.13)) die einzelnen Spektren X_j (Vektor bestehend aus einer Spalte der X-Datenmatrix) auf das Referenzspektrum $\bar{X}$ (Vektor des Referenzspektrums) gefittet. Hierbei werden a_j und b_j für jedes Spektrum berechnet. Die Korrekturkoeffizienten a_j und b_j enthalten die Streuung und den Untergrund. Im Fehlerterm e_j steckt die chemische Information (Kessler 2008, S. 198).

$$X_j = a_j + b_j\bar{X} + e_j \tag{2.75}$$

Nun können die einzelnen Spektren X_j mit den Koeffizienten a_j und b_j korrigiert werden.

$$X(MSC)_j = \frac{X_j - a_j}{b_j} \tag{2.76}$$

Man erhält mit der MSC sehr ähnliche Ergebnisse wie mit der SNV (Kessler 2008, S. 202).

2.3.4 Glättung (Savitzky Golay)

Die Glättung dient dazu, dass Spektrometerrauschen zu beseitigen (Kessler 2008, S. 187). Es wird immer über eine ungerade Anzahl an Datenpunkten n im Spektrum geglättet (mindestens 3), um den mittleren Datenpunkt $(n+1)/2$ zu ersetzen. Mit diesem Vorgehen ersetzt man alle Datenpunkte x_j im Spektrum, bis auf die ersten und letzten $(n-1)/2$ Datenpunkte x_j, welche man entweder unverändert lässt oder weglässt. Im einfachsten Fall wird zum Glätten der Mittelwert zwischen den Datenpunkten gebildet, dann lassen sich die geglätteten Datenpunkte $x(smoothed)_j$ über die Formel (2.77) berechnen. Über je mehr Datenpunkte n geglättet wird, desto stärker ist die Glättung. Es besteht die Gefahr, dass auch über, nicht dem Rauschen zu verschuldende Änderungen, geglättet wird, dies nennt man Überglättung (Kessler 2008, S. 187).

$$x(smoothed)_j = \frac{\sum_{k=-((n-1)/2)}^{(n-1)/2}(x_{j+k})}{n} \tag{2.77}$$

Einer Glättung mittels Mittelwert ist häufig die Polynomglättung (Savitzky-Golay) vorzuziehen, hierdurch können Strukturen im Spektrum besser abgebildet werden. Dabei wird über ein Least-Square-Verfahren (siehe 2.2.2 PCR Formel (2.13)) ein Polynom durch die Datenpunkte n im Spektrum gelegt. Man erhält für jeden Datenpunkt x_j Regressionskoeffizienten B, mit denen sich für den mittleren Datenpunkt $(n+1)/2$ eine „Vorhersage“ treffen lässt, durch diese dann der mittlere Datenpunkt $(n+1)/2$ ersetzt wird (Kessler 2008, S. 187). Als Modell kann ein beliebiges Polynom gewählt werden, solange genügend Stützstellen für die Regression vorhanden sind. Je höher das gewählte Polynom, desto schwächer ist die Glättung, da die Anpassung an die Daten stärker ist. Ein Polynom 0. Grades entspricht der Mittelwertglättung.

2.3.5 Ableitung (Savitzky-Golay)

Eine Ableitung der Spektren eignet sich gut, um Basislinieneffekte zu entfernen und die spektrale Auflösung zu erhöhen, wodurch überlagernde Banden besser zu erkennen sind (Kessler 2008, S. 193).

Die einfachste Art, die Ableitung zu berechnen, ist die Differenz zum nächsten Datenpunkt (Formel (2.78)). Der große Nachteil hierbei ist, dass sich mit jedem Ableitungsschritt das Rauschen erhöht (Kessler 2008, S. 193–194).

$$x(derivated)_j = x_j - x_{j+1} \tag{2.78}$$

Die Lösung hierfür bietet die Ableitung über den Polynomfit (Savitzky-Golay), dabei kommt es zur Glättung der Spektren vor der Ableitung. Die Berechnung des Polynoms und der Regressionskoeffizienten erfolgt genauso wie in 2.3.4 Glättung (Savitzky Golay). Bevor jetzt der Datenpunkt $x(derivated)_j$ berechnet werden kann, muss das Polynom abgeleitet werden, ansonsten wird weiter wie bei der Glättung verfahren (Kessler 2008, S. 195). Bei höheren Ableitungen ist es häufig von Vorteil, die erste Ableitung mehrmals durch Polynomfit zu berechnen, so kommt es zu weniger Rauschen (Kessler 2008, S. 197).

2.3.6 Basislinienkorrektur

Die Basislinienkorrektur dient dazu, den Untergrund aus Spektren zu entfernen (Kessler 2008, S. 190). Ein Ansatz, um den Untergrund zu bestimmen, ist eine iterative Anpassung mithilfe eines Polynoms (Feng et al. 2006, S. 59).

Im Folgenden wird der Algorithmus beschrieben wie der Untergrund berechnet wird (Feng et al. 2006, S. 60). Als Ausgangsdaten wählt man ein Polynom (Formel (2.79)) des Grades n und ein Spektrum. Die y-Daten werden zur y-Datenmatrix $y_{k=0}$ und die x-Daten x werden entsprechend des Polynoms zur X-Datenmatrix X (Formel (2.79)). j steht für den j-ten Wert der y- und x-Daten. Die Laufvariable für die Iterationsschritte ist k.

$$y(x) = a_0 + a_1 x + a_n x^2 + \cdots + a_n x^n \tag{2.79}$$

$$\begin{pmatrix} 1 & x_1 & {x_1}^2 & \dots & {x_1}^n \\ 1 & x_2 & {x_2}^2 & \dots & {x_2}^n \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_j & {x_j}^2 & \dots & {x_j}^n \end{pmatrix} \tag{2.80}$$

Nun kann die Schleife zur Berechnung des Untergrundes beginnen. Zuerst werden die durch Polynomfit vorhergesagten Werte für x berechnet, welche $\hat{b}_k$ genannt werden.

$$\hat{b}_k = X(X^T X)^{-1} X^T y_{(k-1)} \tag{2.81}$$

y_k erhält man durch Vergleich von $\hat{b}_k$ und $y_{(k-1)}$. Ist der Wert in $\hat{b}_k$ kleiner als in $y_{(k-1)}$, dann wird $\hat{b}_k$ übernommen, ansonsten $y_{(k-1)}$.

$$y_{k,j} = \hat{b}_{k,j}\ ; wenn\ \hat{b}_{k,j} < y_{(k-1),j} \tag{2.82}$$

$$y_{k,j} = y_{(k-1),j}\ ; wenn\ \hat{b}_{k,j} \geq y_{(k-1),j} \tag{2.83}$$

Jetzt muss entschieden werden, ob die Anpassung ausreichend gut ist oder ein weiterer Iterationsschritt notwendig sind. Hierfür wurde am Anfang ein Abbruchkriterium mit meist $p < 0.001$ festgelegt. p wird nach Formel (2.84) berechnet. Die doppelten Betragsstriche $\| \quad \|$ besagen, dass hier die Länge des Vektors berechnet wird. Unter dem Bruch wurde im Vergleich zur Quelle (Feng et al. 2006, S. 60) die doppelten Betragsstriche hinzugefügt, da es keinen Sinn ergibt, einen Skalar durch einen Vektor zu teilen. Für den ersten Iterationsschritt wird $\hat{b}_{k-1}$ mit y_0 ersetzt. Wenn das Abbruchkriterium erfüllt wird, ist $\hat{b}_k$ der geschätzte Untergrund, anderenfalls wird k um eins erhöht und wieder zu Formel (2.81) zurückgesprungen.

$$p = \frac{\| \hat{b}_k - \hat{b}_{k-1} \|}{\| \hat{b}_{k-1} \|} \tag{2.84}$$

(2.85)

$$p < 0.001$$

2.3.7 Variablenselektion

Variablenselektionsalgorithmen dienen dazu, bei multivariaten Datensätzen die Variablen auszuwählen, welche am besten die vorherzusagenden Variablen beschreiben. Es sollen Variablen entfernt werden, welche keine oder nur wenig Information liefern und diejenigen, welche stark miteinander korrelieren (Andrade et al. 2004, S. 123).

Es gibt zwei einfachere Vorgehensweisen, wie man bei der Selektion prinzipiell vorgehen kann, die forwards selection (vorwärts Auswahl) und die backwards elimination (rückwärts Eliminierung). Bei der forwards selection werden nach und nach einzelne Variablen hinzugefügt und bei der backwards elimination nach und nach einzelne Variablen entfernt (John Hopkins University, S. 125). Es gibt noch weitere komplexere Ansätze, welche hier nicht näher thematisiert werden. Abgebrochen werden alle Selektionsalgorithmen, sobald eine festgelegte Anzahl an Variablen erreicht wird oder man lässt sie komplett durchlaufen und entscheidet anhand der aufgezeichneten Ergebnisse, wie viele Variablen man nutzen möchte.

2.3.7.1 Procrustes Analysis

Bei der Procrustes Analysis (PA) werden zwei Matrizen miteinander verglichen. Hierfür wird die eine Matrix gestaucht oder gedehnt und gedreht, sodass sie der anderen Matrix möglichst ähnlich ist (Andrade et al. 2004, S. 125–126).
Um die Berechnung durchzuführen, ist es sinnvoll, beide Matrizen zu zentrieren (Andrade et al. 2004, S. 126). Die zentrierten Matrizen sind A und B. Um die Rotationsmatrix Q zu erhalten, wird eine Singulärwertzerlegung (Svd) von $B^T * A$ durchgeführt. Man erhält drei Matrizen U, S und V. Nun multipliziert man die Matrix V mit der transponierten Matrix U, um die Rotationsmatrix Q zu erhalten (Siswadi et al. 2012, S. 859).

$$Svd(B^T A) = USV^T \tag{2.86}$$
$$Q = VU^T \tag{2.87}$$

Die gedrehte Matrix P von B lässt sich durch Multiplizieren mit der Rotationsmatrix Q berechnen.

$$P = B \cdot Q \tag{2.88}$$

Als Güteparameter für die Ähnlichkeit der Matrizen wird die Güte der Anpassung (Goodness of fit GF) genutzt. Hierfür muss zuerst die Procrustes Distanz E berechnet werden. Für diese Berechnung werden die Spuren (engl.Trace) verschiedener Matrizen miteinander verrechnet (Siswadi et al. 2012, S. 860).

$$E = tr(AA^T) - \frac{tr^2(X^T P^T)}{tr(BB^T)} \tag{2.89}$$

Nun kann die Güte der Anpassung GF berechnet werden, welche im Bereich von 0 bis 1 liegt.

$$GF = 1 - \frac{E}{tr(AA^T)} \tag{2.90}$$

Um mit der Procrustes Analyse (PA) Variablen zu selektieren, werden Variablen schrittweise entfernt. Hierfür wird die X-Datenmatrix mittels PA mit der X-Datenmatrix, bei welcher eine Spalte komplett auf null gesetzt wurde, verglichen. Dies wird so oft durchgeführt bis alle Spalten einmal auf null gesetzt wurden. Danach wird die Spalte (Variable) entfernt, bei der die Güte der Anpassung GF am

größten war und das ganze Prozedere wiederholt (Siswadi et al. 2012, S. 862). Es ist auch eine Vorwärtsselektion möglich, hierfür wird immer die Variable mit der kleinsten GF hinzugefügt.

2.3.7.2 PCA - Procrustes Analyse

Bei der Principal Component Analysis - Procrustes Analyse (PCA-PA) wird ähnlich vorgegangen, wie bei der Procrustes Analyse (PA). Es werden 2 X-Datenmatrizen miteinander verglichen, bei der einen wird jetzt aber eine Spalte weggelassen, anstatt sie auf 0 zu setzen. Vor der Procrustes Analyse wird eine PCA durchgeführt. Als Matrizen A und B für die PA (siehe Formel (2.86) bis (2.90)) werden nun aber die Scores, einer vorher festgelegten Anzahl an Komponenten, der PCA verwendet. Es werden ebenfalls alle Spalten einmal weggelassen und dann die Variable entfernt für die der GF am größten war und dieses Prozedere wird dann wie bei der PA wiederholt (Siswadi et al. 2012, S. 862).

2.3.7.3 CARS

Der Competitive Adaptive Reweighted Sampling (CARS) Algorithmus entfernt Variablen ebenfalls schrittweise. Es werden testweise alle Variablen einmal entfernt, und jeweils das PLSR-Model berechnet und dann mittels Kreuzvalidierung der RMSECV bestimmt. Endgültig wird die Variable entfernt, bei der der RMSECV am kleinsten war. Danach geht das Prozedere wieder von vorne los und alle übrigen Variablen werden testweise entfernt (Xu et al. 2014, S. 1092). Dieses Verfahren kann auch mittels Vorwärtsselektion durchgeführt werden.

2.3.8 Varianzkorrektur

Es ist möglich, dass NIR-Messung durch Effekte ungewollt beeinflusst werden. Hierdurch kommt es zu einer Variabilität in den Spektren. Die folgende Methode dient dazu, diese Effekte durch Korrektur zu reduzieren.

2.3.8.1 IIR

Die Independent Interference Reduction (IIR) ist eine Methode, bei der die Interfenzen durch die PCA eines zusätzlichen Datensatzes modelliert werden (Hansen 2001, S. 123), so können Interferenzen und Streueffekte entfernt werden. Hierfür werden zwei Datensätze benötigt. Der Erste ist ein komplexer Datensatz $X_{special}$, bei dem sich alle Parameter ändern, auch der von Interesse für Kalibration. Für diesen Datensatz müssen die Responsewerte $Y_{special}$ bekannt sein. Der Zweite ist ein einfacher Datensatz X_{simple}, welcher nur die Varianz zusätzlicher Effekte abbildet. Im zusätzlichen Datensatz dürfen die Parameter von Interesse nicht variieren, alle störenden Parameter müssen sich aber ändern. Die Responsewerte müssen nicht bekannt sein, solange sichergestellt ist, dass diese sich nicht ändern (Hansen 2001, S. 124).

Datenvorbereitung für den Kalibrierdatensatz (Hansen 2001, S. 125):

Die Reihen der Matrizen werden im Folgenden mit i bezeichnet und die Spalten mit j.

Im ersten Schritt muss die X-Datenmatrix des einfachen Datensatzes X_{simple} zentriert werden. Man berechnet zuerst das Mittelwertspektrum $\bar{X}_{simple}$ von X_{simple} (Formel (2.91)). Danach wird dieses Mittelwertspektrum $\bar{X}_{simple}$ von jedem Spektrum in X_{simple} subtrahiert, um den zentrierten einfachen Datensatz X^c_{simple} zu erhalten.

$$\bar{X}_{simple;\,j} = \frac{\sum_1^i X_{simple;\,i,j}}{i_{X_{simple}}} \tag{2.91}$$

$$X^c_{simple;\,i} = X_{simple;\,i} - \bar{X}_{simple} \tag{2.92}$$

Als nächstes wird die PCA (siehe 2.2.1 PCA) durchgeführt, um die Loadingmatrix P_{simple} zu erhalten.

$$X^c_{simple} = T_{simple} \cdot {P_{simple}}^T \tag{2.93}$$

Von der Loadingmatrix P_{simple} werden die ersten k Spalten weiter genutzt. k ist die Anzahl an Komponenten der PCA, welche genutzt werden sollen.

Das vorhin berechnete Mittelwertspecktrum $\bar{X}_{simple}$ muss für das weitere Vorgehen von $X_{special}$ subtrahiert werden. Man erhält $X^s_{special}$.

$$X^s_{special;\,i} = X_{special;\,i} - \bar{X}_{simple} \tag{2.94}$$

Nun kann die Scorematrix $T_{special}$ berechnet werden (Formel (2.95)). Mit $T_{special}$ und P_{simple} erhält man die Korrekturmatrix $X_{corrections}$ (Formel (2.96)). Die Formeln (2.95) (2.96) lassen sich zur Formel (2.97) zusammenfassen.

$$T_{special} = X^s_{special} \cdot P_{simple} \tag{2.95}$$
$$X_{corrections} = T_{special} \cdot {P_{simple}}^T \tag{2.96}$$
$$X_{corrections} = X^s_{special} \cdot P_{simple} \cdot {P_{simple}}^T \tag{2.97}$$

Zum Schluss muss noch die Korrekturmatrix $X_{corrections}$ von $X^s_{special}$ abgezogen werden (Formel (2.98)), um den korrigierten Datensatz $X_{special,\ corrected}$ zu erhalten.

$$X_{special,\ corrected} = X^s_{special} - X_{corrections} \tag{2.98}$$

Es wird empfohlen, den korrigierten Datensatz nochmals zu zentrieren, bevor eine Regression durchgeführt wird.

Datenvorbereitung für die Vorhersage mit einem bestehenden Modell (Hansen 2001, S. 125–126):
Um Vorhersagen mit einem bestehenden Modell und einem neuen X-Datensatz X_{pred} treffen zu können, muss X_{pred} wie folgt korrigiert werden.

Zuerst wird Mittelwertspektrum $\bar{X}_{simple}$ von jedem Spektrum in X_{pred} subtrahiert werden.

$$X^s_{pred;\,i} = X_{pred;\,i} - \bar{X}_{simple} \tag{2.99}$$

Danach wird die Korrekturmatrix $X_{corrections,pred}$ berechnet.

$$X_{corrections,pred} = X^s_{pred} \cdot P_{simple} \cdot {P_{simple}}^T \tag{2.100}$$

Zum Schluss muss die Korrekturmatrix $X_{corrections,pred}$ von X^s_{pred} subtrahiert werden, um den korrigierten Datensatz $X_{pred,\ corrected}$ zu erhalten.

$$X_{pred,\ corrected} = X^s_{pred} - X_{corrections,pred} \tag{2.101}$$

3 Softwareverzeichnis

R Core Team (2020). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/, zuletzt geprüft am 25.06.2020.

RStudio Team (2020). RStudio: Integrated Development Environment for R. RStudio, PBC, Boston, MA URL http://www.rstudio.com/, zuletzt geprüft am 25.06.2020.

Max Kuhn (2020). caret: Classification and Regression Training. R package version 6.0-86. https://CRAN.R-project.org/package=caret, zuletzt geprüft am 25.06.2020.

Gábor Csárdi (2017). crayon: Colored Terminal Output. R package version 1.3.4. https://CRAN.R-project.org/package=crayon, zuletzt geprüft am 25.06.2020.

Matt Dowle and Arun Srinivasan (2019). data.table: Extension of `data.frame`. R package version 1.12.8. https://CRAN.R-project.org/package=data.table, zuletzt geprüft am 25.06.2020.

Alexandros Karatzoglou, Alex Smola, Kurt Hornik, Achim Zeileis (2004). kernlab - An S4 Package for Kernel Methods in R. Journal of Statistical Software 11(9), 1-20. URL https://www.jstatsoft.org/article/view/v011i09, zuletzt geprüft am 07.08.2020

Tal Galili (2019). installr: Using R to Install Stuff on Windows OS (Such As: R, 'Rtools', 'RStudio', 'Git', and More!). R package version 0.22.0. https://CRAN.R-project.org/package=installr, zuletzt geprüft am 25.06.2020.

Bjørn-Helge Mevik, Ron Wehrens and Kristian Hovde Liland (2019). pls: Partial Least Squares andPrincipal Component Regression. R package version 2.7-2. https://CRAN.R-project.org/package=pls, zuletzt geprüft am 25.06.2020.

Hans W. Borchers (2019). pracma: Practical Numerical Math Functions. R package version 2.2.9. https://CRAN.R-project.org/package=pracma, zuletzt geprüft am 25.06.2020.

Antoine Stevens and Leornardo Ramirez-Lopez (2020). An introduction to the prospectr package. R package Vignette R package version 0.2.0. https://github.com/l-ramirez-lopez/prospectr, zuletzt geprüft am 25.06.2020.

Winston Chang (2019). R6: Encapsulated Classes with Reference Semantics. R package version 2.4.1. https://CRAN.R-project.org/package=R6, zuletzt geprüft am 25.06.2020.

4 Anhang

4.1 Gasoline Dataset

Beim Gasoline Datenset handelt es sich um ein Datenset aus 60 Proben, mit bekannter Oktanezahl. Zu jeder Probe wurden die entsprechenden NIR-Spektren im Bereich von 900 bis 1700nm in 2nm Schritten aufgenommen (Kalivas 1997). Der Datensatz ist im R-Package pls (Bjørn-Helge Mevik, Ron Wehrens and Kristian Hovde Liland (2019)) unter der Variable gasoline aufrufbar. Er kann mit folgender Codezeile angezeigt werden:

```
print(gasoline)
```

4.2 Originaldaten Theorie PCA

*x*1	*x*2
1,1	6,5
2,1	6,5
2,1	5,5
1,1	4,5
2,1	3,5
–0,9	–2,5
–1,9	–4,5
–0,9	–5,5
–2,9	–6,5
–1,9	–7,5

Anhang 1: Originaldaten Theorie PCA (Kessler 2008, S. 30)