Thesis für den Abschluss als B.Sc.
Physikalische Technik

Verwendung eines Clustering-Algorithmus zur Phasenerkennung für dotiertes Hf und Zr

Fakultät für Angewandte Naturwissenschaften und Mechatronik

Trien Maximilian

Hochschule München

Dezember 2020

Verwendung eines Clustering-Algorithmus zur Phasenerkennung für dotiertes Hf und Zr

Trien Maximilian

Physikalische Technik

Beaufsichtigt bei Professor Kersch

Als qualifizierte Abschlussarbeit von
Maximilian Trien
für den Abschluss B.Sc. vom Prüfungsamt genehmigt

Dezember 2020

Vorsitzender Prof. Kersch

Prof. Moreira

Hochschule München

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Seminararbeit mit dem Titel „Verwendung eines Clustering-Algorithmus zur Phasenerkennung für dotiertes Hf und Zr “ selbständig verfasst habe, dass ich sie zuvor an keiner anderen Hochschule und in keinem anderen Studiengang als Prüfungsleistung eingereicht habe und dass ich keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Stellen der Arbeit, die wörtlich oder sinngemäß aus Veröffentlichungen oder aus anderweitigen fremden Äußerungen entnommen wurden, sind als solche kenntlich gemacht.

Ort, Datum Unterschrift

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

ABKÜRZUNGSVERZEICHNIS

ML Maschinelles Lernen

XRD X-Ray Diffraction

PRDF Partial Radial Distribution Function

DR Dimensionsreduzierung

PCA Principal Component Analysis

NMF Nonnegative Matrix Factorization/Nicht-negative Matrixfaktorisierung

t-SNE T-Distributed Stochastic Neighbor Embedding

DB Dichtebasiertes Clustern

ZrO_2 Zirconium(IV)-oxid

HfO_2 Hafnium(IV)-oxid

m monoklin

o orthorhombisch

p-o polar-orthorhombisch

t tetragonal

FDA Funktionale Datenanalyse

Use of a clustering algorithm for phase detection for doped Hf and Zr

Trien Maximilian

University of Applied Science Munich

(Supervised by Professor Kersch)

(Abstract)

The aim of this thesis is to achieve an automated phase recognition of doped hafnium and zirconium oxide by unsupervised learning methods. Since doping can transform the phases of the theoretically simulated crystals, the search and classification of this changed phase is always problematic. Using cluster algorithms and descriptors such as the XRD pattern or the partial radial distribution function, it can be shown that the phases can be clustered into their already known previous phases as well as the new transformed phase. The functional data analysis is used for the basic differentiation of the specific characteristics of the XRD patterns. With the help of dimension reduction methods, these can be isolated and filtered in order to classify them with clustering algorithms. It turns out that the PCA, NMF and TSNE algorithms are particularly well suited for filtering phase characteristics. The k-means cluster algorithm shows clear grouping with single outliers. Thus, a method can be shown which performs the classification of the phase more efficiently and with high accuracy, as well as outliers from these groups can now be quickly visualized for further evaluation.

Verwendung eines Clustering-Algorithmus zur Phasenerkennung für dotiertes Hf und Zr

Trien Maximilian

Hochschule München

(Beaufsichtigt bei Professor Kersch)

(Abstract)

Ziel dieser Arbeit ist es eine automatisierte Phasenerkennung von dotiertem Hafnium- und Zirkonoxid durch Methoden des unüberwachten Lernens zu erreichen. Da durch die Dotierung die Phasen der theoretisch simulierten Kristalle transformieren können, ist die Suche und Einordnung dieser veränderten Phase stets problematisch. Durch Clusteralgorithmen und Deskriptoren der Röntgendiffraktometrie/-Partial Radial Distribution Function lässt sich zeigen, dass die Phasen sich in ihrem bereits vorherigen bekannten Zustand, sowie der neu transformierten Phase clustern lassen. Die funktionale Datenanalyse dient der grundsätzlichen Unterscheidung der spezifischen Charakteristika der XRD-Muster. Mithilfe von Dimensionsreduzierungsmethoden können diese isoliert und gefiltert werden, um sie anschließend mit Clusteralgorithmen einzuordnen. Es stellt sich heraus, dass die PCA, NMF und TSNE Algorithmen sich besonders gut eignen, um Merkmale der Phase zu filtern. Der K-Means Clusteralgorithmus zeigt deutliche Gruppenbildung mit einzelnen Ausreißern auf. Es kann also ein Verfahren aufgezeigt werden, welches die Einordnung der Phase effizienter und mit hoher Genauigkeit ausführt. Wichtige Ausreißer aus diesem Gruppen können nun schnell für eine weitere Begutachtung visualisiert werden.

I. EINLEITUNG

1.1 Motivation und Hintergrund

Maschinelles Lernen hat in den letzten Jahren in vielen Bereichen neue Durchbrüche ergeben und Anwendungen ergeben. Im wirtschaftlichen Sektor werden große Fortschritte in Bereichen des Autonomen Fahrens, der Gesundheitsbranche, sowie in der Energiegewinnung erwartet. Es ist von einer Technologie die Rede, die den Markt und die Zukunft der Menschheit mit- und neugestalten kann.

Durch die immer weiter ansteigende Rechenleistung von Prozessoren, gelingt es heutzutage mit ML-Algorithmen sehr viel bessere Ergebnisse zu erzielen. Hinzu kommt auch die intensivere Forschung in dem Gebiet der Künstlichen Intelligenz, sowie das interdisziplinäre Zusammenarbeiten von unterschiedlichsten Fachgebieten. Das führt dazu, dass wir auch in unserem alltäglichen Leben einen immer größeren Einfluss von Algorithmen erleben. Durch die zunehmende Digitalisierung unserer Gesellschaft fallen gigantische Datenmengen an, die aufgenommen und ausgewertet werden. Aufgrund der Komplexität der Daten und Prozesse bleibt dem menschlichen Verstand oftmals der Zugang zu den darunterliegenden Vorgängen verwehrt. Dieses Phänomen tritt auch in naturwissenschaftlichen Bereichen auf, wo über Sensoren oder Simulationen Daten generiert werden. Mit herkömmlichen Verfahren können Auswertungen aus diesen Datenmengen teilweise nur sehr mühevoll erzielt werden. Maschinelles Lernen sollte deshalb als Tool Verwendung finden, um somit in kürzerer Zeit gleiche oder bessere Erkenntnisse zu gewinnen. Anwendungen in der Physik finden sich heutzutage oft in dem Gebiet der Teilchenphysik, Quantenphysik, Astronomie oder Festkörperphysik. Teilweise werden mithilfe von ML-Algorithmen erstaunlich gute Ergebnisse erreicht, wobei die konventionelle Herangehensweise des Forschers und des ML-Algorithmus oftmals grundsätzlich verschieden sind.

1.2 Ziel und Struktur der Arbeit

Aus diesem Grund beschäftigt sich diese Arbeit mit dem Thema des ML mit Bezug auf einen Datensatz der Festkörperphysik. Dazu stehen Berechnungen aufgrund der Dichtefunktionaltheorie des Supercomputers „SuperMUC“ in Garching zur Verfügung. Es werden Kristalle simuliert, die mit einem Dotierstoff versehen werden, um auf molekularer Ebene die ferroelektrischen Eigenschaften zu verändern. Mit den Verbindungen ZrO_2 und HfO_2 können sehr dünne ferroelektrische Schichten gebildet werden, welche in der Halbleitertechnik ihre Anwendung finden. Das Einbringen des Dotierstoffes in verschiedenen Konzentrationen in die unterschiedlichen Phasen monoklin, polar-orthorhombisch, orthorhombisch und tetraedrisch, führt dazu, dass die Verbindung aus energetischen Gründen nach dem Dotieren in einen Grundzustand zurückkehren will. Hierbei kann das Phänomen auftreten, dass der Kristall mit Dotand seine Phase beibehält oder in eine andere Phase wechselt. Diese kann eine der oben aufgezählten Phasen oder eine gänzlich Unbekannte sein.

Zum jetzigen Standpunkt wird jede simulierte Relaxation einer Zelle über eine Software visualisiert und mit dem Auge begutachtet. Es kann teils anhand der Kristallstruktur erkannt werden, ob die Verbindung mit jeweiligen Dotanden seine Phase geändert hat. Dieser Prozess ist äußerst zeitaufwendig und teilweise durch das Fehlen allgemein gültiger Bewertungskriterien nicht möglich. Deshalb wurde in einem vorausgegangenen Projekt versucht die Daten mit Algorithmen des überwachten Lernens zu klassifizieren. Da ein überwachter ML-Algorithmus als Datenmenge eine möglichst große Anzahl an manuell gekennzeichneten Features benötigt, um eine zufriedenstellende Genauigkeit zu erreichen, ist dies ebenfalls mit einem hohen menschlichen Zeitaufwand verbunden.

Diese Arbeit beschäftigt sich daher mit dem unüberwachten ML, wobei keine vorher bekannten, richtig klassifizierten Phasen vorhanden sein müssen. Es werden nur Features[1] benötigt, die eine möglichst hohe Aussagekraft über die entsprechende Phase aufweisen. Der Algorithmus soll in der Struktur der Daten die richtigen Gruppen auffinden und benennen. Hierbei wird sich besonders den verschiedenen Cluster-Algorithmen gewidmet, da sich die explorative Natur des Vorgehens besonders gut eignet, um eventuell auch unbekannte Phasen im Datensatz zu finden. Auf die Features, die das wichtigste Merkmal zur Beschreibung der Phase sind, wird im späteren Verlauf weiter eingegangen. Ein zusätzlicher

[1] Ein Feature ist eine messbare Variable, die bestimme Eigenschaften des zu betrachtenden Objektes besitzt.

Bestandteil liegt darin, für die gegebenen Daten passende Cluster-Algorithmen auszuwählen und auf die jeweiligen Featurevektoren abzustimmen. Ziel ist es, bei Erreichen einer hohen Genauigkeit der Bestimmung der Phasen, den Algorithmus regelmäßig nach Simulationen aus dem SuperMUC einzusetzen, um eine effizientere Untersuchung möglich zu machen. Die Clusteranalyse der Phasen wird für darauffolgende Projekte zur Grundlagenforschung im Labor für Modellbildung und Simulation benötigt.

Die allgemeine Vorgehensweise wird nun kurz dargestellt und erläutert, um eine bessere Übersicht und Struktur der Arbeit zu gewinnen. Zuerst wird eine kurze Einführung in die Theorie des maschinellen Lernens und dem unüberwachten Lernen gegeben. Zudem wird über wichtiges Hintergrundwissen und einige Begriffe zu den simulierten Daten vorab aufgeklärt. Im zweiten Teil werden Deskriptoren gesucht, die sich zur Strukturerkennung der Kristalle besonders gut eignen. Darauffolgend wird auf die Cluster-Algorithmen, deren Methodik, sowie auf die angepassten Parameter der Algorithmen weiter eingegangen. Anschließend werden die Ergebnisse genauer dargestellt, analysiert und optimiert. Zum Ende der Arbeit wird eine Zusammenfassung mit weiteren Zielen präsentiert, um die These erfolgreich abzuschließen.

II. THEORIE

2.1 Unüberwachtes maschinelles Lernen

Beim überwachten maschinellen Lernen wird vorausgesetzt, dass mit p Dimensionen die Featurevektoren $X_1, X_2, ..., X_p$ und die dazugehörige Antworten $Y_1, Y_2, ..., Y_p$ (Labels) bekannt sind. Mit Anzahl der n Observationen kann damit der Datensatz X einer $n \times p$ Matrix gebildet werden. Ziel ist es aus p Features die Antworten Y vorherzusagen, indem ein Modell für das maschinelle Lernen trainiert und auf noch nicht gesehene Daten angewandt wird [1]. Im Gegensatz dazu sind beim unüberwachten Lernen keine Labels Y bekannt, wobei nur von einem Datensatz X ausgegangen wird. Dadurch verändert sich das Ziel des ML-Algorithmus von einer reinen Vorhersage der Labels Y, zu einer explorativen Untersuchung des Datensatzes X. Hierbei wird besonders Wert auf die Visualisierung der zugrundeliegenden Struktur der Daten gelegt, die durch eine Vielzahl von Tools[2] nutzbar gemacht werden können. Die Vorgehensweise des unüberwachten Lernens ähnelt hier sehr stark dem experimentellen Vorgehen in diversen naturwissenschaftlichen Bereichen wie der Physik oder Chemie. Vor allem in der Physik ist bei vielen Problemen eher die zugrundeliegende Verteilung, die die Daten generiert, von großer Bedeutung [2].

Das nicht Vorhandensein der Label Y sowie die oftmals hohe Anzahl an Dimensionen der Features X stellt das unüberwachte Lernen aber auch vor eine Reihen von Problemen. Im Gegensatz zum überwachten Lernen sind viele Bereiche noch nicht hinreichend erforscht und entwickelt. Es gibt keine klare Vorgehensweise sowie feste Methoden, um die Qualität der Ergebnisse zu bewerten, welche zu einer starken Subjektivität führen. Das Problem der hohen Dimensionalität in Datensätzen kann über gewisse Dimensionsreduzierungsmethoden teilweise behoben werden, wobei dies aber stets mit einem Informationsverlust einhergeht.

[2] Eine große Anzahl an Bibliotheken die in der Sprache Python zu Verfügung stehen.

2.1.1 Dimensionsreduzierung

Die Dimensionsreduzierung (DR) ist direkt verwandt mit der Auswahl der Features sowie der Aufgabe des Gruppieren der Daten. Dieser Vorgang geht immer auch mit einer Reduzierung des Rauschens der Daten einher. Hierbei gibt es zwei verschiedene Klassen zur Auswahl: Lineare und nichtlineare Methoden [3].

In den nachfolgenden Kapiteln zur Dimensionsreduzierung werden jeweils einzelne Optimierungsprobleme gelöst. Der Lernprozess in diesen DR-Methoden geschieht stets über eine Minimierung einer sogenannten Kostenfunktion. Diese Funktion dient als Methode zur Evaluierung, wie gut der jeweilige Algorithmus den Datensatz X modelliert. In unseren Fällen werden als Kostenfunktion die mittlere quadratische Fehlerfunktion und die Kullback-Leibler-Divergenz[3] verwendet. Die am häufigsten verwendete Methode zur Minimierung einer Kostenfunktion ist das Gradientenverfahren.

2.1.1.1 Principal Component Analysis (PCA)

Wenn die einzelnen Features des vorliegenden Datensatzes X miteinander korreliert sind, kann eine Dimensionsreduzierung durch die PCA erfolgen. Die PCA wird durch den Vorgang bestimmt, bei dem die Hauptkomponenten der zugrundeliegenden Daten berechnet werden. Ziel ist es eine Repräsentation der Daten in einer niedrigeren Dimension l zu finden, die so viel wie möglich Variation des gesamten Datensatzes beinhalten. Die Berechnung und Sortierung der Hauptkomponenten Z wird bei der PCA in Unwichtigkeit mit absteigender Reihenfolge sortiert. Die k-te Hauptkomponente steht orthogonal zu allen Hauptkomponenten davor. Jede neue Dimension ist eine Linearkombination von p Featurevektoren. Ein Hauptkomponentenvektor $Z_{(i)} = (Z_1, \ldots, Z_l)_{(i)}$ lässt sich mit den Reihenvektoren $X_{(i)}$ wie folgt ausdrücken:

$$Z_{k(i)} = X_{(i)} \times \phi_{(k)} \qquad \text{mit} \qquad i = 1, \ldots, n \qquad k = 1, \ldots, l \tag{1}$$

Die Elemente $\phi_{(k)} = (\phi_1, \ldots, \phi_p)_{(k)}$ sind im allgemeinen Sprachgebrauch als Gewichte der Hauptkomponenten bekannt. Diese Hauptkomponenten sind an die Normierungsbedingung $\sum_{k=1}^{l} \phi_k^2 = 1$ gebun-

[3] Auch als “Information Gain„ bekannt, dass ein Maß für die Unterschiedlichkeit zweier Wahrscheinlichkeitsverteilungen darstellt.

den. Zudem kann die Annahme getroffen werden, dass der Mittelwert eines jeden Features p sich bei Null befindet. Um die Varianz des ersten Gewichtsvektors $\phi_{(1)}$ zu berechnen, muss folgende Bedingung erfüllt sein: [1]

$$\phi_{(1)} = (\phi_1, \ldots, \phi_p) = \arg\max_{\|\phi\|=1} \left\{ \sum_i (Z_1)^2_{(i)} \right\} \tag{2}$$

Nachdem der erste Gewichtsvektor $\phi_{(1)}$ gefunden wurde, können die Hauptkomponenten der Reihe berechnet werden. Das Problem in (2) kann mit der Eigenwertszerlegung/Spektralzerlegung aus der Linearen Algebra gelöst werden. Zuerst wird aus p Feature der Matrix X der Mittelwert berechnet. Daraufhin soll von X die Kovarianzmatrix $Cov(X)$ berechnet werden. Diese wird benötigt um die dazugehörigen Eigenvektoren mit Eigenwerten zu finden. Anschließend sind die Eigenvektoren nach absteigenden Eigenwerten zu sortieren, um l Eigenvektoren mit den größten Werten auszuwählen und die Matrix A mit $p \times k$ zu bilden. Im letzten Schritt wird mit Hilfe der Matrix W die Transformation $W = X \times A$ durchgeführt.

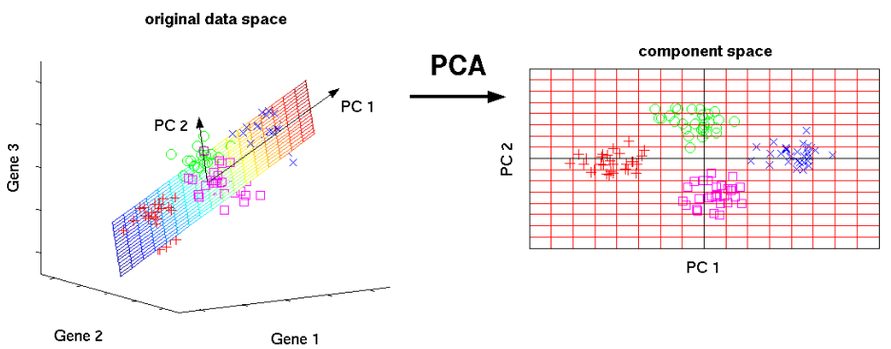


Abb. 2.1: Eine visuelle Repräsentation der ersten und zweiten PCA eines Gen-Datensatzes[4]

2.1.1.2 T-Distributed Stochastic Neighbor Embedding (t-SNE)

T-SNE ist im Gegensatz zur PCA ein nicht-linearer ML-Algorithmus, der ebenfalls zur Datenvisualisierung benutzt wird. Seine Besonderheit ist es lokale Strukturen im hochdimensionalen Datensatz bei der Filterung zu erhalten. Diese Eigenschaft ist besonders schwierig und problematisch bei linearen Algorithmen wie der PCA, die eher die globale Struktur der Daten abbilden. T-SNE verbindet hierbei also Strukturen auf großer und kleiner Ebene und hat sich damit als äußert erfolgreiche Visualisierungsmethode etabliert [2]. Diese konvertiert Ähnlichkeiten zwischen Datenpunkten in gemeinsame Wahrschein-

lichkeiten und versucht, die Kullback-Leibler-Divergenz zwischen den gemeinsamen Wahrscheinlichkeiten der niedrigdimensionalen Einbettung und der hochdimensionalen Daten zu minimieren. Zuerst wird eine Reihe $X_{(1)}, \ldots, X_{(n)}$ aus dem Datensatz X ausgewählt. Ziel des t-SNE-Algorithmus ist es, p-Dimensionen der jeweiligen Reihen des Datensatzes X in den 3-D Raum abzubilden. Der Ansatz hinter t-SNE sitzt im Stochastic Neighbor Embedding, die jede Nachbarschaft eines Datenpunktes x_i aus dem Reihenvektoren $X_{(i)} = x_1, \ldots, x_p$ mit einer Wahrscheinlichkeitsverteilung versieht[2].

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|x_i - x_k\|^2/2\sigma_i^2)} \quad und \quad p_{i|i} = 0 \tag{3}$$

Hierbei stellt $p_{j|i}$ die bedingte Wahrscheinlichkeit dar, dass ein Datenpunkt x_j einen Nachbarpunkt x_i als Nachbar auswählt. Dabei werden die Nachbarn zu ihrem Verhältnis aus der Wahrscheinlichskeitsdichte unter einer Gaußkurve bei dem Ausgangspunkt x_i zentriert. Außerdem gilt zusätzlich die Bedingung $\sum_j p_{j|i} = 1$ für alle i sowie die Definition:

$$p_{i|j} = \frac{p_{j|i} + p_{i|j}}{2N} \tag{4}$$

Es wird eine bedingte Wahrscheinlichkeitsverteilung P_i über alle anderen Datenpunkte beim Datenpunkt x_i gebildet. Die Bandbreite des Gaußverteilung σ_i wird über den Hyperparameter[4] Perplexität [5] $PP(P_i)$ bestimmt, der auf einen konstanten Wert gesetzt wird und mit der lokalen Entropie [6] $H(P_i)$ wie folgt zusammenhängt.

$$PP(P_i) := 2^{H(P_i)} = 2^{-\sum_j p_{j|i} \log_2 p_{j|i}} \tag{5}$$

Die Bandbreite wird also auf die Dichte der Daten angepasst, die zur Konsequenz führt, dass niedrige Werte von σ_i in Bereichen mit hoher Dichte in Daten verwendet werden. T-SNE bildet eine Verbindung zwischen $p_{j|i}$ und einer zu Formel 3 ähnlichen Wahrscheinlichkeitsverteilung $q_{j|i}$. Der niedrig dimensio-

[4]Hyperparameter ist ein Parameter, der für die Steuerung des Trainingsalgorithmus verwendet wird.

[5]Perplexität ein Maß dafür, wie gut eine Wahrscheinlichkeitsverteilung oder ein Wahrscheinlichkeitsmodell eine Stichprobe vorhersagt [5].

[6]Entropie ein Maß für den mittleren Informationsgehalt einer Variablen [6].

nale Raum $y_1, \ldots, y_n$ weist nun so viele Ähnlichkeiten zu $x_1, \ldots, x_n$ auf wie möglich.

$$q_{j|i} = \frac{\exp(-\|\mathrm{y}_i - \mathrm{y}_j\|^2)}{\sum_{l \neq k} \exp(-\|\mathrm{y}_k - \mathrm{y}_l\|^2))} \tag{6}$$

Wichtig für die Durchführung der Transformation ist, dass die Datenpunkte von einer Gaußverteilung auf eine studentsche t-Verteilung (q in Abbildung 2.2) durchgeführt wird, um nicht ähnliche Datenpunkte weiter voneinander abzubilden.

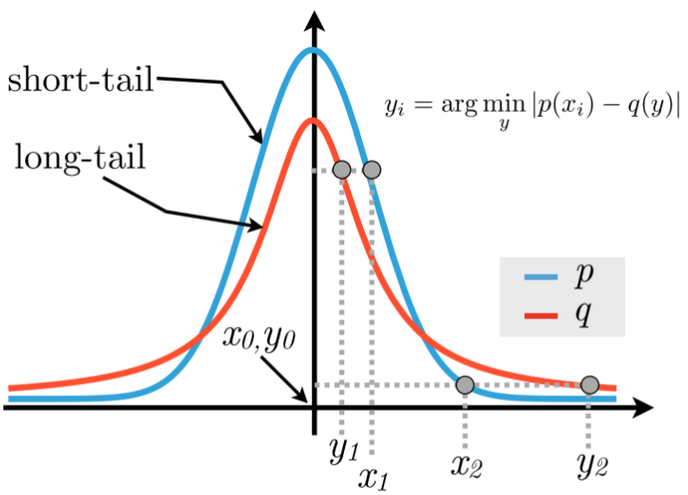


Abb. 2.2: Zu sehen ist der Datenpunkt x_i des hochdimensionalen Datensatzes sowie y_i des niedrigdimensionalen Datensatzes, der durch eine Minimierung der Bedingung $|p(x_i) - q(y)|$ berechnet wird. [2]

Um die Orte der Punkte y_i ausfindig zu machen, muss die Kullback-Leibler-Divergenz KL der Gaußverteilung (short-tail) zu der t-Verteilung (long-tail) verwendet werden. Die Minimierung der Divergenz wird mit der Methode des Gradientenverfahren durchgeführt, welches das niedrig dimensionale Mapping berechnet.

$$C(Y) = \sum_{\mathrm{i}} \mathrm{KL}(\mathrm{P_i} \mid\mid \mathrm{Q_i}) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}} \tag{7}$$

Q_i bildet die bedingte Wahrscheinlichkeit über alle anderen transformieren Datenpunkte beim Datenpunkt y_i. T-SNE besitzt eine Kostenfunktion $C(Y)$, die nicht konvex ist, d.h. mit unterschiedlichen Initialisierungen werden unterschiedliche Ergebnisse produziert [7]. Dafür können durch die Nichtlinearität des Algorithmus deutlich komplexere Zusammenhänge besser visuell dargestellt werden.

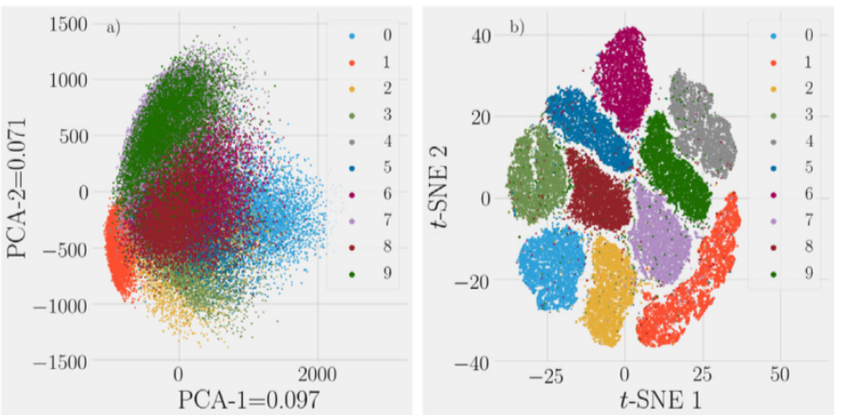


Abb. 2.3: Visualisierung des Datensatzes MNIST (a) Ersten beiden Hauptkomponenten (b) t-SNE mit Perplexität von 30 und deutlich besserer Trennung des Datensatzes [2]

2.1.1.3 Nicht-negative Matrixfaktorisierung (NMF)

„Die nicht-negative Matrixfaktorisierung ist zu einem weit verbreiteten Werkzeug für die Analyse hochdimensionaler Daten geworden, da diese automatisch spärliche und aussagekräftige Merkmale aus einem Satz nicht-negativer Datenvektoren extrahiert“ [8]. Der vorliegende Datensatz X mit $n \times p$ Matrix wird in zwei weitere Matrizen W und H zerlegt. Diese Zerlegung geschieht ähnlich wie zur PCA, nur dass die Koeffizienten der Linearkombinationen positiv sein müssen. Der Zwang nach positiven Koeffizienten führt zur additiven Kombination der NMF-Komponenten um den ursprünglichen Datensatz X abzubilden. Die Zerlegung kann nur in einer Näherung zu der ursprünglichen Matrix X zurück transformiert werden.

$$X \approx WH \tag{8}$$

Schematisch ist sich dies an dem Beispiel der Zerlegung eines 2D-Bildes vorzustellen. Dabei repräsentiert die Matrix W die Gewichte und H die Komponenten der Grundmatrix X. Die Dimensionen werden dahingehend reduziert, weil W in $(m \times l)$ und H mit $(l \times n)$ vorliegt und l eine deutlich niedrigere Dimension aufweist. Es gibt viele verschiedene Arten von NMF-Algorithmen sowie unterschiedliche Methoden zur Lösung des Problems. Um die Matrizen W und H zu finden wird ebenfalls eine Kostenfunktion C aufgestellt, die die Differenz zwischen X und WH misst. Ein Beispiel wird hier an der einfachen Frobenius Norm (Squared Error Funktion) aufgezeigt, mit der Aufgabe bei der gegebenen Matrix X die beiden

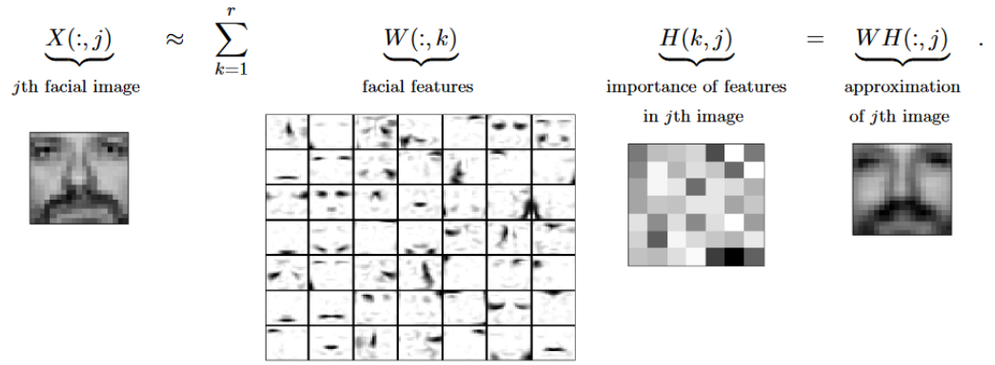


Abb. 2.4: Visuelle Zerlegung eines Gesichts aus der Datenbank CBCL [8]

zerlegten Komponenten zu finden, indem die Kostenfunktion minimiert wird.

$$C(\mathrm{W},\mathrm{H}) = \|\mathrm{X} - \mathrm{WH}\|^2 \tag{9}$$

Mit der Kostenfunktion in 9 besitzt der NMF-Algorithmus durch die Orthogonalitätsbedingung $HH^T = I$ eine grundlegende Clustereigenschaft. Um die Kostenfunktion zu lösen muss diese erneut minimiert werden, welches z.B. mit dem Verfahren der "Multiplicative Weight Update Method"von Lee und Seung's [9] durchgeführt werden kann. Im Gegensatz zu anderen Dimensionsreduzierungsmethoden ist NMF durch die additive Art der Zusammenstellung der Matrizen leichter interpretierbar.

Ein Vergleich zwischen beiden DR-Methoden wird anhand des Beispiels der Zerlegung eines Gesichts in Abbildung 2.4 gegeben. Die Algorithmen NMF und PCA lernen, ein Gesicht durch lineare Kombination von Basisbildern zu repräsentieren. Zu sehen ist, dass NMF einzelne Komponenten (Nase, Augen) herausgreift, was gut mit der intuitiven Vorstellung von Gesichtern übereinstimmt, da erstens nur additive Kombinationen in NMF erlaubt sind und dies mit der intuitiven Vorstellung der Kombination zu einem Ganzen kompatibel ist. Die PCA hingegen liefert verrauschte Komponenten, die wenig interpretierbar sind, weil erstens bei der PCA jedes Gesicht durch eine Linearkombination aller Basisbilder approximiert wird und zweitens die Einträge der faktorisierten Vektoren ein beliebiges Vorzeichen haben können[10]. Da es sich dabei in der Regel um Aufhebungen zwischen positiven und negativen Elementen handelt, haben viele Basisbilder keine intuitive Bedeutung [10]. Dieses Beispiel kann analog ebenfalls auf die Zerlegung eines XRD-Musters übersetzt werden, indem die Basisbilder die ausschlaggebenden

Peakpositionen eines Röntgendiffraktogramms darstellen.

2.1.2 Cluster Methoden

Bei der Clusteranalyse geht es darum, die vorhandenen Informationen aus einem ungelabelten Datensatz X zu nutzen, um eigenständig Labels zu generieren. Die passende Auswahl der Algorithmen sowie die Validierung sind oft nicht eindeutig bestimmbar. Um vorab Aussagen über die Qualität der Daten zu treffen, muss die zugrundeliegende Wahrscheinlichkeitsverteilung, aus denen die Daten gewonnen wurde, bekannt sein [11]. Dabei sind die Algorithmen in verschiedene Gruppen einzuteilen: partitionierende-, hierarchische- und dichtebasierte Verfahren[7].

2.1.2.1 K-Means

Der Algorithmus K-Means gehört zu den partitionierenden Methoden und ist aufgrund seiner Einfachheit eine sehr beliebte Wahl des Clusterns. Es werden verschiedene nicht überschneidbare Gruppen gesucht, die im vornherein im Parameter K (Anzahl der Cluster) genau vorgegeben werden müssen. Hierbei ist darauf zu achten, dass jeder Datenpunkt der Reihe X_i nur genau eine Gruppe zugewiesen werden kann.

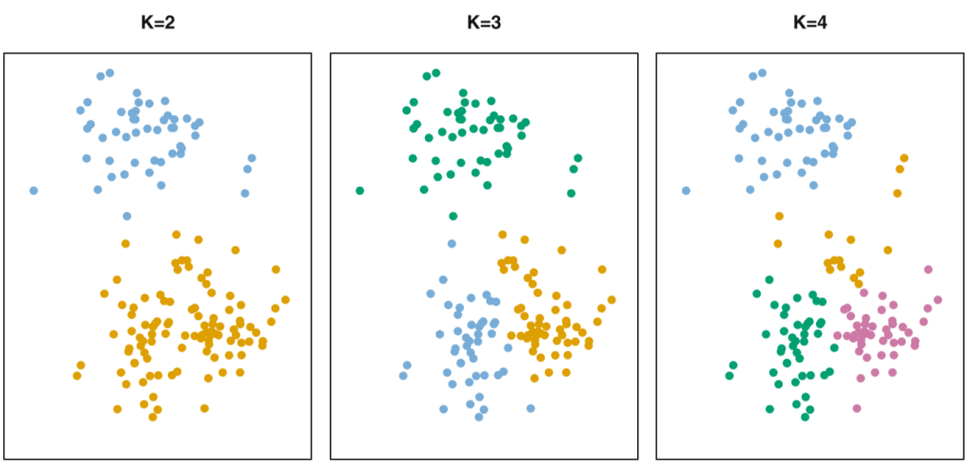


Abb. 2.5: K-Means Algorithmus mit unterschiedlicher Initialisierung der Anzahl der Gruppen [1]

In Abbildung 2.5 ist zu sehen, dass eine gewisse Kenntnis der Vorstellung der Anzahl der Cluster vorhanden sein muss. Mit der Elbow-Methode kann zwar die ungefähre Anzahl der Gruppen abgeschätzt

[7]Zu den oben genannten Methoden gibt es weitere Verfahren, die aber in dieser Arbeit keine Anwendung finden

werden, welches aber allenfalls als grobe Richtlinie zu benutzten ist.

Wir definieren Cluster $C_1, ..., C_K$, die die jeweiligen zugeordneten Datenpunkte besitzen. Ziel des Algorithmus ist es die Variation der verschiedenen Gruppen so gering wie möglich zu halten. Wenn für die Implementierung der quadratische euklidische Abstand gewählt wird, sieht die Optimierungsbedingung wie folgt aus.

$$\arg\min_{\mathrm{C}} \sum_{i=1}^{K} \sum_{\mathrm{X} \in C_i} \left\| X_{(i)} - \boldsymbol{\mu}_i \right\|^2 \tag{10}$$

Um dieses Problem zu lösen und ein lokales Optimum zu finden, kann der nachfolgende schematische Ablauf verwendet werden. Wichtig ist hierbei aber, dass nach einem Auffinden eines lokalen Minimums der Algorithmus mehrfach berechnet wird. Dies erlaubt es die Richtigkeit des aufgefundenen Minimums besser abzuschätzen.

Algorithm 1: K-Means Clustering

Result: Gruppierung des Datensatzes in k Cluster

1. Initialisierung: Zufälliges Zuordnen von 1 bis K zu jedem Datenpunkt.
2. Iteriere solange bis sich die Clusterzuordnung an einem Grenzwert kaum mehr verändert.
   (a) Berechnung des Cluster-Schwerpunktes für jeden der K-Cluster. Der k-te Cluster-Schwerpunkt ist der Vektor des p-Featuremittelwert für die Datenpunkte im k-ten Cluster.
   (b) Zuordnen eines jeden Datenpunktes zu einem Cluster, dessen Schwerpunkt am nächsten liegt (definiert durch Euklidischen Abstand).

Problematiken die bei K-Means auftreten können, sind Initialisierungen, die zu falschen lokalen Minima führen. Dieses Problem kann durch die Verwendung von K-Means++ verbessert werden. Zusätzlich wird die Annahme gleicher Clustervarianzen getroffen, was bei sehr starken Abweichungen zu unrealistischen Ergebnissen führt. Auch die Clustergrößen sollten alle in der gleichen Anzahl vorliegen, da die Gruppen immer in der Mitte zwischen zwei Clusterzentren geteilt werden.

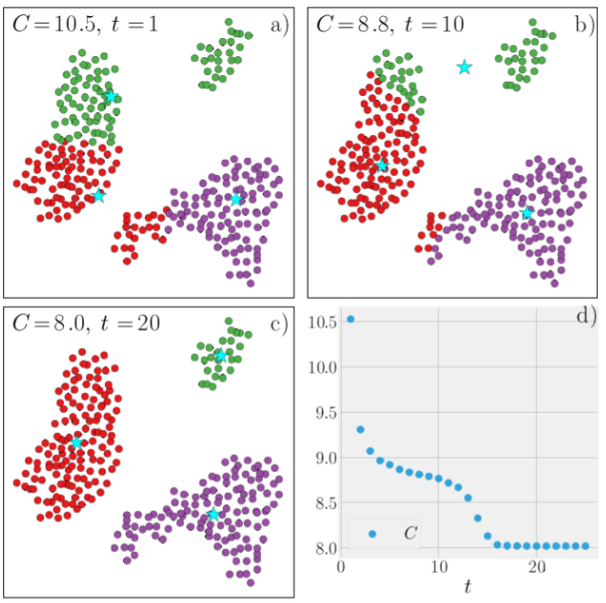


Abb. 2.6: a) (c) K-Means Ablauf mit unterschiedlicher Anzahl von Iterationsschritten t, Sterne stellen den jeweils neu berechneten Clustermittelpunkt dar, (d) minimierte Kostenfunktion C in lokalem Optimum [2]

2.1.2.2 Agglomerative Cluster Algorithmen

Um die Nachteile des K-Mean Algorithmus zu verbessern, wurde das Gebiet der agglomerativen Clusteralgorithmen erfunden. Hierbei handelt es sich wiederum um ein Teilgebiet der hierarchischen Cluster Algorithmen, die der Bottom-Up Methode gehorcht. Das bedeutet, dass jeder Datenpunkt einem Cluster zugeordnet wird und mit aufsteigender Hierarchie Cluster zusammengefügt werden. Visualisiert wird dies beim hierarchischen Clustern meist mit einem Dendogramm. Wie in Abbildung 2.7 zu sehen ist, muss durch visuelle Betrachtung eine entscheidende Grenzschwelle gewählt werden, die die Anzahl der Cluster bestimmt. Von den beginnenden Clustergruppen werden durch verschiedene Fusionierungsmethoden (Linkage) [8] Gruppen zusammengefügt, bis nur noch ein einziger Cluster übrig ist. Hierbei ist zwischen dem Distanzmaß z.B. euklidische Distanz und dem Fusionierungsalgorithmus zu unterscheiden. Das Distanzmaß berechnet den Abstand von zwei Datenpunkten, der Fusionierungsalgorithmus hingegen berechnet den Abstand zwischen zwei Clustern.

Das Ergebnis des K-Means Algorithmus kann zwar mit ausgewählten Methoden nachgebildet werden, benötigt aber durch die höhere Komplexität einen deutlich längeren Zeitraum. Daher sollte hierar-

[8] Eine Übersicht der Fusioninierungsmethoden ist im Physics Report "A high-bias, low-variance introduction to machine learning for physicists "zu finden. [2]

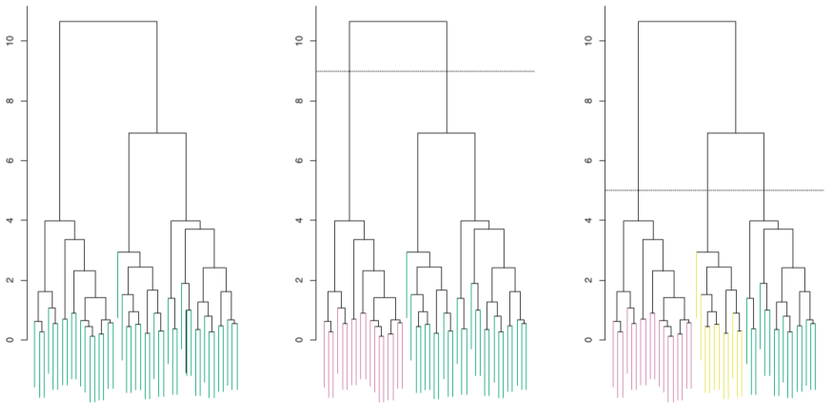


Abb. 2.7: Berechnetes Dendogramm mit Fusionierung durch Complete Linkage und euklidischer Distanz, mit unterschiedlicher Grenzschwelle werden im mittleren Bild zwei und rechts drei Clustergruppen erzeugt. [1]

chisches Clustern nur für mittel-große Datensätze verwendet werden [2]. Nachfolgend ist ein allgemeiner Ablauf eines agglomerativen, hierarchischen Clusteralgorithmus [1] zu sehen.

Algorithm 2: Agglomeratives hierarchisches Clustering

---

1. Beginnend mit n Beobachtungen und einem Maß (wie dem euklidischen Abstand) aller $\binom{n}{2} = \frac{n(n-1)}{2}$ paarweisen Unähnlichkeiten, wird jedem Datenpunkt ein eigener Cluster zugewiesen.

2. Für $i = n, n-1, ..., 2$:

   (a) Untersuchung alle paarweisen Inter-Cluster-Unähnlichkeiten zwischen den i-Clustern. Identifizierung des Cluster-Paars, das am wenigsten unähnlich ist, mit Verschmelzung der beiden Cluster. Die Unähnlichkeit zwischen diesen beiden Clustern zeigt die Höhe im Dendrogramm an, bei dem die Fusion stattfinden sollte.

   (b) Berechnen der neuen paarweisen Inter-Cluster-Unähnlichkeiten zwischen den $i-1$ verbleibenden Clustern.

2.1.2.3 Dichtebasiertes Clustering

Die vorherigen besprochenen Methoden setzen voraus, dass die Daten aus überlagernden Gaußverteilungen generiert worden sind. Deshalb können keine willkürlichen Formen von Clustern in Gruppen aufgeteilt werden. Um eine generalisierte Methode zu finden, die unabhängig von der sphärischen Form Cluster gruppieren kann, wurde das dichtebasierte Clustern eingeführt. Auch der Nachteil des Spezifizierens der Clusteranzahl ist bei den moderneren DB-Clustering Methoden nicht mehr nötig. Vielmehr hängt die Qualität der geclusterten Daten von der sogenannten Hyperparameteroptimierung ab [3].

Ein Algorithmus aus der Kategorie DB-Clustering ist der DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [12]. Es werden zwei Hyperparameter verwendet, wobei davon der Radius ϵ zur Dichtebestimmung der innerhalb liegenden Punkte vorhanden ist. Mit der Anzahl der minimalen Punkte $MinPts$ wird eine Grenze gesetzt, die bestimmt, wann ein Datenpunkt als *dicht* bezeichnet wird (ein sog. Kernobjekt). Zudem gibt es Datenpunkte, die zwar von Kernobjekten durch den Radius ϵ erreichbar sind, sich selbst aber nicht als *dicht* (dichte-erreichbare Punkte) bezeichnen lassen. Wenn Datenpunkte außerhalb von ϵ liegen, werden sie als Rauschen in eine separate Kategorie zugeordnet. Dieses Vorgehen wird in der Abbildung 2.8 nochmals anschaulich verdeutlicht.

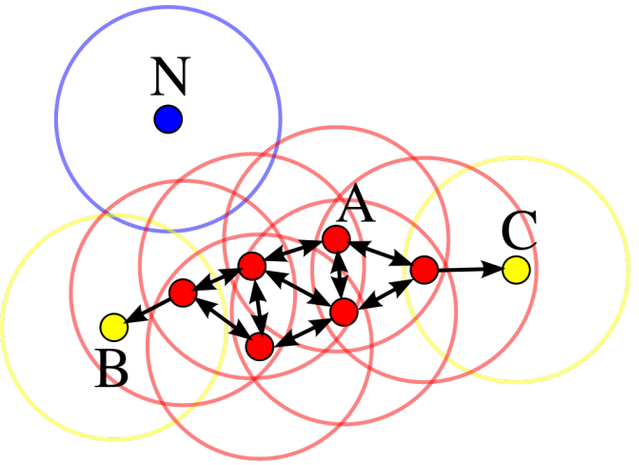


Abb. 2.8: Rote Punkte durch A gekennzeichnet sind Kernpunkte, B und C in Gelb sind dichte-erreichbare Punkte, Punkt N in blau wird als Rauschen eingeordnet [12]

Aus Abbildung 2.8 lässt sich durch die maximale Anzahl der in A eingeschlossenen Punkte der Wert $minPts = 4$ ablesen. Die Vorgehensweise kann in einem algorithmischen Schema nochmals festgehalten werden.

Algorithm 3: Allgemeiner DBSCAN [12]

---

1. Berechnen der Nachbarn jedes Punktes mit Identifizierung der Kernpunkte
2. Benachbarte Kernpunkte zu Clustern zusammenfassen
3. Für jeden Nicht-Kernpunkt:
   (a) Wenn möglich, einem benachbarten Kernpunkt hinzufügen
   (b) Sonst zum Rauschen hinzufügen

Wenn ein Datensatz sehr stark von einer sphärischen Form sowie von nicht gleich großen Clustern abweicht, kann der DBSCAN gegenüber K-Means deutliche Verbesserungen erreichen. In Abbildung 2.9 sind willkürliche Clusterformen mit K-Means und DBSCAN gruppiert. Sichtbar ist sofort, dass DBSCAN sehr gut mit verschiedenen Formen von Clustergruppen arbeiten kann. Wenn in dem zugrundeliegenden Datensatz viel Rauschen vorhanden ist, sollte die Wahl auf DBSCAN fallen, da die Rauschpunkte in einem extra Cluster untergebracht werden. Durch die vielen positiven Aspekte gegenüber K-Means darf sich der Algorithmus als einer der meist zitierten in diesem Bereich nennen.

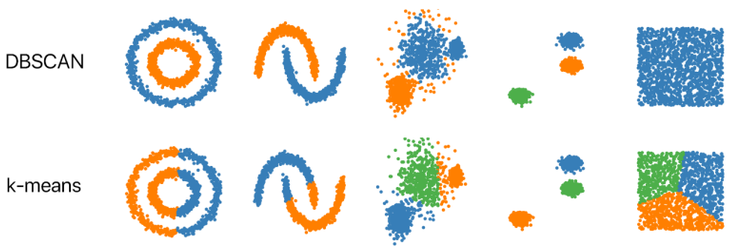


Abb. 2.9: DBSCAN und K-Means im Vergleich [12]

Allerdings sind auch einige Nachteile zu beachten, die die Leistung des Algorithmus eindeutig verschlechtern. Wenn die Dichten von verschiedenen Clustergruppen zu sehr voneinander abweichen, scheitert der Algorithmus in den meisten Fällen. Auch bei einer hohen Dimensionalität des Datensatzes ist die Qualität der Cluster schlechter, da eine genaue Bestimmung der Dichte schwieriger wird [3].

2.2 Auswahl der Deskriptoren

Bisher war die Vorgehensweise mit dem gegebenen Datensatz und seiner Gesamtanzahl der Features p, als ein Matrix X in den ML-Cluster Algorithmus hineinzugehen. Hierbei wurde einer der wichtigsten Schritte des ML übersprungen - das sogenannte Feature Engineering. Oftmals haben die bekannten Features vorerst einen sehr niedrigen Informationsgehalt über die gewünschte Vorhersage Y der Labels. Wenn der ML-Algorithmus ohne diesen Schritt durchgeführt wird, ist die Qualität der Vorhersagen meist mangelhaft. Allerdings kann durch eine explorative Datenanalyse und konkretes Fachwissen des Anwendungsgebietes das Feature Engineering erfolgen.

Da sich diese Arbeit mit dem Bereich von Atomsystemen und Kristallstrukturen beschäftigt, werden passende Deskriptoren aus entsprechenden Feldern gesucht. Ziel ist es also aus den bereits vorhandenen xyz-Kristallkoordinaten der simulierten Kristalle eine bessere Beschreibung der Struktur zu finden, sodass diese mit höherer Genauigkeit durch den ML-Algorithmus bestimmt werden kann. Dafür sind aus [13] [14] die folgenden Deskriptoren ausgesucht worden, auf die im Folgenden näher eingegangen wird. Zudem wurden weitere Deskriptoren getestet, die aber für eine Strukturerkennung unzureichend waren und deshalb in Teil III nur kurz erwähnt werden.

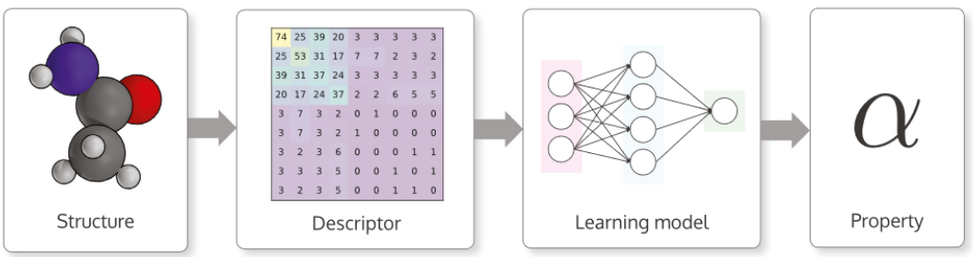


Abb. 2.10: Ablauf zwischen Kristallstruktur und ML-Modell mit der Vorhersage eines Labels (α würde in diesem Fall der Kristallphase entsprechen). [14]

2.2.1 Partial Radial Distribution Function

Um die Struktur eines Kristalls zu repräsentieren gibt es zahlreiche Möglichkeiten wie z.B. die Koordinaten der Atome, Angular Distribution Function oder Radial Distribution Function. Für einen ML-Algorithmus sind diese Repräsentationen oftmals nicht geeignet. Deshalb wurden Deskriptoren entwickelt, die mit ML-Algorithmen besser zu verarbeiten sind. Unter diesen befindet sich die sogenannte PRDF, welche auch oft als Pair Correlation Function bekannt ist und zahlreiche Anwendung in der Molekulardynamik besitzt. Die PRDF gibt die Wahrscheinlichkeit an, ein Atom in einem bestimmten Abstand r von einem Ursprungsatom aufzufinden. Die durchschnittlichen interatomischen Abstände gelten in der PRDF als Strukturinformationen des Kristalls, die über die PRDF-Peakposition sichtbar gemacht werden können. Über die Halbwertbreite (FHWM) kann die strukturelle Unordnung und durchschnittliche Koordinationszahl [9] abgelesen werden.

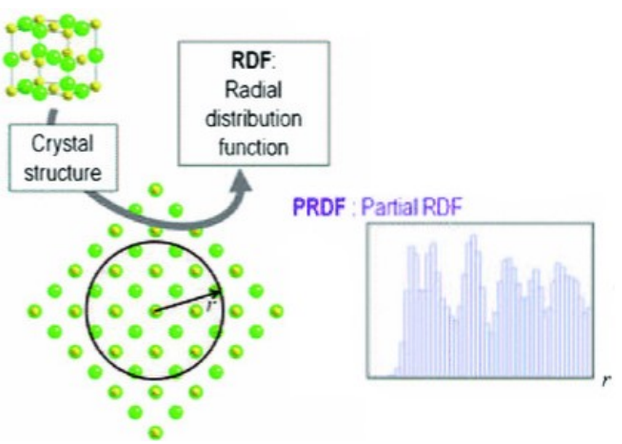


Abb. 2.11: Partial Radial Distribution Function in Histogrammdarstellung mit wählbarer Klassengröße [13]

[9] Diese gibt Auskunft über die nächsten Nachbarn einer zentralen Struktureinheit

Um zur Histogrammdarstellung zu kommen werden paarweise die Distanzen $d_{\alpha_i\beta_i}$ für die Atome α und β berechnet. Hierfür wird eine sogenannte Dichte angenommen, die sich von Typ β in Bezug auf α in einer Schale des Radius r und Breite dr befindet. Wenn dabei der Mittelwert aller Atome gewichtet wird, lässt sich der Ausdruck der PRDF [15] wie folgt schreiben:

$$g_{\alpha\beta}(r) = \frac{1}{N_\alpha V_r} \sum_{i=1}^{N_\alpha} \sum_{j=1}^{N_\beta} \delta(d_{\alpha_i\beta_i} - r)\delta(r + dr - d_{\alpha_i\beta_i}) \tag{11}$$

Der Deskriptor wird also in einer Featurematrix X, durch die PRDF-Repräsentation aller möglicher Kombinationen von spezifischen Atomen ausgedrückt. Diese Art von Deskriptor ist invariant in der Translation, Rotation und Auswahl der Einheitszelle. Durch diese Generalisierung ist diese Art von struktureller Darstellung des Kristalls eine häufig verwendete Methode. Zusammenfassend lässt sich sagen, dass mit der PRDF Details in lokalen Strukturen gut aufzudecken sind, welche durch eine herkömmliche Strukturanalyse mit einem XRD-Muster nicht zu erkennen sind [16]. Im Vergleich dazu wird bei der Pulverdiffraktometrie jeweils die mittlere kristallographische Struktur ausgewertet und in ein XRD-Muster übersetzt [17].

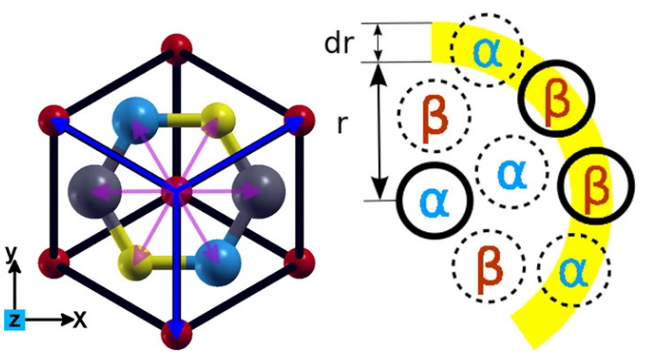


Abb. 2.12: Links: Kristall mit Basisvektoren in Pink und Bravaisvektoren in Blau, Rechts: Berechnung der PRDF $g_{\alpha\beta}(r)$ über Schale dr [15]

2.2.2 Pulverröntgendiffraktometrie

Zur herkömmlichen Phasendetektion von Kristallen ist die Analyse des Röntgendiffraktogramms die am weitesten verbreitete Methode. Hierbei wird oftmals auf die Pulveranalyse zurückgegriffen, da sie viele Probleme vereinfacht und für die Peak-Detektion ausreichend ist. In einem Röntgendiffraktogramm wird die Intensität der reflektierten Röntgenstrahlen gegenüber dem Winkel 2θ aufgetragen. Der

mathematische Formalismus befindet sich im reziproken Raum und kann durch eine Fourierrücktransformation in den realen Raum abgebildet werden. In diesem repräsentiert die gespeicherte Information die Elektronendichte und gibt damit direkte Strukturinformationen des Kristalls an [18]. Die Position und die Intensität der XRD-Peaks sind ausschlaggebende Kriterien für die Phasendetektion. Diese sind jeweils charakteristisch für eine Phase, wobei die Identifizierung durch einen Abgleich des XRD-Musters zu einem Referenzmuster geschieht.

Der physikalische Zusammenhang zwischen Winkel θ und der Wellenlänge der einfallenden Röntgenstrahlen ist wie folgt:

$$2d \sin\theta = n\lambda \tag{12}$$

Die Beziehung der Gleichung 12 zu den Kistalldimensionen wird über den Netzebenenabstand d, die Gitterkonstante a und die Millersche Indizes hkl erklärt. Für ein kubisches System kann für d und a folgendes abgeleitet werden:

$$d_{hk\ell} = \frac{a}{\sqrt{h^2 + k^2 + \ell^2}} \tag{13}$$

Durch eine Probenverschiebung kommt es bei experimentellen Daten oftmals zu dem sogenannten Peak-Shifting. Hierbei handelt es sich um einen systematischen Fehler, der durch einen Korrekturfaktor im nach hinein meist behoben werden kann. Zusätzlich kann ein Peak-Shifting durch eine erhöhte Dotierstoffkonzentration auftreten, da sich dadurch die Gitterkonstante a verändert. Ebenfalls kann es zu dem Peak-Broadening kommen, der durch Gitterfehler oder kleinere Kristallitgrößen in nanokristallinen Materialien auftritt [16].

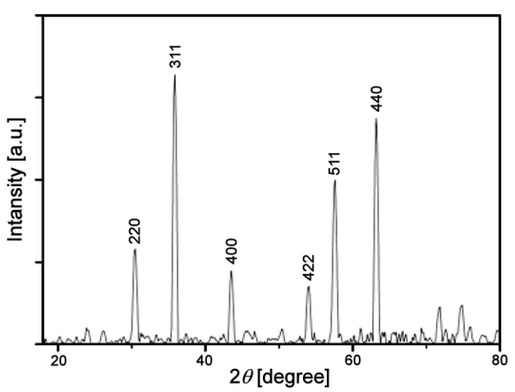


Abb. 2.13: XRD-Muster einer gesinterten Lithium-Ferrit-Probe [19]

III. Implementation und Analyse der Cluster-Algorithmen

3.1 Data Cleaning

Wie in der Arbeit [20] besprochen wurde , sind die vorhandenen Rohdaten durch Ab-initio-Berechnung auf dem SuperMUC generiert worden. Die Daten sind dabei in einem üblichen Format der Kristallographie abgelegt. Hierbei handelt es sich um das CIF (Crystallographic Information File) Format, welches in dem Programmcode 1 zu sehen ist. In diesem sind Zelllängen a, b, c, Winkel α, β, γ sowie die relativen Koordinaten der einzelnen Atome angegeben.

```
data_image0
_cell_length_a       10.1
_cell_length_b       10.2733
_cell_length_c       10.4456
_cell_angle_alpha    89.9055
_cell_angle_beta     80.7256
_cell_angle_gamma    89.8586
_symmetry_space_group_name_H-M    "P 1"
_symmetry_int_tables_number       1
loop_
  _symmetry_equiv_pos_as_xyz
  'x, y, z'
loop_
  _atom_site_label
  _atom_site_occupancy
  _atom_site_fract_x
  _atom_site_fract_y
  _atom_site_fract_z
  _atom_site_thermal_displace_type
  _atom_site_B_iso_or_equiv
  _atom_site_type_symbol
  Ag1      1.0000 0.12509  0.01454  0.16654  Biso   1.000  Ag
```

```
Hf1      1.0000 0.13878  0.23230  0.39586  Biso   1.000  Hf
Hf2      1.0000 0.35928  0.26897  0.09998  Biso   1.000  Hf
Hf3      1.0000 0.36123  0.47925  0.35267  Biso   1.000  Hf
O1       1.0000 0.04263  0.16723  0.08323  Biso   1.000  O
O2       1.0000 0.02661  0.08233  0.35055  Biso   1.000  O
O3       1.0000 0.46327  0.41914  0.16899  Biso   1.000  O
O4       1.0000 0.46299  0.33052  0.41904  Biso   1.000  O
O5       1.0000 0.27022  0.12429  0.23551  Biso   1.000  O
...
```

Listing 1: CIF Beispiel aus dem Rohdatensatz

Simuliert wurden 3450 Kristalle, wobei davon die Hälfte aus Hafniumoxid und die andere Hälfte aus Zirkonoxid besteht. In diese Verbindungen mit unterschiedlichen Zellgrößen wurden 58 verschiedene Dotanten [10] eingebracht. Für jeden Dotanten in unterschiedlichen Positionen (substitutionell oder interstitiell) wurden die Phasen monoklin, orthorhombisch, polar-orthorhombisch und tetragonal simuliert.

Aus diesen simulierten Kristallen können durch den Prozess des Feature Extraction die Informationen in Tabelle 3.1 gespeichert werden. Wenn das CIF-File extrahiert und in einen Datentyp «Dataframe» der Python Bibliothek Pandas [11] gespeichert wird, dient sich diese Repräsentation besonders gut um ML anzuwenden.

	probe	a	b	c	α	β	γ
0	Ag_HfO2_cat_3.125_222_m	10.1	10.2733	10.4456	89.9055	80.7256	89.8586
1	Ag_HfO2_cat_3.125_222_o	10.0193	10.2454	10.4346	89.8294	90.3988	89.8634
2	Ag_HfO2_cat_3.125_222_p-o	9.94732	10.3706	10.007	90.07239	89.5534	89.9541
3	Ag_HfO2_cat_3.125_222_t	10.022	10.2129	10.022	89.9967	89.1962	90.0033
...	...	...	...	...	...	...	
3446	Zr_ZrO2_inter_6.25_221_m	10.2991	10.374	5.358	90.2049	78.8053	89.7413
3447	Zr_ZrO2_inter_6.25_221_o	10.1467	10.3513	5.34184	89.83559	91.48459	90.0146
3448	Zr_ZrO2_inter_6.25_221_p-o	10.1418	10.1418	5.13816	89.9999	89.99960	90.0
3449	Zr_ZrO2_inter_6.25_221_t	10.1418	10.1418	5.13819	90.0	90.0	90.0

Tabelle 3.1: Dataframe Format (3450 Reihen x 7 Spalten) mit Features der Zellenlänge und Zellenwinkel zu verschiedenen Kristallen

[10] Eine vollständige Liste der Dotanten befindet sich um Anhang A A.1

[11] Pandas ist eine Bibliothek der Programmiersprache Python zur Datenanalyse, sowie Aufbereitung und Manipulation. [21] [22]

In der Spalte Probe sind die allgemeinen Informationen, in der Reihenfolge Dotand, Verbindung, Dotierart, Atomprozentzahl des Dotanden und Zellengröße [12] des Kristalls extrahiert worden. Wenn die Tabelle 3.1 genauer betrachtet wird, lässt sich aus den Features sofort erkennen, dass die monokline Phase für β jeweils $80°$ beträgt. Diese sollte mit einem Algorithmus also sofort erkannt und richtig eingeordnet werden können. Für die Phasen orthorhombisch, polar-orthorhombisch und tetragonal hingegen verhalten sich alle Werte der Winkel um den Mittelwert $90°$. Daher müssen in nachfolgenden Tests passende Deskriptoren gefunden werden, um die Phasen aller Kristalle besser zu erkennen. (siehe Abschnitt 2.2)

3.2 Funktionale Datenanalyse

Zuerst wird eine Methode der funktionalen Datenanalyse ausprobiert, mit dem Gedanken die XRD-Muster auf Ähnlichkeit der Funktionskurven zu gruppieren. Hierbei ist der Vorteil vorhanden, sich die XRD-Muster der unterschiedlichen Phasen graphisch visualisieren zu lassen. Dafür wird die Python-Bibliothek «scikit-fda»[13] als Visualisierungstool benutzt. Um das theoretische Röntgendiffraktogramm zu berechnen wird ein weiteres Tool «scikit-ued» [14] angewandt. Um von einem simulierten Kristall das zugehörige XRD Muster zu bekommen, muss aus dem CIF Format ein Kristallobjekt der Biblothek ASE gemacht werden. Das Atomic Simulation Environment (ASE) ist eine der meistgenutzten Werkzeuge und Python-Module zum Visualisieren und Analysieren von atomischen Simulationen [23]. Das Objekt «Atoms» in ASE kann ein einzelnes Atom oder eine periodische Struktur darstellen. Zuerst werden die Bibliotheken im einem Pythonskript importiert und dazugehörige Klassen geladen.

```
from ase.io import read
from ase.visualize import view
from ase.spacegroup import crystal # Kristallobjekt in ASE
from crystals import Crystal # Kristallobjekt für XRD-Muster
import matplotlib.pyplot as plt
import numpy as np
from skued import powdersim # XRD-Simulation
```

[12] Zum Beispiel bedeutet die Zellengröße 221: $10\mathring{A} \times 10\mathring{A} \times 5\mathring{A}$

[13] Mehr Infos unter: scikit-fda: Functional Data Analysis in Python

[14] Scikit-ued ist ein open-source Python Package für Datenanalyse und Modellierung von (ultraschneller) Elektronenbeugung

```
from pathlib import Path
```

Listing 2: Bibliotheken zum Visualisieren des Kristalls sowie Simulation des XRD-Musters

In einer Funktion `view_crystal` wird das CIF Format übergeben und in ein Atomobjekt von ASE umgewandelt.

```
def view_crystal(cif_path):
    # visualize the crystal
    s = read(cif_path)
    view(s, block=True)
```

Listing 3: Visualisierung der Abbildung 3.1

Beispielsweise lässt sich die erste Reihe der Tabelle 3.1 über das CIF Format aus Listing 1 wie folgt in 3D darstellen:

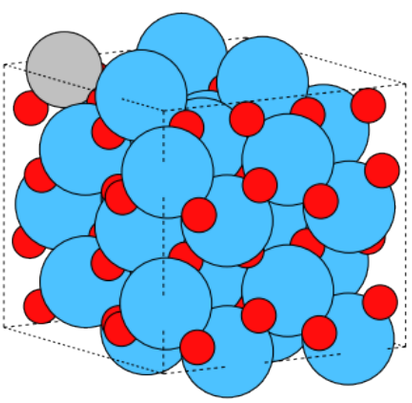


Abb. 3.1: Hafniumoxid ($AgHf_{31}O_{64}$) mit Ag als Dotant, Einheitszelle 222 und monokliner Phase, Hafnium (blau), Sauerstoff (rot)

Um nun diesen Kristall zu einem XRD-Muster umzuwandeln, wird eine weitere Funktion `pow_sim` geschrieben. Im Gegensatz zu wie im Abschnitt 2.2.2 besprochen wird auf der X-Achse die Größe q mit Einheit $\frac{1}{\AA}$ aufgetragen. Parameter, die übergeben werden, sind das Kristallobjekt, der Bereich der Streuvektornorm q und die Halbwertsbreite des gaußschen und lorentzianischen Teils des Voigt-Profils. Zurückgegeben werden die Intensitätswerte, die anschließend mit dem maximalen Intensitätswert normiert werden.

```
def pow_sim(path):
    filename = Path(path).name
```

```
3
4     # load cif file to an ase object
5     cry = read(path)
6
7     # crystal object for powdersim in skued lib
8     crys = Crystal.from_ase(cry)
9
10    # powder simulation with range of q
11    q = np.linspace(1, 7, 512)
12    diff = powdersim(crys, q) # fwhm_g=0.03, fwhm_l=0.06 Standard
13
14    # matplotlib config
15    plt.figure()
16    plt.plot(q, diff/diff.max(), '-b', label=filename, alpha= 0.3)
17    plt.legend()
18    plt.xlim([q.min(), q.max()])
19    plt.xlabel('$q (1/\AA)$')
20    plt.ylabel('Diffracted intensity (A.u.)')
21    plt.title('Crystal diffraction ')
22    plt.show()
```

Listing 4: Skript zur Simulation des Beugungsmuster an Kristallen

Der Vektor q wird je nach dem gewünschten Ausschnitt des Diffraktogramms ausgewählt. Für die Simulation wird ein Gaußprofil mit einer Lorenzkurve gefaltet, die zur Ausgleichsrechnung der Diffraktogrammkurven benutzt wird. Die in Abbildung 3.2 zu sehende theoretische Kurve zeigt die charakteristischen Peaks einer monoklinen Kristallphase. Ziel ist es nun eine passende Vektorlänge q zu finden, die die wichtigsten Informationen in das Spektrum abbildet. Daraufhin wird von jedem Kristall ein Spektrum simuliert und diese im Gesamten als funktionelle Daten geclustert.

3.2.1 Simulations- und Clusterergebnisse der FDA

Zur besseren Übersicht wird der Datensatz zuerst in HfO_2 und und ZrO_2 aufgeteilt. Dies kann hierbei ohne Genauigkeitsverluste des Cluster-Algorithmus gemacht werden, da diese nicht von der Gesamtmenge der Datenanzahl abhängt. Wenn mit funktionalen Daten gearbeitet wird, bietet sich eine angepasste Version des K-Means Algorithmus an, der ebenfalls über die Bibliothek «scikit-fda» vorhanden ist. Um

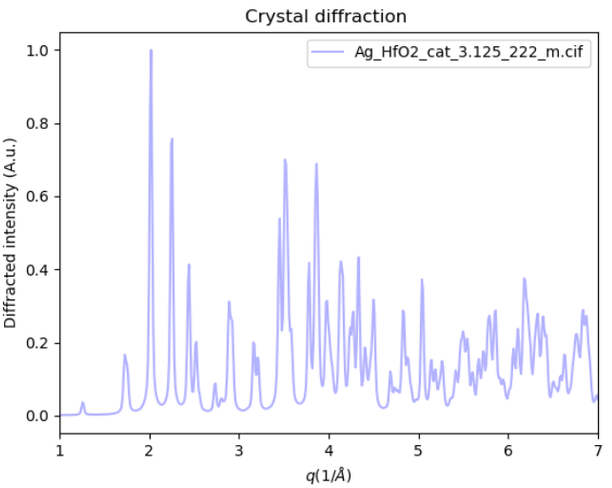


Abb. 3.2: Beugungsdiffraktogramm eines HfO_2 Kristalls, Dotant substitutionell Ag, Zelle 222, monoklin

das simulierte Spektrum in ein passendes Format zu bringen, wird ein «FDataGrid-Objekt» [15] benötigt, welchem der q Vektor und Intensitätswerte übergeben werden. Das Spektrum kann anschließend noch mit einem Kernel [16] geglättet werden, damit Ausreiser und Streuung der Daten weniger stark ins Gewicht fallen. Zuletzt wird der K-Mean-Algorithmus auf die geglätteten Kurven angewandt. Hierbei muss die Anzahl der gewünschten Cluster übergeben werden.

```
1  def kmeans_algo(knn_fd, n_clusters, savefile):
2      # Computes n clusters and labels for time series data
3
4      kmeans = KMeans(n_clusters=n_clusters)
5      kmeans.fit(knn_fd) # knn_fd (datagrid) smoothed time series data.
6      pred_labels = kmeans.predict(knn_fd)
7      np.savetxt(savefile, pred_labels, fmt='%f')
8      print(pred_labels)
9      plot_clusters(kmeans, knn_fd)
10     plt.show()
```

Listing 5: K-Means-Clustering mit Visualisierung

Es wird ein Testversuch mit 4 Clustern gestartet, da sich insgesamt mindestens 4 bekannte Phasen im Datensatz befinden sollen. Das Ergebnis ist in Abbildung 3.4 zu finden. Hierbei fällt sofort auf, dass

[15] Siehe Anhang B Listing 13, Zeile 52-55
[16] Implementation durch eine Funktion smooth_datagrid ist im Anhang B, Listing 13 zu finden.

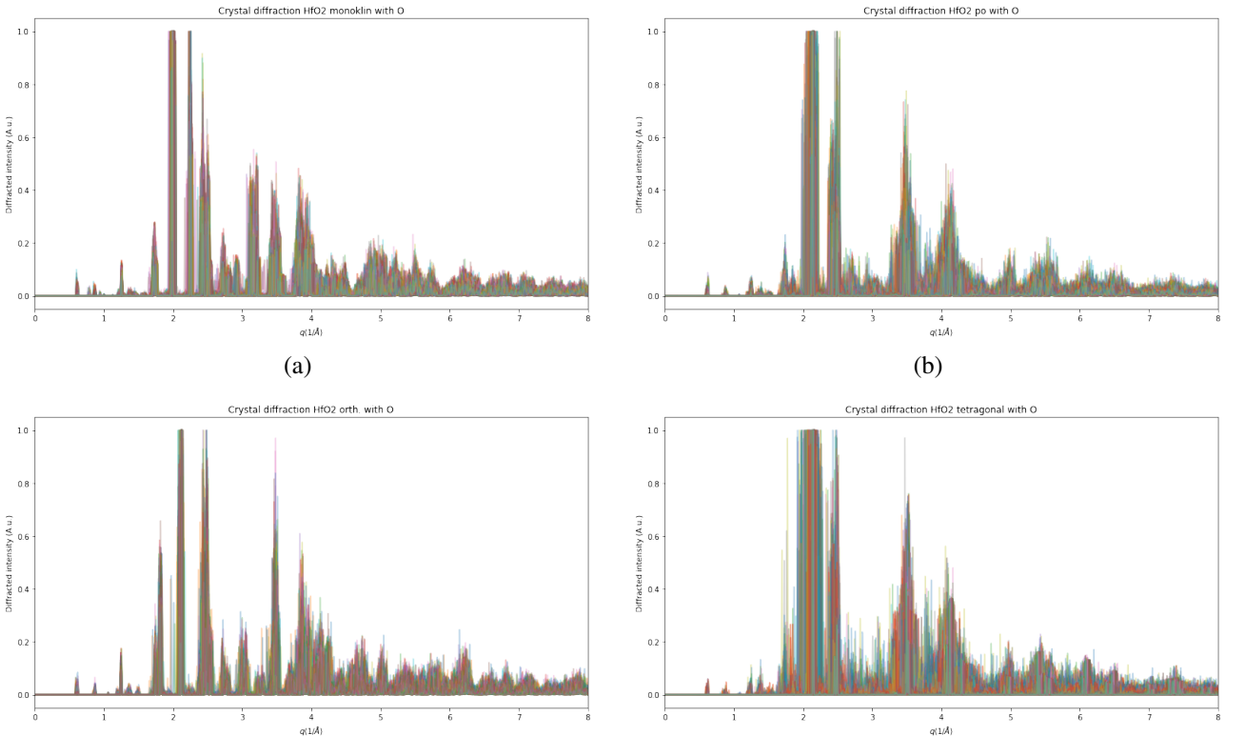


Abb. 3.3: Simulierte XRD-Muster für die 4 verschiedenen Phasen (a) m, (b) p-o, (c) o, (d) t des Hafniumoxid-Datensatzes, Vergrößerung der Bilder sind im Anhang zu finden. A A.1

die XRD-Muster der tetragonalen und polar-orthorhombischen Phase sich kaum unterscheiden. Der Cluster Algorithmus kann diese daher nicht auseinanderhalten und weist sie zufälligen Gruppen zu. Diese Ähnlichkeit der XRD-Muster kann in Abbildung 3.3 in (b) und (d) genauer betrachtet werden. Zusätzlich ist ebenfalls ersichtlich, dass die XRD-Muster der anderen Phasen visuell deutliche Unterschiede aufweisen.

Für eine schnelle Validierung der Ergebnisse der Clustergruppen, werden zum Teil gelabelte Daten verwendet, die von einer vorausgehenden Doktorarbeit gesammelt wurden. Die Phasen mit hoher Wahrscheinlichkeit zur Transformation sind dabei manuell ausgewählt worden und mit einem Label „Transformiert“ und „Nicht-Transformiert“ versehen. Alle übrig geblieben Phasen wurden mit der bereits bestehenden Phase gelabelt, da diese sich nicht verändert hat. Diese Labels sind dennoch nur als grobe Abschätzung zu verwenden und repräsentieren nicht die sogenannten Ground-Truth-Labels. Sie dienen im weiteren Verlauf der Arbeit nur als Referenz und einfache Validierungsmethode, die zusätzlich weiteren Methoden unterworfen werden muss. Diese Labels sind ebenfalls einer Aufbereitung (Abschnitt 3.1)

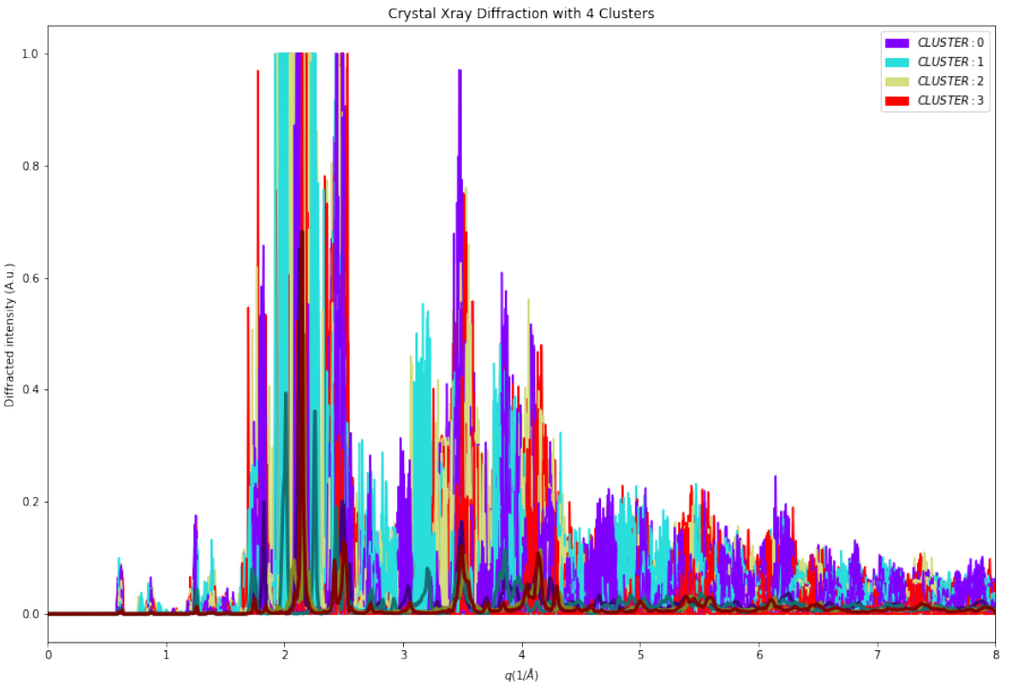


Abb. 3.4: XRD-Muster mit K-Means-Clustering Algorithmus in 4 Phasen, m (Cluster 1), o (Cluster 0), p-o (Cluster 3), t (Cluster 2)

unterzogen worden, die passend zu dem Datensatz HfO_2 und ZrO_2 gefiltert ist. Es wurden unwichtige Features entfernt und die Kennung „Probe“ für die Identifizierung nachgebildet. Daran lässt sich der Datensatz sehr einfach sortieren, sodass Fehler bei der Zuordnung zwischen Labels und Features vermieden werden. Ein Ausschnitt davon ist in Tabelle 3.2 zu sehen.

	ID	new_labels
0	Ag_HfO2_cat_3.125_222_m	0
1	Ag_HfO2_cat_3.125_222_o	2
2	Ag_HfO2_cat_3.125_222_p-o	1
3	Ag_HfO2_cat_3.125_222_t	3
...	...	...
3444	Zr_ZrO2_inter_6.25_212_p-o	4
3445	Zr_ZrO2_inter_6.25_212_t	3
3446	Zr_ZrO2_inter_6.25_221_m	0
3447	Zr_ZrO2_inter_6.25_221_o	2
3448	Zr_ZrO2_inter_6.25_221_p-o	4
3449	Zr_ZrO2_inter_6.25_221_t	3

Tabelle 3.2: Dataframe Format (3450 Reihen x 2 Spalten) mit voraussichtlichen Labels der Phasen zu verschiedenen Kristallen, labels_cleaning.ipynb

Die „new_labels“ Spalte ist wie folgt sortiert: monoklin 0, polar-orthorhombisch 1, orthorhombisch 2, tetragonal 3, „unbekannte Phase“ 4. In der Tabelle sieht man zum Beispiel, dass bei den Daten mit Index 3444 und 3448 eine vermutliche Phasenveränderung stattgefunden hat. Diese Labels werden nun zusammen mit dem K-Means-Algorithmus verwendet und die Ergebnisse in einer Konfusionsmatrix dargestellt. Vorerst wird aber das unbekannte Label ausgeblendet und ein Durchlauf mit vier Clustern gestartet.

	o (0)	m (1)	t (2)	p-o (3)
o (0)	325	0	0	20
m (1)	0	456	4	0
t (2)	52	11	263	134
p-o (3)	58	1	95	306

Tabelle 3.3: Konfusionsmatrix-XRD mit Cluster Labels in Spalten aufgetragen, tatsächliche Labels in Reihen

Aus der Tabelle 3.3 kann die Erkenntnis gewonnen werden, dass die tetragonale und polar-orthorhombische Phase in XRD-Muster erneut deutlich überlappen. Für die monokline Phase werden 456 Kristalle richtig eingeordnet. Vier tetragonale Phasen werden dagegen der monoklinen zugeschrieben. In der Spalte p-o und t hingegen ist eine richtige Zuordnung nicht möglich. Deshalb wird vorerst das XRD-Muster der t und p-o Phasen weitergehend analysiert, um Möglichkeiten zu finden diese besser zu separieren.

Da bei der herkömmlichen X-Ray Diffraktometrie die Intensität eine Funktion der Kernladungszahl Z ist, kann der Sauerstoff nicht hinzureichend detektiert werden. Hafnium und Zirkonium, besitzen eine wesentlich höhere Anzahl an Elektronen und tragen damit deutlich stärker zum mittleren Intensitätsverlauf bei. Die beiden Phasen t und p-o weisen auf fundamentaler Ebene sehr große Ähnlichkeiten auf und können deshalb nicht getrennt werden. Eine Möglichkeit wäre es, von der Röntgendiffraktometrie wegzugehen und zu der deutlich feiner auflösbaren Neutronenstreuung zu wechseln. Auch können in den jeweiligen CIF Formaten einer Ab-initio Rechnung die Elemente nachträglich manipuliert werden, um eine bessere Auflösung der Phasen zu erzeugen. Zum Beispiel ist es möglich den Sauerstoff mit einem höherwertigen Element auszutauschen, sodass über die entsprechende Kernladungszahl Z weitere Informationen zu gewinnen sind.

Daraufhin wird Neutronenstreuung mit dem Datensatz HfO_2 ausprobiert, welches eine leichte Ver-

besserung zu den Röntgenstrahlen in der Konfusionsmatrix zeigt [17]. Mit dem Verändern des Sauerstoffs an der jeweiligen Position, durch ein höherwertiges Element (z.B. Hafnium oder Zirkon), wurde ebenfalls eine zufriedenstellende Verbesserung und deutlichere Trennung der XRD-Muster erreicht.

	o (0)	m (1)	t (2)	p-o (3)
o (0)	318	0	27	0
m (1)	0	457	3	0
t (2)	12	5	430	13
p-o (3)	13	1	141	305

Tabelle 3.4: Konfusionsmatrix mit Cluster Labels in Spalten aufgetragen, tatsächliche Labels in Reihen

Die Konfusionsmatrix in Tabelle 3.4 kann im Vergleich zu der XRD-Matrix eine leichte Verbesserung aufzeigen. Die Richtig-Positiven Elemente [18] der Matrix können für das tetragonale Label von 263 auf 430 verbessert werden.

3.2.2 FDA mit zusätzlicher unbekannter Phase

Nun werden die unbekannten Phasen ebenfalls hinzugenommen und ein weiteres Mal der Cluster-Algorithmus K-Means angewandt. Nach der Berechnung des Durchlaufs ergibt sich eine 5x5 Konfusionsmatrix (Tabelle 3.5). Problematisch bei der Validierung mit 5 Clustern, ist jedoch die Überprüfung und Zuordnung der Labelnummern. Zu diesem Zeitpunkt muss durch eine visuelle Betrachtung des Musters erkannt werden, welche die unbekannte Phase ist. Dies ist in meisten Fällen nicht sofort ersichtlich und kann nur durch Untersuchungen der einzelnen Neutronenspektren erfolgen. Im weiteren Verlauf dieser Arbeit werden alternative Methoden kennengelernt, die dennoch eine bessere Reproduzierbarkeit und Genauigkeit hervorrufen.

Die Matrix kann wie folgt interpretiert werden. Die Kristallphasen orthorhombisch und monoklin zeigen nur ein geringes Verlangen ihre Phase wechseln zu wollen. Durch den Algorithmus wurden 76 Kristallphasen der ursprünglich orthorhombischen Phase zu der unbekannten Phase eingeordnet. Hierbei ist zu bedenken, dass bereits vorweg eine Einordnung in „Unbekannt“ und der weiteren möglichen 4 Phasen stattgefunden hat. Dabei sind 185 Phasen richtig zu der veränderten Phase zugeteilt worden. Auffällig ist jedoch die polar-orthorhombische Phase, bei der zusätzlich 152 Phasen zu der unbekannten

[17] Ergebnisse hierzu lassen sich im Anhang A A.2 und A.3 oder online im Github Repository unter Neutron_Scattering.ipynb finden.

[18] Elemente auf der Hauptdiagonalen der Matrix.

	o (0)	m (1)	t (2)	p-o (3)	unbekannt (4)
o (0)	262	0	7	0	76
m (1)	0	449	0	0	11
t (2)	0	0	218	2	25
p-o (3)	0	0	3	249	152
unbekannt (4)	3	2	72	9	185

Tabelle 3.5: Konfusionsmatrix mit 5 Cluster Labels in Spalten aufgetragen, tatsächliche Labels in Reihen

Phase gruppiert worden sind. Dies könnte sich dennoch um einen Fehler handeln, der aufgrund der nicht klar trennbaren Clustergruppen auftritt. Wichtig ist also die visuelle Betrachtung der Cluster in einer 2D oder 3D Ebene, um zusätzlich eine bessere Übersicht der Gruppen zu erlangen. Daher wird sich dafür entschieden, von der funktionalen Betrachtung der Streuungsspektren wegzugehen und diese vorweg mit Dimensionsreduzierung auf die wichtigsten Bestandteile zu reduzieren.

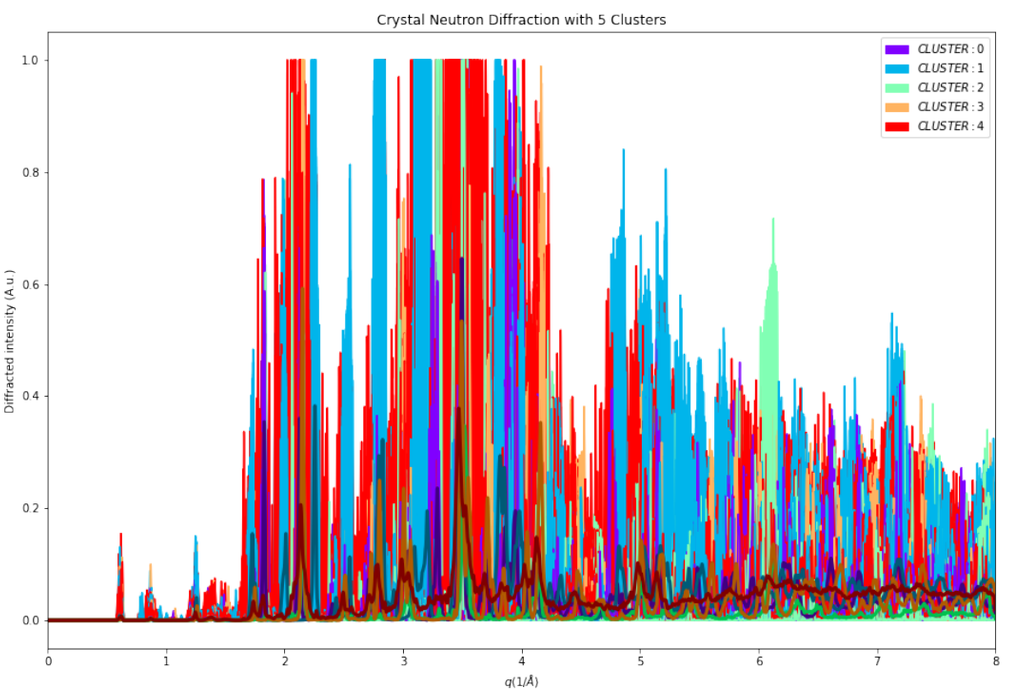


Abb. 3.5: Neutronenstreuung mit K-Means Clustering Algorithmus in 5 Phasen, m (Cluster 1), o (Cluster 0), po (Cluster 3), t (Cluster 2), unbekannt (Cluster 4)

3.3 Feature Engineering mit Deskriptoren

Im nächsten Abschnitt werden alternative Python-Bibliotheken ausprobiert, um verschiedene Deskriptoren aufzubauen. Hierunter befindet sich die Bibliothek «Matminer» [24] mit der eine Vielzahl an Deskriptoren im Bereich der Materialwissenschaften schnell und einfach kreiert werden können. Von diesen wurden die Partial Radial Distribution Function, Sine-Coulomb Matrix und XRD-PowderPattern getestet. Mit der Sine-Coulomb Matrix, die sich ebenfalls für einen Strukturdeskriptor eignet, konnte dennoch keine Phasenerkennung stattfinden. Zusätzlich wird ebenfalls das gesamte Neutronenspektrum als Deskriptor getestet, wie es bereits in FDA Abschnitt 3.2 angewandt wurde.

Um die Kristallstruktur aus dem CIF Format kompatibel mit der Matminer Bibliothek zu machen, muss die ASE Kristallstruktur in einen eigenen Datentyp mit dem Namen «Structure» umgewandelt werden. Eine Durchführung dieser Umwandlung kann im Anhang B, Listing 14 gefunden werden. Wenn der gespeicherte Dataframe geladen wird, sieht dieser wie folgt aus:

	structure	name
0	[[1.38867986 2.3864784 4.36482979] Hf, [3.587...	Ag_HfO2_cat_3.125_222_m.cif
1	[[1.40181386 2.3944349 4.3885007] Hf, [3.606...	Ag_HfO2_cat_3.125_222_o.cif
2	[[-8.57055872 -7.98390112 -6.2853237] Hf, [-6...	Ag_HfO2_cat_3.125_222_p-o.cif
3	[[-8.73105937 2.58337834 3.5977214] Hf, [-6...	Ag_HfO2_cat_3.125_222_t.cif
...	...	...
1721	[[1.2262443 -10.12402918 -3.43749088] Hf, ...	Zr_HfO2_inter_6.25_221_m.cif
1722	[[1.39997305 0.20795941 1.85144277] Hf, [1.451...	Zr_HfO2_inter_6.25_221_o.cif
1723	[[-8.97250742 -0.20890362 1.50328354] Hf, [-8...	Zr_HfO2_inter_6.25_221_p-o.cif
1724	[[1.26076659 9.90848891 1.27892142] Hf, [1.260...	Zr_HfO2_inter_6.25_221_t.cif

Tabelle 3.6: Pymatgen Composition Objekt in Strukturspalte für Matminer Bibliothek mit HfO_2 Daten

Mit diesem Structure Format kann nun in Jupyter Notebooks ein Deskriptor schnell und einfach aufgebaut werden, der automatisch jede einzelne Reihe mit einer Kristallstruktur umwandelt. Dies wird am Beispiel der PRDF gezeigt, die bereits in Abschnitt 2.2 eingeführt wurde. Bei der PRDF müssen als Parameter der Radius, die Bin-Size, beinhaltete und nicht beinhaltete Elemente übergeben werden. Eine Liste der Dotanten A, Tabelle A.1 wird geladen und in diese Gruppen aufgeteilt. Für die beinhalteten Elemente werden Hafnium und Sauerstoff ausgewählt, da diese in den Kristallen gleichmäßig verteilt vorkommen und so die Struktur gut wiedergeben.

```
1 from matminer.featurizers.structure import PartialRadialDistributionFunction
```

```
2
3 prdf = PartialRadialDistributionFunction(cutoff = 15,
4                                          bin_size = 0.5,
5                                          include_elems=['Hf', 'O'],
6                                          exclude_elems=dotant)
7 df = prdf.fit_featurize_dataframe(df, 'structure')
```

Listing 6: Build PRDF in Matminer

Die Dimension des Ergebnis aus Listing 6 kann über die Variable „cutoff“ bestimmt werden, da diese direkt mit der Anzahl der gebildeten Paare zusammenhängt. Bei mehreren Tests und Durchläufen stellt sich heraus, dass ein „cutoff“ von 15 mit „bin_size“ 0.5 die besten Ergebnisse liefert. Hierbei ist auch darauf zu achten, dass alle Kristalle über die verschieden großen Einheitszellen abgebildet werden können. Sobald der Radius über den Rand der Zelle hinausgeht, erweitert der Algorithmus die Zelle automatisch mit sich selbst.

	Hf-Hf r=0.0-0.5	Hf-Hf r=0.5-1.0	Hf-Hf r=1.0-1.5	Hf-Hf r=1.5-2.0	Hf-Hf r=2.0-2.5	...
0	0.0	0.0	0.000000	0.000000	0.000000	...
1	0.0	0.0	0.000000	0.000000	0.000000	...
2	0.0	0.0	0.000000	0.001087	0.160830	...
3	0.0	0.0	0.000000	0.000000	0.168740	...
...	...	...	...	...	...	
1722	0.0	0.0	0.000000	0.000000	0.000000	...
1723	0.0	0.0	0.000000	0.008603	0.153937	...
1724	0.0	0.0	0.000000	0.000000	0.177419	...

Tabelle 3.7: PRDF Deskriptor mit cutoff = 15, bin_size = 0.5, 1725 Reihen $\times$ 92 Spalten, HfO_2 Daten

Die Tabelle 3.7 besitzt nun Paare aus Hf und O in verschiedenen Kombinationen zu jeweils unterschiedlichen Radien. Insgesamt sind mit dem Radius 15 $\mathring{A}$ damit 90 Kombinationspaare (Hf-Hf, Hf-O, O-O) möglich.

Analog kann dasselbe Verfahren zum Erzeugen eines Deskriptors auch mit dem XRD-Muster angewandt werden, wobei erneut für jede Reihe des Dataframes, über einen Bereich von 2θ die Intensität berechnet wird.

```python
from matminer.featurizers.structure import XRDPowderPattern
xrd = XRDPowderPattern(two_theta_range=(5, 60)) # a small range is chosen
df = xrd.fit_featurize_dataframe(df, 'structure')
```

Listing 7: Build XRD-Diffraktion in Matminer

Im Gegensatz zu der vorherigen Methode kann in dieser Bibliothek der Winkelverlauf nur in $1°$ Schritten durchzogen werden. Die in Abschnitt 3.2 erzeugten XRD- und Neutronendiffraktogramme haben im Vergleich zu diesem Deskriptor eine sehr hohe Auflösung.

	xrd_0	xrd_1	xrd_2	xrd_3	xrd_4	...
0	1.922885e-124	3.662136e-106	2.274709e-89	4.608160e-74	3.044665e-60	...
1	3.833496e-120	2.186820e-102	4.377218e-86	3.074328e-71	7.576510e-58	...
2	4.614158e-144	6.228116e-123	1.627338e-103	8.231053e-86	8.059163e-70	...
3	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	...
...	...	...	...	...		
1722	1.628877e-14	1.055123e-09	2.353980e-06	1.808785e-04	4.786917e-04	...
1723	5.637057e-15	8.154305e-10	3.094216e-06	3.079948e-04	8.042024e-04	...
1724	1.052662e-19	2.655485e-12	3.598342e-07	2.619174e-04	1.024070e-03	...

Tabelle 3.8: XRD Deskriptor mit two_theta_range (5, 60), 1725 Reihen $\times$ 56 Spalten, HfO_2 Daten

3.4 Clusteranalyse der ausgewählten Deskriptoren

In diesem Abschnitt werden nun Clusteranalysen mit verschiedenen Dimensionsreduktionsmethoden durchgeführt. Dabei sollen Röntgenstreuung (feine und grobe Auflösung), Neutronenstreuung sowie die PRDF getestet werden.

3.4.1 XRD-Muster mit Dimensionsreduzierung und Cluster-Algorithmen

Der Datensatz der Röntgendiffraktometrie aus Abschnitt 3.2 mit 1725 Reihen und 16.000 Spalten wird geladen. Im ersten Schritt wird die sogenannte Elbow-Methode angewandt, die zur Bestimmung der Anzahl der Cluster zur Verwendung kommt. Damit kann ebenfalls die Hypothese überprüft werden, ob tatsächlich 5 Clustergruppen im Datensatz vorhanden sind.

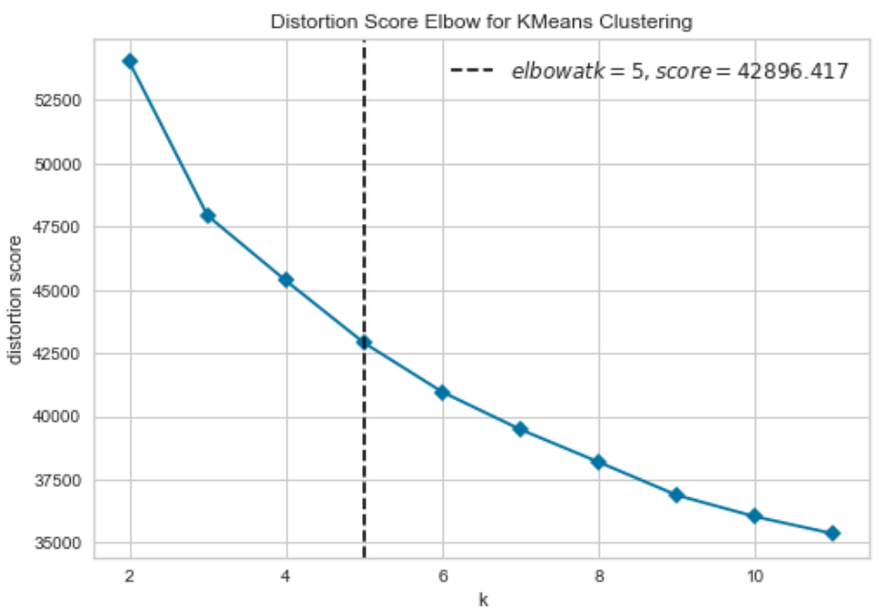


Abb. 3.6: Elbow-Plot mit feinem XRD-Muster

In Abbildung 3.6 ist auf der y-Achse der Wert „Distortion“ aufgetragen, welcher über die Summe der quadrierten Abstände von jedem Punkt zu seinem zugeordneten Zentrum berechnet wird. Es werden mehrere Durchgänge des K-Means-Algorithmus durchlaufen, die auf der x-Achse mit der Anzahl der Cluster dargestellt werden. Da diese Kurve oftmals einem Ellenbogen ähnelt und ein guter Schätzwert der Clusteranzahl sich bei dem Knickpunkt des Ellenbogen befindet, wird die Methode als „Elbow-Methode“ bezeichnet. Es lässt sich also zeigen, dass es durchaus Sinn macht mit der Clusteranzahl von

fünf zu starten.

Als nächstes muss überlegt werden, ob eine Normalisierung/Standardisierung der Daten notwendig ist. Dies hat großen Einfluss auf die Möglichkeit der Gruppierung, da bei einer Standardisierung Abstände zwischen Gruppen gestreckt oder gestaucht werden. Vorerst wird keine Standardisierung vorgenommen, da sich die Daten bereits zwischen 0 und 1 befinden und einer bestimmten Verteilung gehorchen. Im weiteren Verlauf werden dennoch verschiedene Standardisierungsmethoden getestet, wie mit einem Beispiel in Listing 8 demonstriert wird.

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler() # Scale Data e.g. 0 to 1
X = scaler.fit_transform(df)
```

Listing 8: Standardisierung mit Sklearn MinMaxScaler

Unser Dataframe „df“ wird zu der Matrix X transformiert und daraufhin mit einer PCA Analyse untersucht, bei der sich die ersten 150 Hauptkomponenten angeschaut werden.

```
mPCA = PCA(n_components=150)
PrincipleComponents = mPCA.fit_transform(X)
variance = mPCA.explained_variance_ratio_
variance_ratio = np.cumsum(np.round(variance, decimals=3)*100)
```

Listing 9: PCA mit 150 berechneten Hauptkomponenten

Das Ergebnis der Varianz von Hauptkomponente zu Hauptkomponente wird in der Matrix „variance_ratio“ dargestellt. Die erste Hauptkomponente repräsentiert zu 15% die Varianz des gesamten Datensatzes. Selbst mit 100 Hauptkomponenten wird insgesamt nur eine 94.4 prozentige Varianz des ursprünglichen Datensatzes erreicht. Vorerst ist noch abzuklären, ob eine niedrige Varianz, in zwei oder drei Dimensionen, die Phasenerkennung beeinflusst.

$$variance\_ratio = \Big[15.0, 27.5, 33.3, 38.1, 42.7, ..., 96.9, 96.9, 96.9, 96.9\Big] \tag{14}$$

Die Varianz der Hauptkomponenten kann ebenfalls nochmals graphisch veranschaulicht werden. In Abbildung 3.7 ist hierfür die prozentuale Repräsentation gegenüber der Anzahl der Hauptkomponenten aufgetragen.

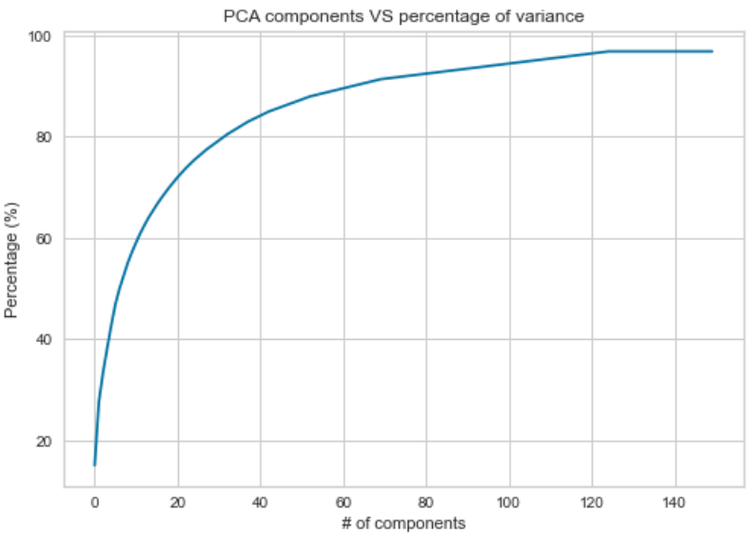


Abb. 3.7: Varianz der 150 Hauptkomponenten

In der Abbildung oberhalb ist ersichtlich, dass nach 95% zusätzliche Komponenten keinen weiteren Gewinn an Informationen bringen. Deshalb wird sich für diese Grenze entschieden, mit 100 Hauptkomponenten eine Gruppierung vorzunehmen.

```
pca = PCA(n_components=100) # 100 components with 95%
PCA = pca.fit_transform(X)
```

Listing 10: PCA mit 100 berechneten Hauptkomponenten

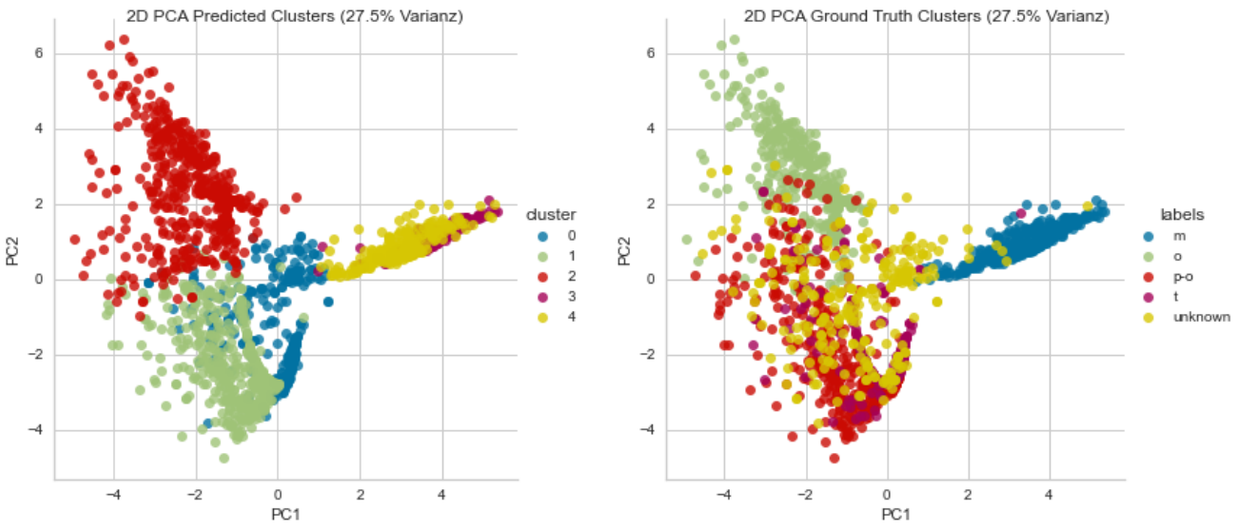
Die neue Matrix „PCA“ besitzt 1725 Reihen auf 100 Spalten und kann mit dem K-Means-Algorithmus geclustert werden. Zur Beurteilung der Cluster wird mit dem Silhouettenkoeffizient gearbeitet, der einen Wert zwischen 0 und 1 für die Stärke der Strukturierung der Objekte berechnet [25].

```
from sklearn import metrics
kmeans = KMeans(n_clusters=5)
PCA_labels = kmeans.fit_predict(PCA)
metrics.silhouette_score(PCA, PCA_labels, metric='l2') # l2 norm = euclidean norm
```

Listing 11: K-Means mit Silhouettenkoeffizienten

Es ergibt sich ein Silhouettenkoeffizient von 0.16, der einer sehr schwach vorhandenen Clusterbildung entspricht. Das Ergebnis der Clustergruppen wird in der Variable «PCA_labels» gespeichert und entspricht einer zufälligen Nummerierung der Gruppen von 0 bis 4. Um diese mit den Ground-Truth-Labels (m, p-o, o, t, unbekannt) zu vergleichen, müssen diese ebenfalls in die dazugehörigen Integer-

werte umgewandelt werden. Mit der gesamten Klassifizierung der vorhandenen Labels berechnet sich ein Accuracy-Score von 54.2%. Wenn die zwei ersten Hauptkomponenten nun graphisch gegeneinander aufgetragen werden, ergibt sich eine Gruppierung wie in Abbildung 3.8.



(a) (b)

Abb. 3.8: PCA mit K-Means Cluster in 2D, XRD mit hoher Auflösung, (a) K-means, (b) Ground-Truth-Labels

Bei einem Vergleich mit den Ground-Truth-Labels (Abbildung 3.8) wird aber klar, dass durch die geringe Varianz und dem jetzigen Verfahren keine klare Trennung der Gruppen möglich ist.

Für eine 3D Visualisierung der PCA siehe Anhang A, A.5,A.4 oder unter „Cluster with high resolution XRD.ipynb “.

Bevor weitere Cluster-Algorithmen angewandt werden, sind zusätzliche Dimensionsreduzierungsmethoden zu testen, die eine bessere Repräsentation im 2D/3D-Raum darstellen sollen. Die Reduzierung von Röntgendiffraktogrammen mittels der NMF (Abschnitt 2.1.1.3), wie sie in [26] vorgeschlagen wurde, wird nun ausgiebig getestet.

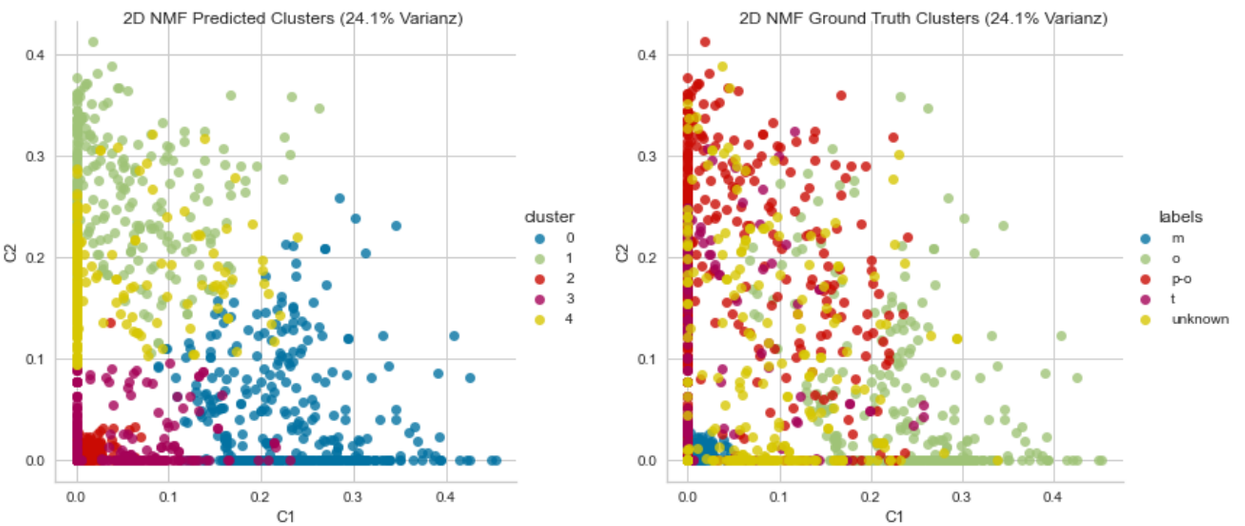
```
from sklearn.decomposition import NMF

model = NMF(n_components=4, init='random', random_state=0, max_iter = 2000)
nmf_features_W = model.fit_transform(X) # Split in W and H Matrix
```

```
5 nmf_componentes_H = model.components_
```

Listing 12: NMF-Dekomposition der Dimensionen des XRD

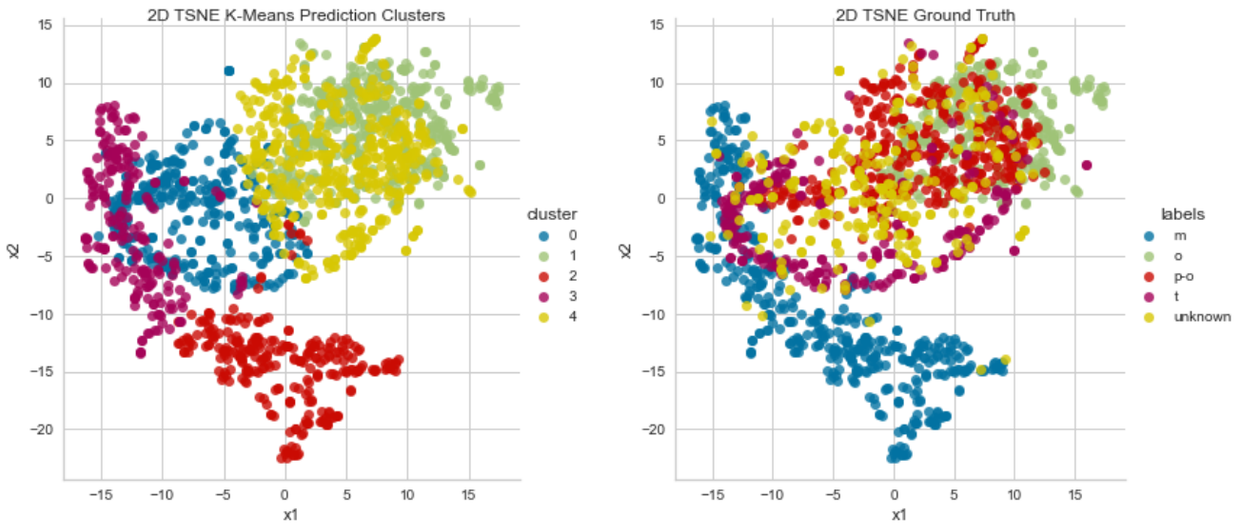
Dieses Vorgehen eignet sich besonders um mit XRD-Daten umzugehen, da bei der Reduzierung physikalische interpretierbare Ergebnisse entstehen [26]. Es werden nur die Gewichte der Matrix W in den K-Means-Algorithmus gegeben und nochmals der Silhouettenkoeffizient sowie Accuracy-Score berechnet. Ein Varianzwert von 31% im dreidimensionalen Raum besitzt zu vorherigem Ergebnis keine starke Veränderung. Eine deutliche Verbesserung zeigt hingegen der Silhouettenkoeffizient von 0.5 auf.



(a) (b)

Abb. 3.9: NMF K-Means-Cluster in 2D, XRD mit hoher Auflösung, (a) K-means, (b) Ground Truth Labels

Zusätzlich wird noch der TSNE-Algorithmus als Reduktions- und Visualisierungsmethode getestet. Wie in Abbildung 3.10 zu sehen ist, können dichtere und besser trennbare Cluster gefunden werden. Das Problem der falschen Zuordnung wird durch die bereits besprochenen Probleme des K-Means-Algorithmus hervorgerufen. Der Zwang nach gleich großen und ähnlich geformten Gruppen, wird im späteren Verlauf mit weiteren Algorithmen versucht zu beheben.



(a) (b)

Abb. 3.10: TSNE K-Means-Cluster in 2D, XRD mit hoher Auflösung, (a) K-means, (b) Ground Truth Labels

3.4.2 Testreihe mit Dimensionsreduzierung und Cluster-Algorithmen

Im vorherigen Abschnitt wurde der schematische Ablauf des Testens und Validierens aufgezeigt. Nun werden verschiedene Kombinationen und Algorithmen ausprobiert, bei denen auf eine ausführliche Schritt für Schritt Anleitung verzichtet wird. Im Gegensatz dazu werden nur die Ergebnisse präsentiert und im späteren Verlauf diskutiert. Zum Test stehen Neutronenstreuung, XRD-Fein, XRD-Grob, XRD mit ausgetauschten Sauerstoffatomen, XRD mit Atomzellwinkeln 80/90° [19] und PRDF bereit.

Deskriptor	Silhouettenkoeffizient	Accuracy
XRD grob (0-180), PCA	0.44	X
XRD grob (0-180), NMF	0.57	0.58
XRD grob (0-180), TSNE	0.44	0.78
XRD grob (5-65), PCA	0.59	0.76
XRD grob (5-65), NMF	0.61	X
XRD grob (5-65), TSNE	0.43	X

Tabelle 3.9: Versuchsreihe mit K-Means, verschiedene Deskriptoren, Dimensionsreduzierungsmethoden (X = nicht gruppierbar)

Nicht in allen Fällen ist eine richtige Gruppierung durch den K-Means-Algorithmus möglich, da eine große Anzahl an Falschzuweisungen geschieht. Allgemein lässt sich aber sagen, dass der Silhouettenko-

[19] Die simulierte Zelle hat leicht abweichende Werte von 80/90°. Diese werden auf den Mittelwert der Winkel zurückgesetzt.

effizient bei dem XRD, mit niedrigerer Auflösung (1° Schrittweite), höhere Werte besitzt. Eine erneute Skalierung der XRD-Daten, zwischen null und eins, hat in einigen Fällen eine bessere Gruppierung ermöglicht.

Deskriptor	Silhouettenkoeffizient	Accuracy
Neutron (0-180), PCA	0.39	X
Neutron (0-180), NMF	0.43	X
Neutron (0-180), TSNE	0.38	0.84
XRD grob, ohne O(5-65), PCA	0.51	0.90
XRD grob, ohne O (5-65), NMF	0.77	0.87
XRD grob, ohne O (5-65), TSNE	0.50	**0.91**
XRD grob (5-65), Winkel 80/90° PCA	0.55	X
XRD grob (5-65), Winkel 80/90° NMF	0.65	X
XRD grob (5-65), Winkel 80/90° TSNE	0.51	0.75
PRDF, PCA	0.56	**0.91**
PRDF, NMF	0.68	**0.91**
PRDF, TSNE	0.51	0.88

Tabelle 3.10: 2. Versuchsreihe mit K-Means, weitere Deskriptoren in unterschiedlichen Varianten und DR-Methoden (X = nicht gruppierbar)

Die Tabelle 3.10 macht deutlich, dass der Silhouettenkoeffizient im Bereich 0.4-0.7 keine genauen Aussagen über die Accuracy der Cluster treffen kann. Er muss stattdessen als grobe Tendenz verwendet werden, die nicht immer mit der richtigen Gruppierung der gewünschten Cluster übereinstimmt. Vielmehr spielt die visuelle Betrachtung der Gruppen nach der Dimensionsreduzierung eine wichtige Rolle, bei dem in einigen Fällen ein Label X als „nicht gruppierbar“ verteilt wird. Durch einige Deskriptoren sind die Gruppen zu eng nebeneinander oder durcheinander verteilt und können deshalb auch mit keiner alternativen Clustermethode getrennt werden. Trotzdem wird sofort deutlich, dass die XRD Muster ohne Sauerstoff und PRDF unabhängig von der Dimensionsreduzierung deutlich bessere Ergebnisse liefern. Diese sollen mit weiteren Testdurchläufen geprüft werden, indem Hyperparameter und andere Cluster-Algorithmen zur Verwendung kommen. Außerdem werden für die besten Deskriptoren ebenfalls Berechnungen zu dem ZrO_2 Datensatz angestellt, um diese mit der Genauigkeit der HfO_2-Gruppierung zu vergleichen.

Deskriptor	S(o) (HfO_2/ZrO_2)	Accuracy (HfO_2/ZrO_2)
XRD grob, ohne O(5-65), PCA, PC=15 (Varianz 95%, SS)	0.59/0.63	**0.93**/0.85
XRD grob, ohne O (5-65), NMF, C=8/38, SS	0.59/0.36	**0.93**/0.88
XRD grob, ohne O (5-65), TSNE, Perplexity = 48/30, SS	0.56/0.46	0.92/0.92
PRDF, (0-20, 0.5 Schritte) PCA, PC=18/19, (Varianz 95%, SS)	0.42/0.38	0.92/0.85
PRDF, (0-20, 0.5 Schritte) NMF, C = 21/33, SS	0.38/0.30	0.92/**0.93**
PRDF, (0-20, 0.5 Schritte) TSNE, Perplexity = 62/65, SS	0.56/0.53	0.91/0.91

Tabelle 3.11: 3. Versuchsreihe K-Means mit erweiterten Hyperparametern (SS = StandardScaler, S(o) = Silhouettenkoeffizient)

Für den ersten Deskriptor (XRD grob, ohne O (5-65)) wird die Hauptkomponentenanzahl mit verschieden Werten getestet, wobei PC = 15 das beste Ergebnis (Acc = 0.93, Varianz = 95%) liefert. Ebenfalls wurden für den ersten Deskriptor verschiedene Streuwinkel abgefahren welche, zeigen, dass ein Bereich von 5° bis 65° tatsächlich die bestmögliche Variante darstellt, ohne Verluste in der Genauigkeit der Cluster einbüßen zu müssen [20]. Es ergibt sich ebenfalls, dass mit einer weiteren Standardskalierung (SS), von null bis eins, die Gruppen besser geclustert werden können. Mit der Hyperparameteroptimierung von Komponenten und Perplexität können verbesserte Werte in der Genauigkeit gefunden werden. Um eine Überanpassung der Hyperparameter zu vermeiden, wird der gleiche Vorgang ebenfalls für ZrO_2 wiederholt. Dabei ergeben sich leicht geringere Werte in der Genauigkeit, mit jeweils anderen Hyperparametern. Auch die nicht deterministische Vorgehensweise der DR-Methoden ist zu beachten, weshalb für die gleichen Hyperparameter teilweise unterschiedliche lokale Minimas gefunden werden.

Unter der Verwendung der TSNE-Methode kann mit beiden Deskriptoren eine gute visuelle Trennung der Gruppen erreicht werden. Dies erleichterten die Validierung und Zuweisung der Phasen, um eine verallgemeinerte und reproduzierbare Methode zu erhalten. Der Hyperparameter der Perplexität befindet sich in einem optimalen Bereich zwischen 30 und 70. Die Matrixfaktorisierung ist bereits mit verschiedenen Komponenten zwischen 5 und 40 getestet worden. Hierbei muss bei hoher Komponentenanzahl der stark ansteigende Rechenbedarf beachtet werden. Ebenfalls ist die visuelle Trennung der Gruppen nicht allzu stark ausgeprägt, die die Genauigkeit der Ergebnisse aber dennoch nicht verschlechtern.

[20] Getestet wurden Kombinationen von (5-90), (0-120), (5-45), (20-60) und (0-200), die entweder gleich oder schlechter im Wert der Genauigkeit waren.

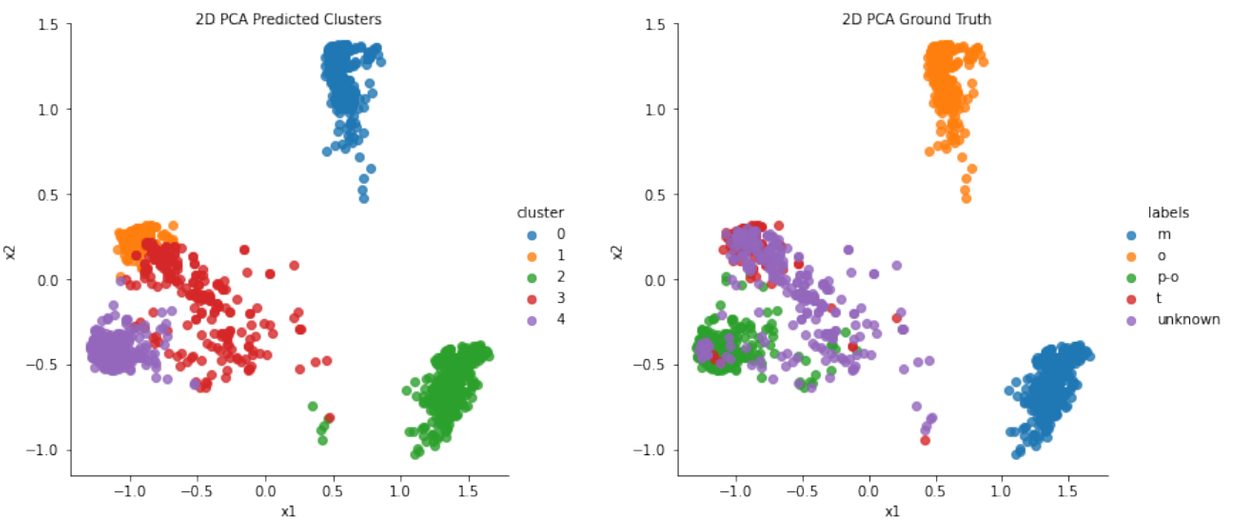
Zum Schluss werden weitere Algorithmen, wie DBSCAN und hierarchisches Clustern getestet und validiert. Dies macht nur in begrenzten Fällen Sinn, da einige Ergebnisse bereits eine geeignete Gruppierung durch die Dimensionsreduzierung aufweisen. Hierarchisches Clustern zeigt mit verschiedenen Linkage-Verfahren keine deutliche Verbesserung auf. Das Ward-Linkage, das verwandt mit dem K-Means-Algorithmus ist, zeigt ähnlich gute Ergebnisse auf [21]. Der DBSCAN/OPTICS-Algorithmus wird getestet, da sich einige Gruppen nicht in der passenden Form für den K-Means-Algorithmus befinden. Hierfür wird eine Anzahl an verschiedenen Hyperparametern „minPts“ getestet und mit der jeweiligen höchsten Genauigkeit ausgegeben. Da die Daten nach der Dimensionsreduzierung aber oftmals eng zusammen liegen, ist auch hier eine Unterscheidung der Gruppen im Vergleich zum Rauschen schwer möglich.

[21] Beispiel: AgloCluster_DR_XRD.ipynb im Github Repo

IV. ERGEBNISSE

Die zuvor gewonnenen Ergebnisse aus Tabelle 3.11 zeigen, dass die beiden Deskriptoren gut geeignet sind, um die Phasen in ihre verschiedenen Gruppen einzuteilen. Der Genauigkeitswert lässt sich mit den bereits bekannten Labels über den Datensatz berechnen. Nun soll eine visuelle Interpretation der Ergebnisse folgen, um mögliche falsche Label zu erkennen und die Gruppierung sinnvoll einzuordnen. Dazu wird eine Automatisierung in Python vorgenommen, die es ermöglicht Hyperparameter, Dimensionsreduzierung und Algorithmen schnell und einfach auszuprobieren. Die dafür benötigten Funktionen werden in zwei Pythonskripte [22] geschrieben und im jeweiligen Jupyternotebook geladen.

Zuerst werden die Ergebnisse des XRD mit grober Struktur und PCA/K-Means betrachtet und bewertet. Die PCA wurde mit 15 Hauptkomponenten berechnet, bei dem der K-Means-Algorithmus folgendes Ergebnis erreicht:



(a) (b)

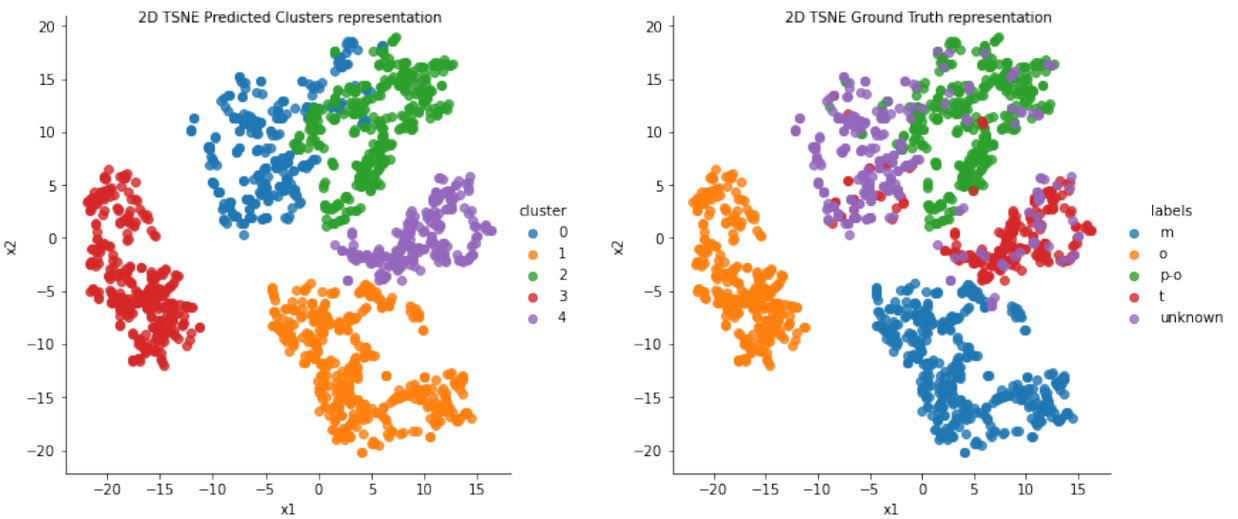
Abb. 4.1: PCA K-Means Cluster in 2D, HfO_2 XRD mit niedriger Auflösung, (a) K-Means, (b) Ground-Truth-Labels

Die Phasen der monoklinen und orthorhombischen Kristalle sind sehr gut separiert und vom Rest der Phasen abgespalten. Auch die Gruppen der tetragonalen und polar-orth. Phasen bilden eindeutige, einzelne Gruppen. Die Vermutung einer eventuellen unbekannten Phase bestätigt sich hierbei, da diese

[22] Siehe Anhang B Listing 17 und 18

in der 2D Ebene der Hauptkomponenten eine weitere Gruppe bilden[23]. Diese ist dennoch nicht eindeutig abzuspalten und bildet somit eine Art Mischphase zwischen polar-orth. und tetragonaler Phase. Auffällig ist auch, dass einige Phasen, die als unbekannt gelabelten Phase vorhanden sind, eindeutig im Cluster der tetragonalen oder polar-orth. Phase liegen. Dies ist genauer zu überprüfen und könnte ein Hinweis auf ein falsch zugeordnetes Label sein.

Da die PCA zusammen mit K-Means im Vergleich auf beide Datensätze HfO_2 und ZrO_2, aber hohe Schwankungen aufweist, wird die in der Genauigkeit konstantere Methode TSNE mit K-Means ebenfalls visuell betrachtet. Für den DR-Algorithmus wird eine Perplexität zwischen 40-60 ausgewählt.



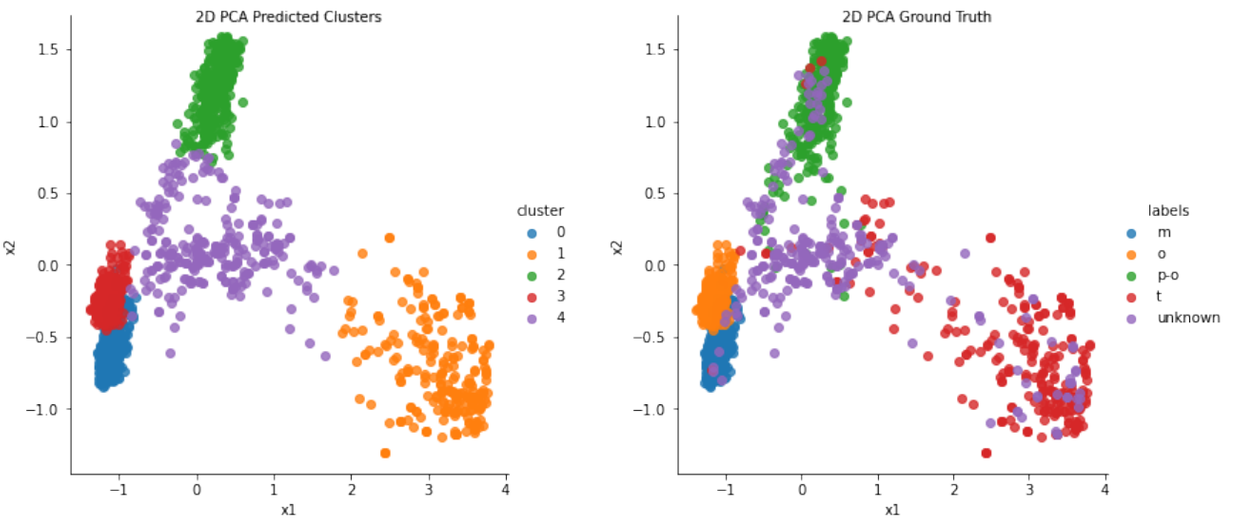
(a) (b)

Abb. 4.2: PCA K-Means-Cluster in 2D, HfO_2 XRD mit niedriger Auflösung, (a) K-Means, (b) Ground-Truth-Labels

Die Ergebnisse sehen analog zu der PCA vielversprechend aus, wobei die tetragonale, polar-orth. und unbekannte Phase ebenfalls teilweise überlappen. Da der Datensatz von Grund auf in ungefähr gleich große Phasengruppen aufgeteilt ist, eignet sich der K-Means-Algorithmus hierfür besonders gut.

[23]Eine bessere Visualisierung in 3D ist im GithubRepo zu finden.

Die PRDF in Schritten von 0 bis 20, zwischen Hafnium und Sauerstoff, wird nun ebenfalls mit der PCA reduziert. Dabei wird eine Hauptkomponentenanzahl von 18 ausgewählt.



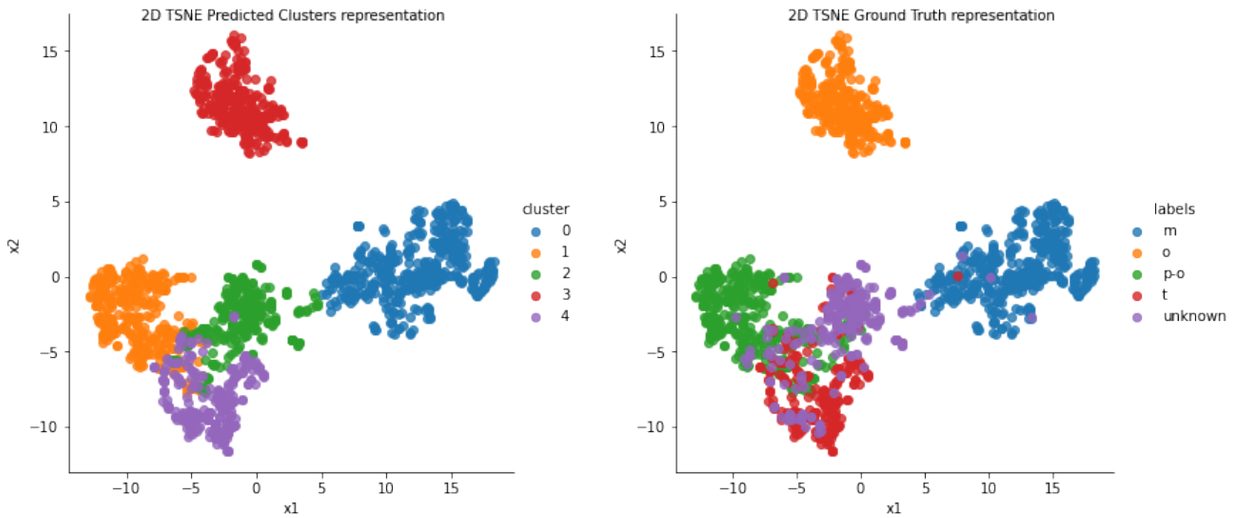
(a) (b)

Abb. 4.3: PCA K-Means-Cluster in 2D, HfO_2 PRDF mit (0-20, 0.5 Schritte) Auflösung, (a) K-Means, (b) Ground-Truth-Labels

Die PRDF schafft es die Phasen gut voneinander zu trennen, obwohl die Sauerstoffdaten zuvor nicht manipuliert worden sind. Daher kann mit der PRDF ein verallgemeinertes Verfahren gewonnen werden, welches mit theoretisch simulierten Daten arbeiten kann, ohne die Atomparameter zu verändern. Dies war im Falle der XRD-Simulationen nicht möglich, weshalb die Sauerstoffatome gegen ein höherwertiges Element ausgetauscht wurden.

Die schwankende Varianz der ersten zwei oder drei Dimensionen der Reduzierung von bis zu 40% hat keinen großen Einfluss auf die Erkennung der Phasen. Das zeigt wiederum, dass nur einige wenige Feature ausreichen, um die markanten Stellen zur Phasenerkennung ausfindig zu machen.

Die Ähnlichkeit der polar-orthorhombischen und tetragonalen Phase, die im XRD sowie in der PRDF zu sehen sind, lassen sich mit dem jetzigen Deskriptor nicht weiter unterscheiden. Die visuelle Begutachtung des XRD Musters, zwischen der polar-orth.- und tetragonalen Phase, sind auch im experimentellen Verfahren durch das Auge nur schwer zu unterscheiden. Eine Verbesserung konnte die Neutronenstreuung aufweisen, die aber aufgrund der hohen Auflösung mit den DR-Methoden nicht ausreichend reduziert werden kann.



(a) (b)

Abb. 4.4: TSNE K-Means-Cluster in 2D, HfO_2 PRDF mit (0-20, 0.5 Schritte) Auflösung, (a) K-Means, (b) Ground-Truth-Labels

Um zu sehen, wie sich die Phasendetektion bei einer Mischung der Verbindung verhält, wurden zusätzlich der HfO_2 und ZrO_2 Datensatz zusammen betrachtet. Da die Elemente aber in ihrer Ordnungszahl abweichen, verschieben sich die Phasen m, o, p-o und t der jeweiligen Verbindung und es bilden sich acht anstelle von vier Gruppen im transformierten Raum. Wenn die Verbindungen zusammen betrachtet werden sollen, muss zuvor Hafnium mit Zirkon ausgetauscht werden, um eine Normierung zu bewirken. Alle weiteren Berechnungen zu ZrO_2 und zusätzlichen Algorithmen sind jeweils in dem entsprechenden Github Repository zu finden.

Die Problematik der Ähnlichkeit zwischen der tetragonalen und polar-orth. Phase kann mit den vorhandenen Deskriptoren nicht weiter verfeinert werden. Dieses Problem bleibt für nachfolgende Arbeiten bestehen, wobei die Neutronenstreuung in geringerer Auflösung vielversprechende Ergebnisse liefern könnte. Sobald zusätzliche Datensätze vorhanden sind, sollten weitere Tests mit Verbindungen und Dotanden angestellt werden, um wichtige Aussagen für eine Generalisierung der Ergebnisse treffen zu können.

V. ZUSAMMENFASSUNG UND AUSBLICK

Ziel dieser Arbeit war es theoretisch simulierte Kristalle der Verbindungen Hafnium- und Zirkonoxid in seine Phasen zu unterteilen. Dafür wurden die Verbindungen in vier Kristallphasen (monoklin, orthorhombisch, polar-orthorhombisch, tetragonal) auf dem „SuperMuc“ berechnet und in einem Ab-Initio-Datensatz hinterlegt. Ein großer Teil der Arbeit war die Aufbereitung der Daten, um daraufhin einen passenden Deskriptor zu finden, der die Phase hinzureichend beschreibt. Durch das Einbringen von Dotierstoffen in die Kristallverbindungen bilden sich teilweise starke und weniger starke Verzerrungen in den Kristallen. Diese beeinflussen die Kristallphase und können zu einer Transformation der Phase führen. Die Problematik der Einsortierung von eventuell auftretenden unbekannten Phasen, benötigt stets die manuelle Begutachtung jedes einzelnen Kristalls, was zu einem enormen Zeitaufwand führt. Dieses Problem konnte durch die Verwendung eines Cluster-Algorithmus vereinfacht werden, der die bereits bekannten Phasen effizient und deutlich voneinander trennt.

Durch die funktionale Datenanalyse der XRD-Muster konnten die wichtigen Merkmale der Phasen in einem Intensitätsplot betrachtet werden. Hierbei wurde eine Genauigkeit von 80-85% erreicht, wobei einige Phasen offensichtlich falsch zugeordnet wurden. Da diese Methode zusätzlich in der Flexibilität der einsetzbaren Algorithmen eingeschränkt ist, wurde der Übergang zu den traditionellen Methoden des maschinellen Lernens in scikit-learn gemacht. Mit dem Deskriptor des XRD-Musters in grober Auflösung (0-65) wird eine durchschnittliche Genauigkeit von 92% erreicht, die anhand von manuell erstellten Labels validiert wurde. Das Auffinden von fünf anstatt vier trennbarer Gruppen ist unerwartet und bedarf zusätzlichen weiteren Untersuchungen. Die dabei auftretenden physikalischen Besonderheiten sollen in anschließenden Arbeiten identifiziert werden. Auch der alternative Deskriptor der PRDF zeigt zufriedenstellende Ergebnisse, wenn für die Berechnung Hafnium/Zirkon- und Sauerstoffelemente ausgewählt sind. Die Dimensionsreduzierungsmethode TSNE kann in nahezu allen Testdurchläufen dichte und trennbare Gruppen ausbilden, die visuell in zwei und drei Dimensionen veranschaulicht werden.

Mit diesem Verfahren kann die manuelle Erstellung von Labels zur Phasenveränderung deutlich verbessert werden. Zudem gelingt es Ausreißer aus Gruppen durch die visuelle Betrachtung schnell zu erkennen und damit weitere Untersuchungen der betroffenen Kristalle vorzunehmen. Die explorative Natur

des unüberwachten Lernens bietet für diese Anwendung eine gute Mischung zwischen Automation der Phasendetektion und dem traditionellen Vorgehen einer XRD-Analyse.

VI. DATENVERFÜGBARKEIT

Die simulierten Datensätze zu HfO_2 und ZrO_2 mit Dotanten und allen weiteren verarbeiteten Datensätze stehen online unter `https://github.com/MaxiTrien/ml-crystal-phase-prediction/tree/master/data` zur Verfügung.

LITERATUR

[1] G. James, D. Witten, T. Hastie und R. Tibshirani, "An Introduction to Statistical Learning: With Applications in R", in. Springer Publishing Company, Incorporated, 2014, S. 373–385, 386–401, ISBN: 1461471370.

[2] P. Mehta, M. Bukov, C.-H. Wang, A. G. Day, C. Richardson, C. K. Fisher und D. J. Schwab, "A high-bias-variance introduction to Machine Learning for physicists", *Physics Reports*, Jg. 810, S. 67, 72, Mai 2019, ISSN: 0370-1573. DOI: 10.1016/j.physrep.2019.03.001. Adresse: http://dx.doi.org/10.1016/j.physrep.2019.03.001.

[3] C. C. Aggarwal und C. K. Reddy, "Data Clustering: Algorithms and Applications", in, 1st. Chapman und amp; Hall/CRC, 2013, S. 373, 113–115, ISBN: 1466558210.

[4] Matthias Scholz, Max Planck Institut, *PCA transformation*, [Online; accessed Oktober 15, 2020]. Adresse: http://phdthesis-bioinformatics-maxplanckinstitute-molecularplantphys.matthias-scholz.de/fig_pca_illu3d.png.

[5] P. F. Brown, V. J. D. Pietra, R. L. Mercer, S. A. D. Pietra und J. C. Lai, "An Estimate of an Upper Bound for the Entropy of English", *Comput. Linguist.*, Jg. 18, Nr. 1, S. 31–40, März 1992, ISSN: 0891-2017.

[6] C. E. Shannon, "A mathematical theory of communication.", *Bell Syst. Tech. J.*, Jg. 27, Nr. 3, S. 379–423, 1948. Adresse: http://dblp.uni-trier.de/db/journals/bstj/bstj27.html#Shannon48.

[7] L. Van Der Maaten, "Accelerating T-SNE Using Tree-Based Algorithms", *J. Mach. Learn. Res.*, Jg. 15, Nr. 1, S. 3221–3245, Jan. 2014, ISSN: 1532-4435.

[8] N. Gillis, "The Why and How of Nonnegative Matrix Factorization", *ArXiv*, Jg. abs/1401.5226, 2014.

[9] D. Lee und H. Seung, "Learning the Parts of Objects by Non-Negative Matrix Factorization", *Nature*, Jg. 401, S. 788–91, Nov. 1999. DOI: 10.1038/44565.

[10] X. Lu, Q. Jiang und B. Meng, "A Comparison of Three Different Matrix Factorization Techniques, PCA, ICA, and NMF, for Unsupervised Machine Learning", 2013. Adresse: https://cnx.org/exports/f84946a2-17ea-495f-95ba-6f2c336d8239@1.4.pdf/comparison-of-three-different-matrix-factorization-techniques-for-unsupervised-machine-learning-1.4.pdf.

[11] V. K. Vemuri, "The Hundred-Page Machine Learning Book", *Journal of Information Technology Case and Application Research*, Jg. 22, Nr. 2, S. 136–138, 2020. DOI: 10.1080/15228053.2020.1766224. eprint: https://doi.org/10.1080/15228053.2020.1766224. Adresse: https://doi.org/10.1080/15228053.2020.1766224.

[12] E. Schubert, J. Sander, M. Ester, H. P. Kriegel und X. Xu, "DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN", *ACM Trans. Database Syst.*, Jg. 42, Nr. 3, Juli 2017, ISSN: 0362-5915. DOI: 10.1145/3068335. Adresse: https://doi.org/10.1145/3068335.

[13] A. Seko, A. Togo und I. Tanaka, "Descriptors for Machine Learning of Materials Data", in *Nanoinformatics*, I. Tanaka, Hrsg. Singapore: Springer Singapore, 2018, S. 3–23, ISBN: 978-981-10-7617-6. DOI: 10.1007/978-981-10-7617-6_1. Adresse: https://doi.org/10.1007/978-981-10-7617-6_1.

[14] L. Himanen, M. O. Jäger, E. V. Morooka, F. Federici Canova, Y. S. Ranawat, D. Z. Gao, P. Rinke und A. S. Foster, "DScribe: Library of descriptors for machine learning in materials science", *Computer Physics Communications*, Jg. 247, S. 106 949, 2020, ISSN: 0010-4655. DOI: https://doi.org/10.1016/j.cpc.2019.106949. Adresse: http://www.sciencedirect.com/science/article/pii/S0010465519303042.

[15] K. T. Schütt, H. Glawe, F. Brockherde, A. Sanna, K. R. Müller und E. K. U. Gross, "How to represent crystal structures for machine learning: Towards fast prediction of electronic properties", *Physical Review B*, Jg. 89, Nr. 20, Mai 2014, ISSN: 1550-235X. DOI: 10.1103/physrevb.89.205118. Adresse: http://dx.doi.org/10.1103/PhysRevB.89.205118.

[16] Scott A Speakman, *Introduction to X-Ray Powder Diffraction Data Analysis*, URL: http://prism.mit.edu/xray/introductiontoxrpddataanalysis.pdf, Massachusetts Institute of Technology.

[17] Elena Willinger, *Analysis of Local Structure by Atomic Pair Distribution Function*, URL: http://www.fhi-berlin.mpg.de/acnew/department/pages/teaching/pages/teaching__wintersemester__2016_2017/elena_willinger__analysis_of_local_structure_by_atomic_pair_distribution_function__170203.pdf, Fritz-Haber-Institut der Max-Planck-Gesellschaft.

[18] L. Jin-woong, W. Park, J. Lee, S. Singh und K.-S. Sohn, "A deep-learning technique for phase identification in multiphase inorganic compounds using synthetic XRD powder patterns", *Nature Communications*, Jg. 11, S. 86, Dez. 2020. DOI: 10.1038/s41467-019-13749-3.

[19] M. Dasari, G. Gajula, R. Hanumantha, A. Chintabathini, S. Kurimella und S. Bharadwaj, *XRD Pattern*, [Online; accessed Oktober 27, 2020], 2017. Adresse: https://www.researchgate.net/publication/315987813_Lithium_ferrite_The_study_on_magnetic_and_complex_permittivity_characteristics.

[20] M. Falkowski, C. Künneth, R. Materlik und A. Kersch, "Unexpectedly large energy variations from dopant interactions in ferroelectric HfO2 from high-throughput ab initio calculations", *npj Computational Materials*, Jg. 4, Dez. 2018. DOI: 10.1038/s41524-018-0133-4.

[21] W. McKinney, "Data Structures for Statistical Computing in Python", in *Proceedings of the 9th Python in Science Conference*, S. van der Walt und J. Millman, Hrsg., 2010, S. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

[22] T. pandas development team, *pandas-dev/pandas: Pandas*, Version latest, Feb. 2020. DOI: 10.5281/zenodo.3509134. Adresse: https://doi.org/10.5281/zenodo.3509134.

[23] A. H. Larsen, J. J. Mortensen, J. Blomqvist, I. E. Castelli, R. Christensen, M. Duak, J. Friis, M. N. Groves, B. Hammer, C. Hargus, E. D. Hermes, P. C. Jennings, P. B. Jensen, J. Kermode, J. R. Kitchin, E. L. Kolsbjerg, J. Kubal, K. Kaasbjerg, S. Lysgaard, J. B. Maronsson, T. Maxson, T. Olsen, L. Pastewka, A. Peterson, C. Rostgaard, J. Schiøtz, O. Schütt, M. Strange, K. S. Thygesen, T. Vegge, L. Vilhelmsen, M. Walter, Z. Zeng und K. W. Jacobsen, "The atomic simulation environmenta Python library for working with atoms", *Journal of Physics: Condensed*

Matter, Jg. 29, Nr. 27, S. 273 002, 2017. Adresse: http://stacks.iop.org/0953-8984/29/i=27/a=273002.

[24] L. Ward, A. Dunn, A. Faghaninia, Zimmermann, S. N. E. R. Bajaj, Q. Wang, M. J. H., J. Chen, B. K., D. M., C. K., A. M., P. K., S. G., J. F. I. und J. A., "Matminer: An open source toolkit for materials data mining", S. 60–69, 2018.

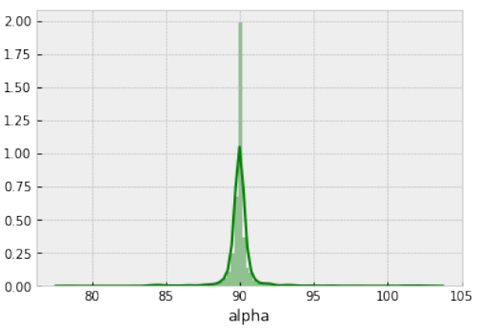
[25] P. Rousseeuw, "Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis", *J. Comput. Appl. Math.*, Jg. 20, Nr. 1, S. 53–65, Nov. 1987, ISSN: 0377-0427. DOI: 10.1016/0377-0427(87)90125-7. Adresse: https://doi.org/10.1016/0377-0427(87)90125-7.

[26] V. Stanev, V. Vesselinov, A. Kusne, G. Antoszewski, I. Takeuchi und B. Alexandrov, "Unsupervised Phase Mapping of X-ray Diffraction Data by Nonnegative Matrix Factorization Integrated with Custom Clustering", *npj Computational Materials*, Jg. 4, S. 1–2, Feb. 2018. DOI: 10.1038/s41524-018-0099-2.

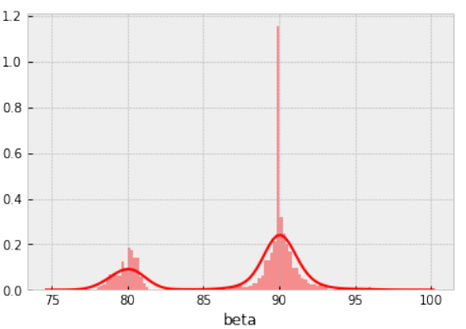
A. Bilder und Tabellen

Ag	Al	As	Au	B	Ba	Be	Bi	C	Ca	Cd	Ce	Co	Cr	Cs
Cu	Fe	Ga	Ge	H	Hf	Hg	In	Ir	K	La	Li	Mg	Mn	Mo
N	Na	Nb	Ni	Os	P	Pb	Pd	Pt	Rb	Re	Rh	Ru	Sb	Sc
Si	Sn	Sr	Ta	Tc	Ti	Tl	V	W	Y	Yb	Zn	Zr		

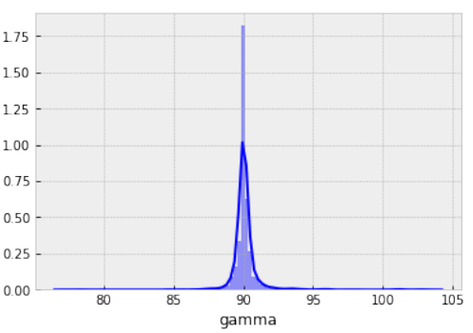
Tabelle A.1: Liste der ausgewählten Dotanten.



(a)

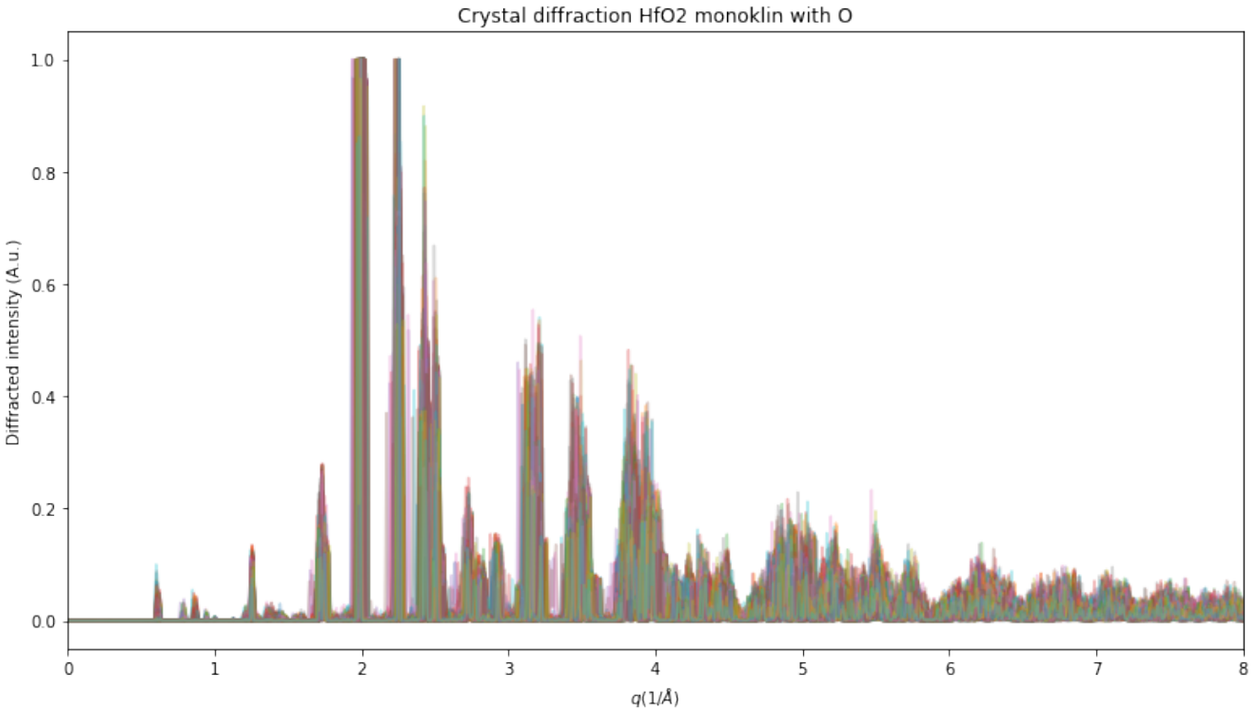


(b)

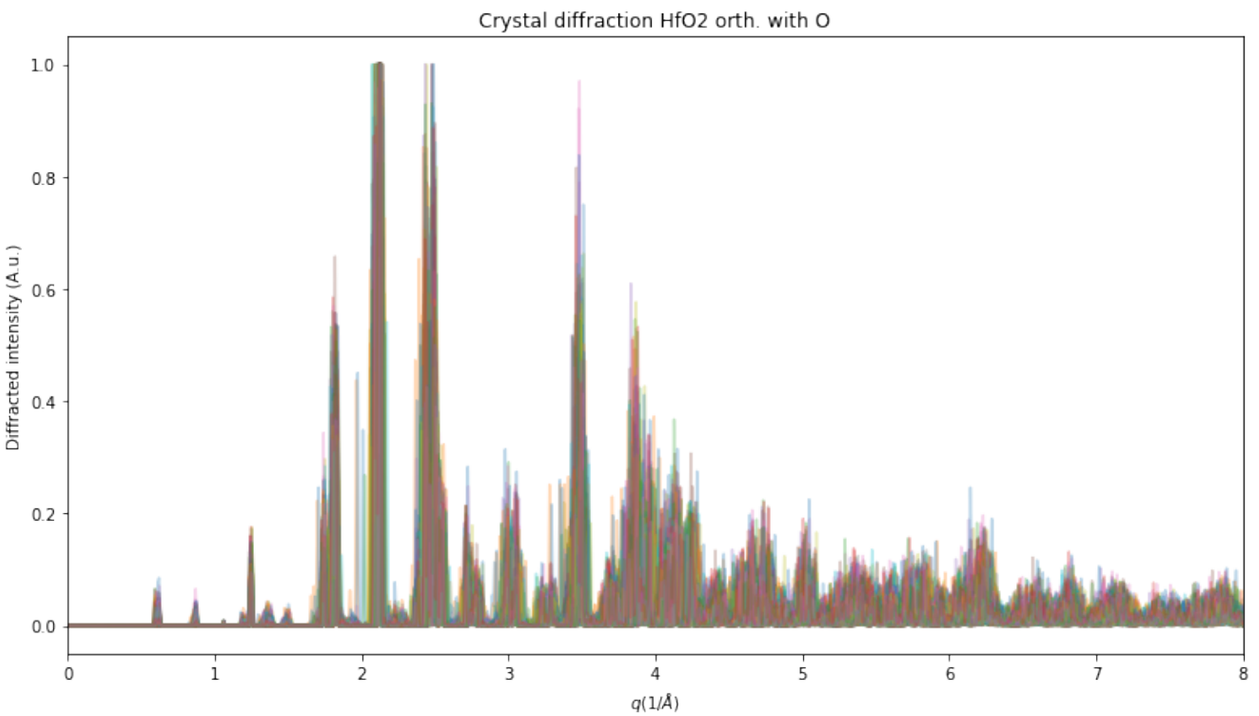


(c)

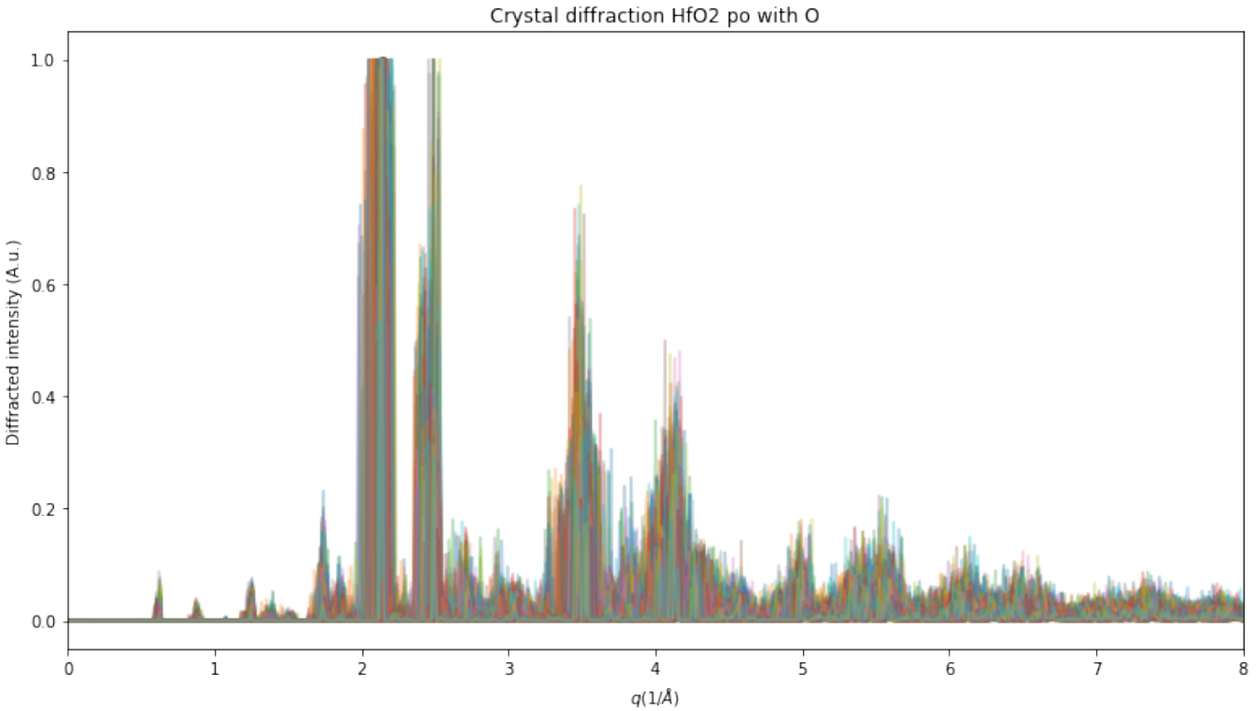
Abb. A.1: Winkelverteilung von α, β, γ über den gesamten Datensatz, aus Skript Data Exploration.ipynb


Crystal diffraction HfO2 monoklin with O
Diffracted intensity (A.u.)
q(1/Å)

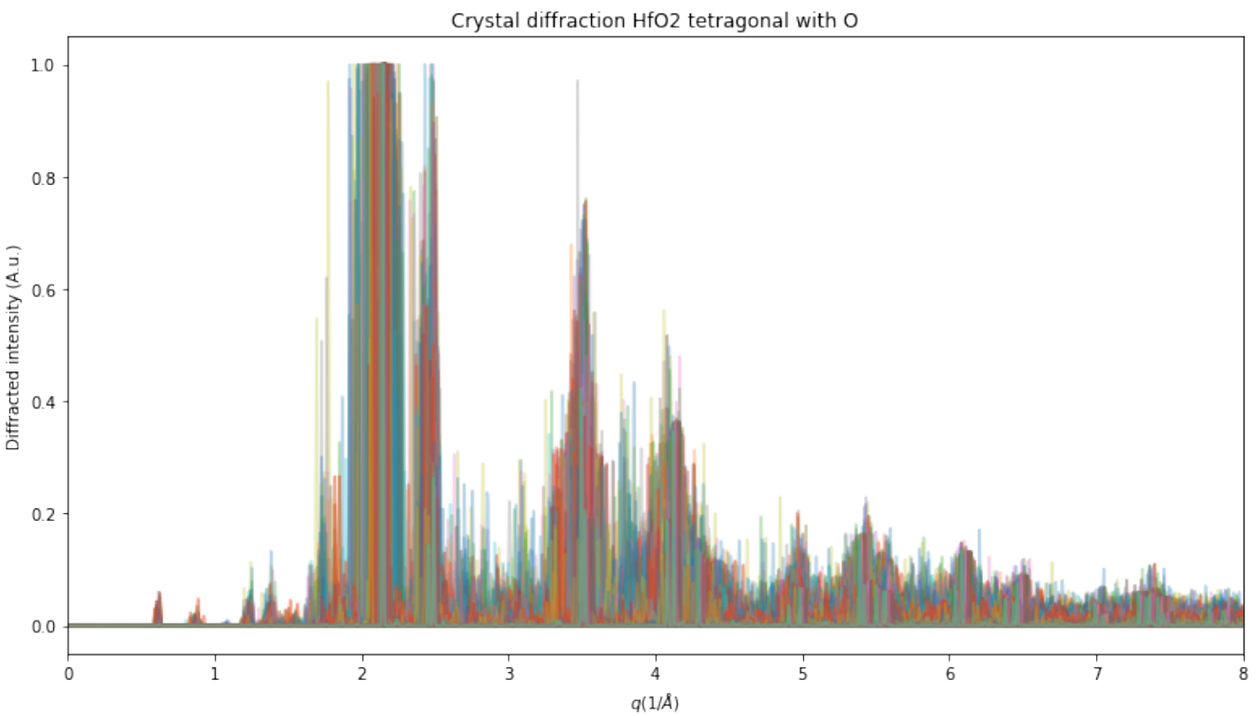
(a)


Crystal diffraction HfO2 orth. with O
Diffracted intensity (A.u.)
q(1/Å)

(b)



(c)



(d)

Abb. A.1: Röntgenstreuung an HfO_2 Kristall mit Phasen monoklin(a), orthorombisch (b), polar-orth. (c), tetragonal (d), Normalisiert auf 1.0

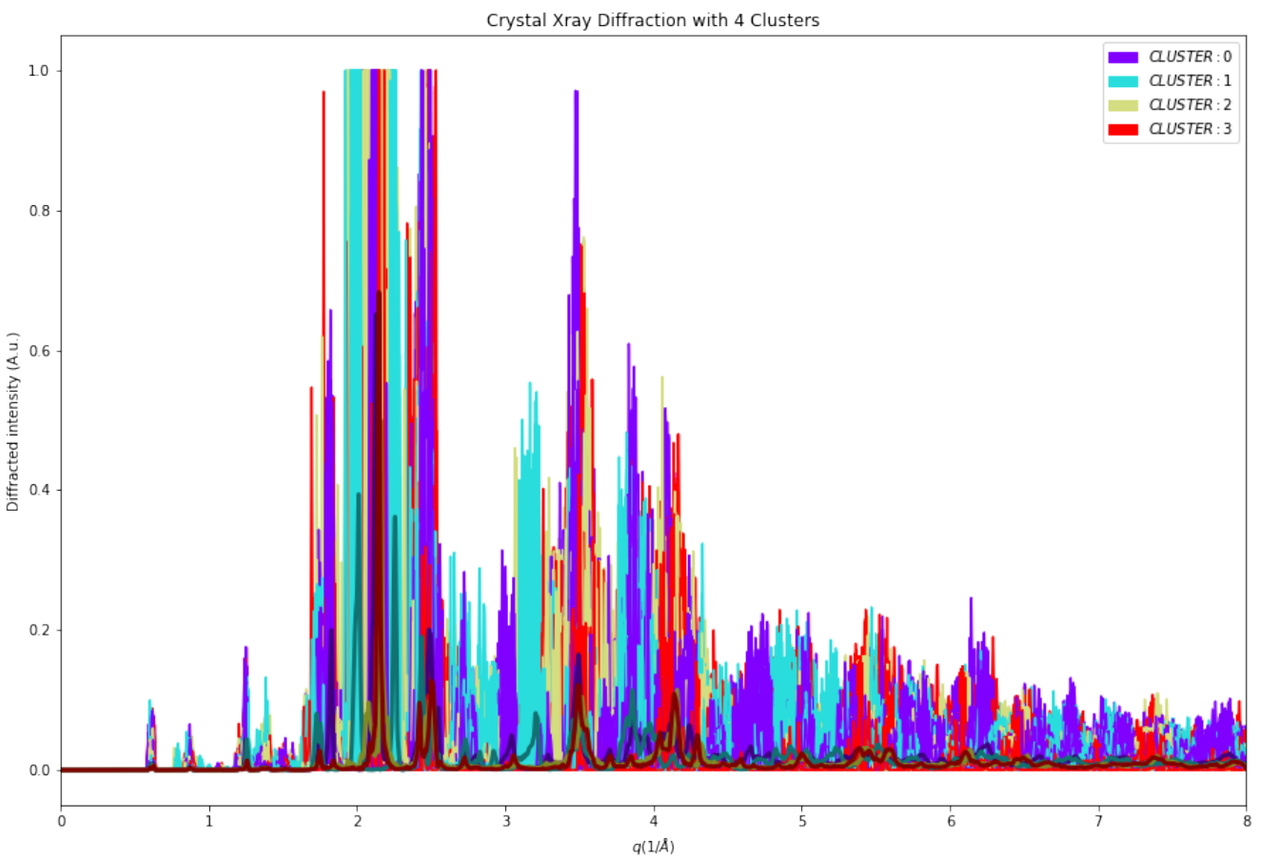
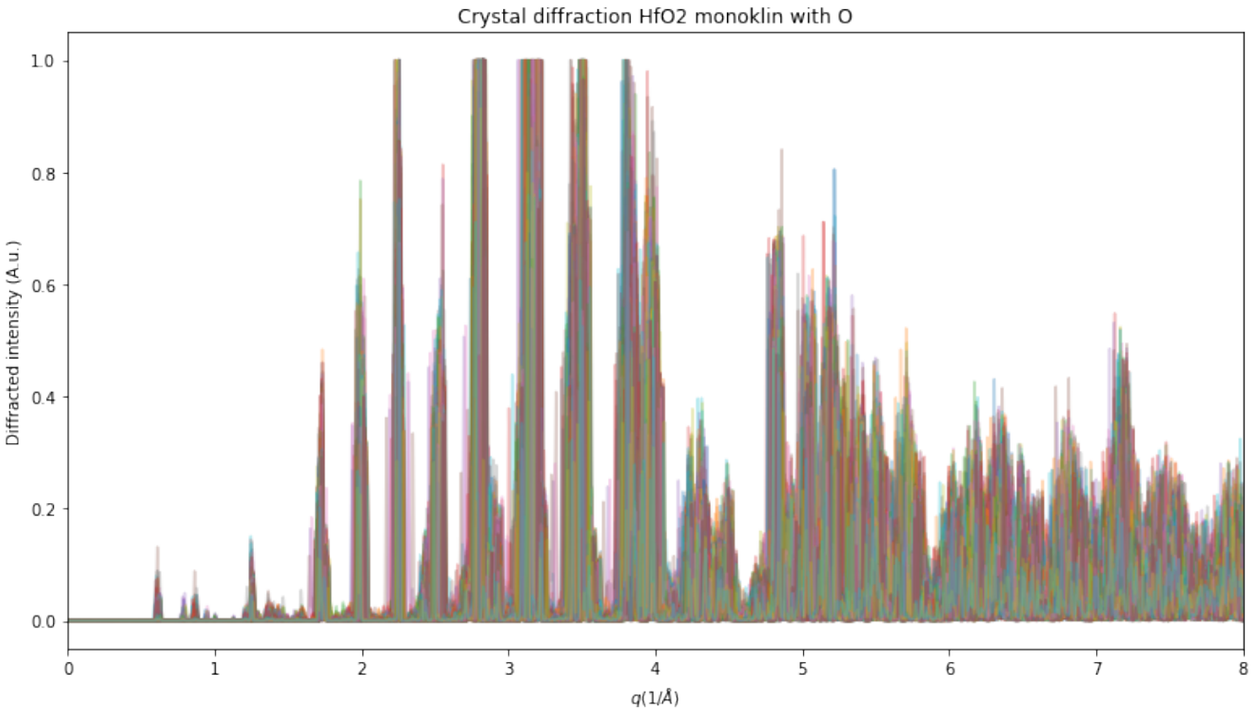
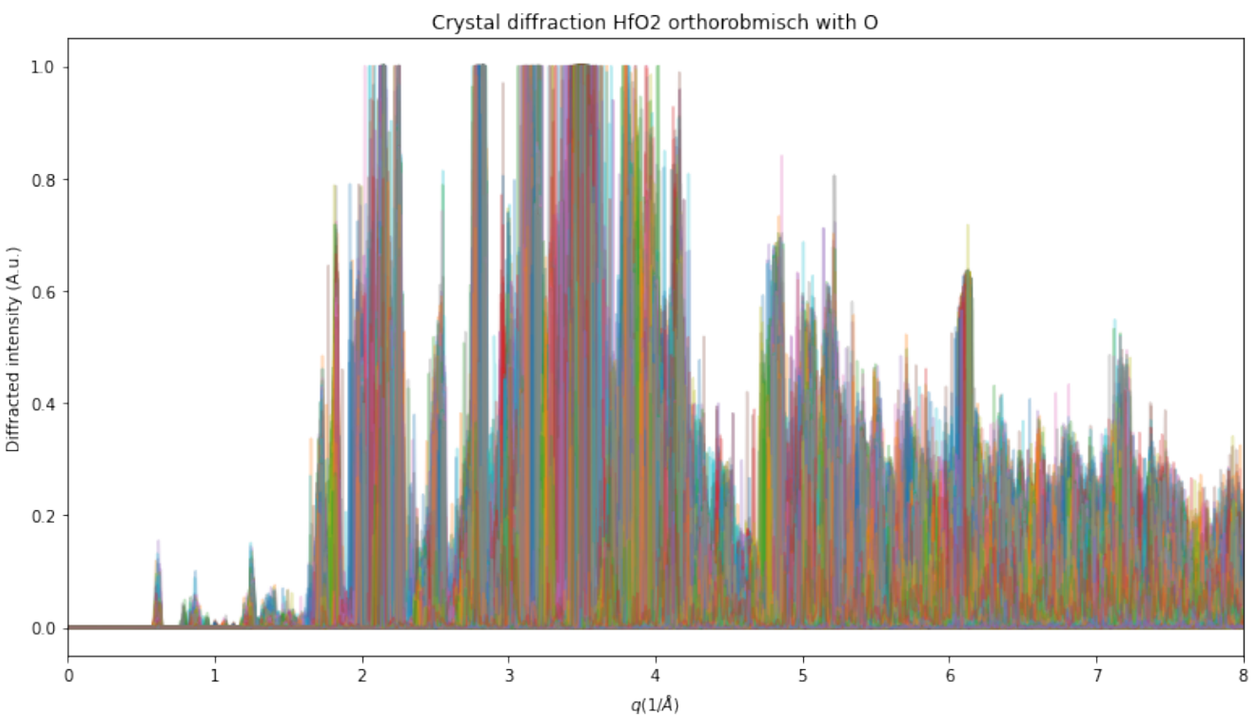


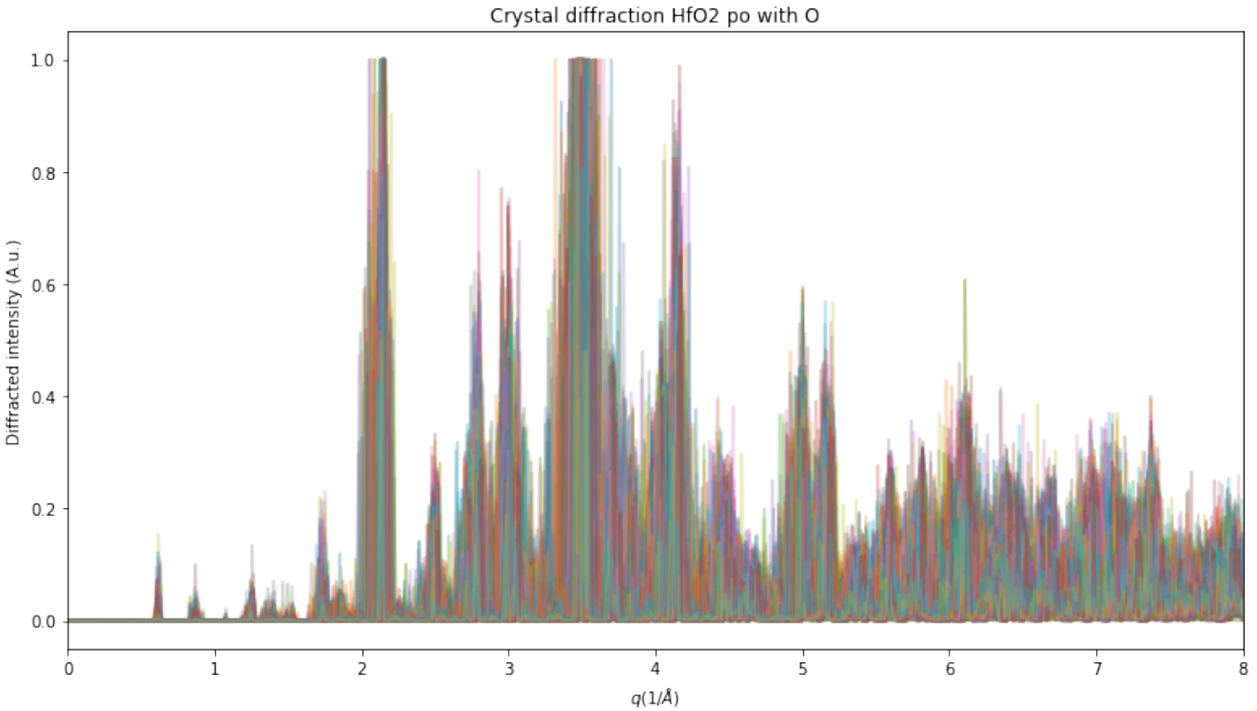
Abb. A.2: Verbesserte Erkennung durch Neutronenstreuung mit K-Means-Clustering Algorithmus in 4 Phasen, m (Cluster 1), o (Cluster 0), po (Cluster 3), t (Cluster 2)


Crystal diffraction HfO2 monoklin with O
Diffracted intensity (A.u.)
q(1/Å)

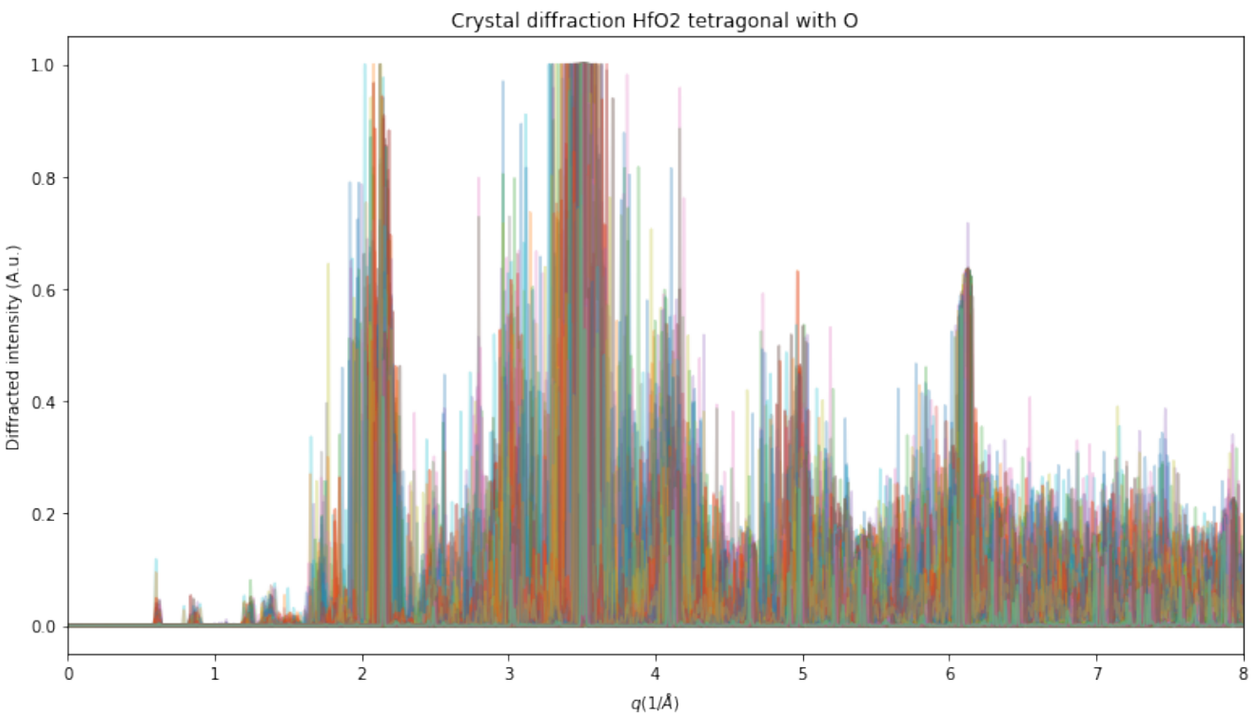
(a)


Crystal diffraction HfO2 orthorobmisch with O
Diffracted intensity (A.u.)
q(1/Å)

(b)



(c)



(d)

Abb. A.2: Neutronenstreuung an HfO_2 Kristall mit Phasen monoklin(a), orthorombisch (b), polar-orth. (c), tetragonal (d), Normalisiert auf 1.0

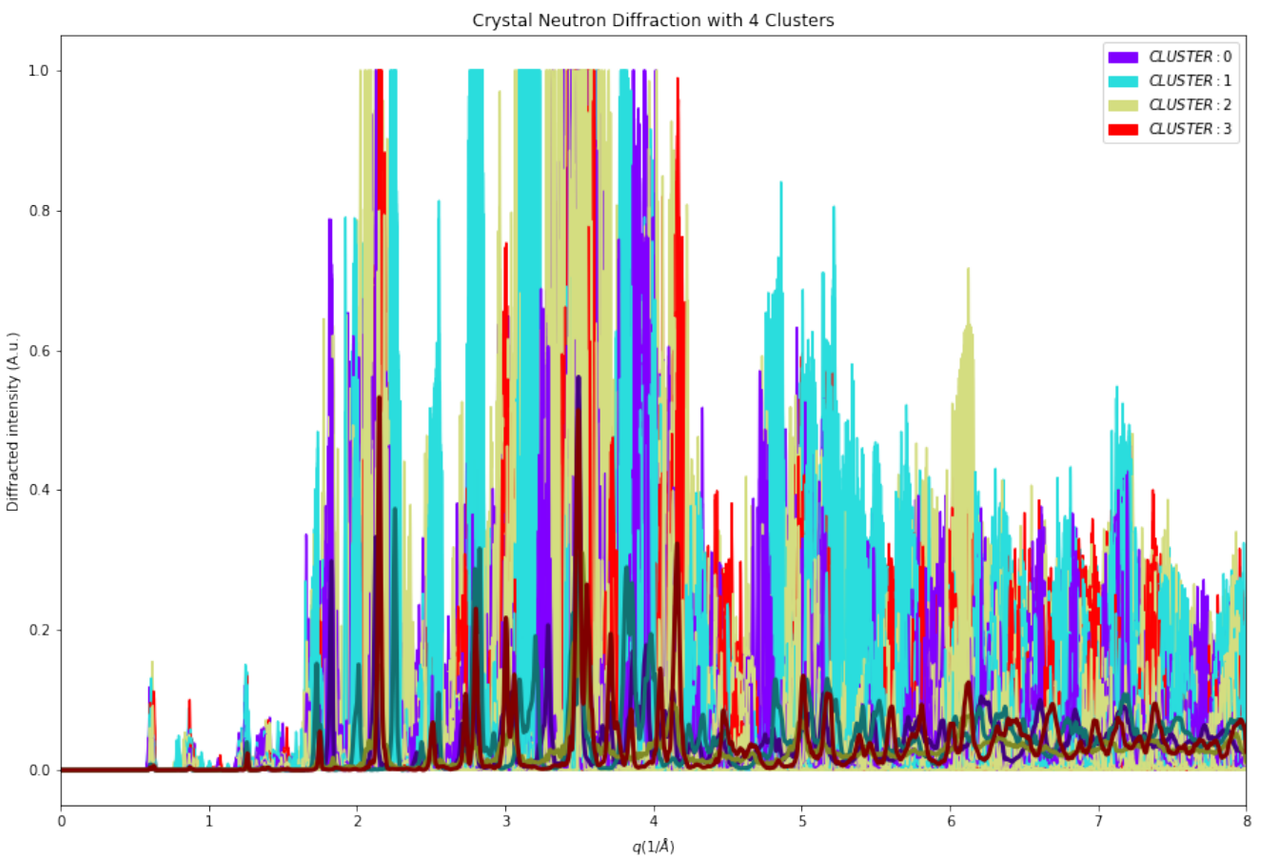


Abb. A.3: Verbesserte Erkennung durch Neutronenstreuung mit K-Means-Clustering Algorithmus in 4 Phasen, m (Cluster 1), o (Cluster 0), po (Cluster 3), t (Cluster 2)

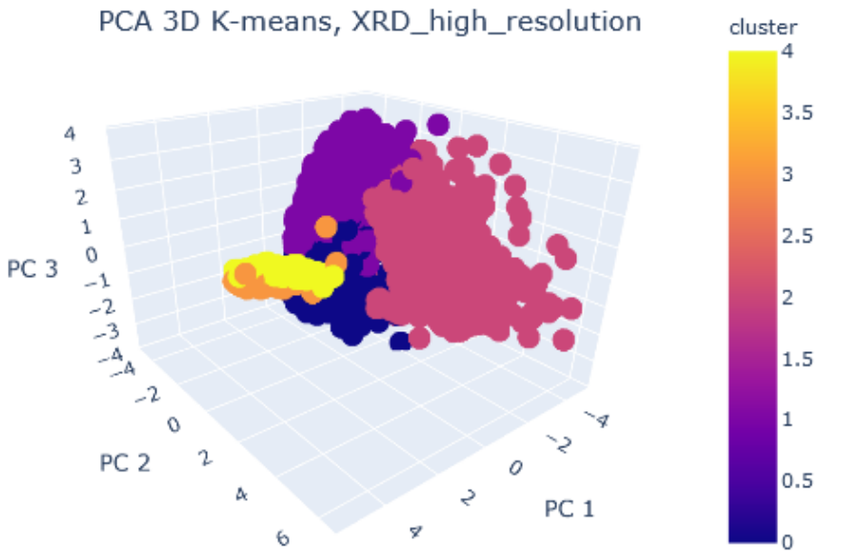


Abb. A.4: 3D PCA mit K-Means, XRD Auflösung 1725x16000

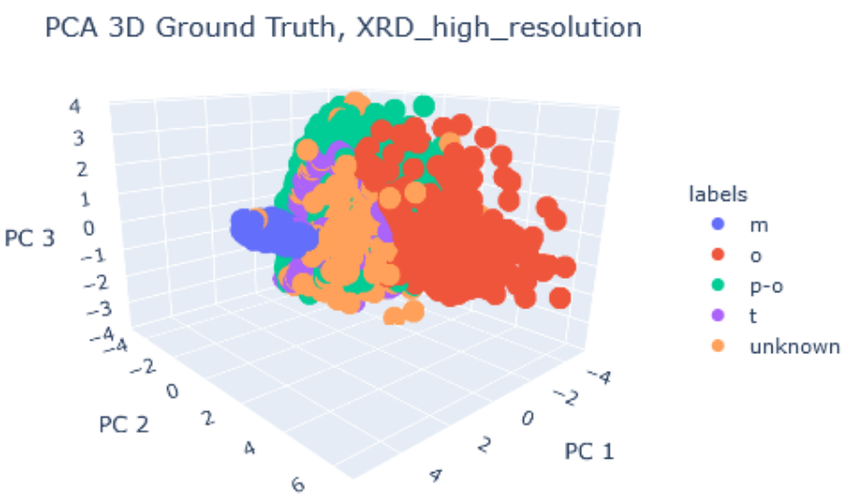


Abb. A.5: 3D PCA mit Ground-Truth-Labels, XRD Auflösung 1725x16000

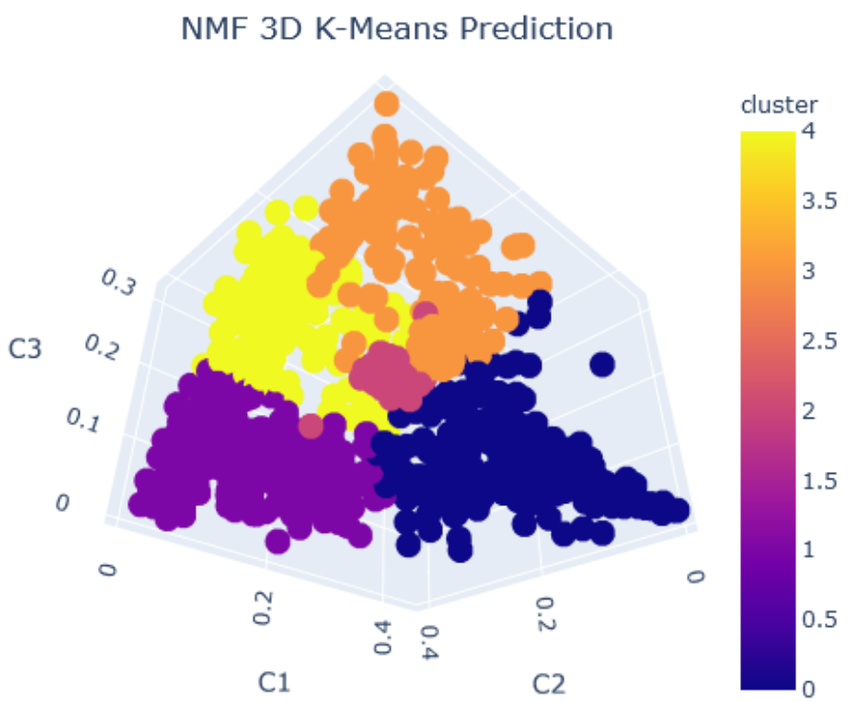


Abb. A.6: 3D NMF mit K-Means, XRD Auflösung 1725x16000

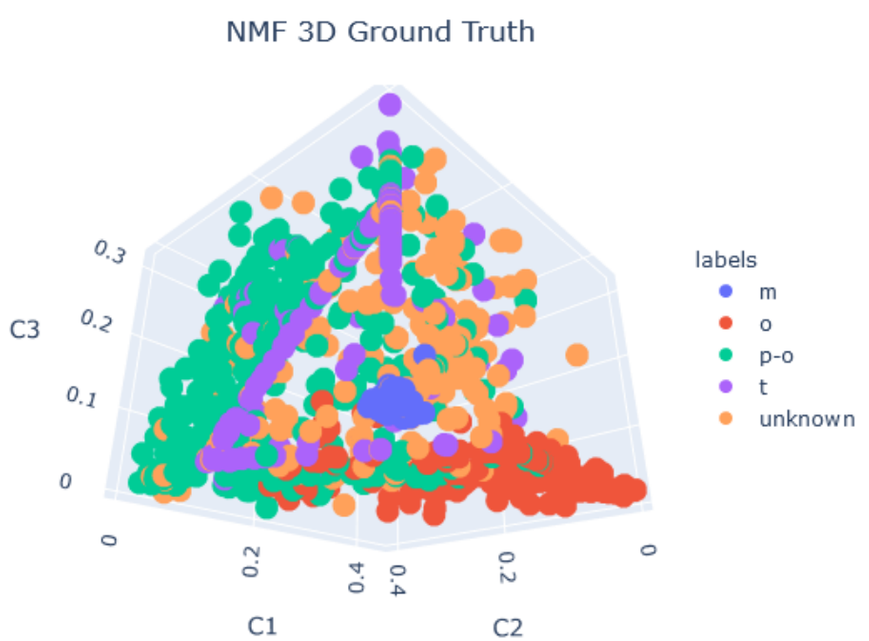


Abb. A.7: 3D NMF mit Ground-Truth-Labels, XRD Auflösung 1725x16000

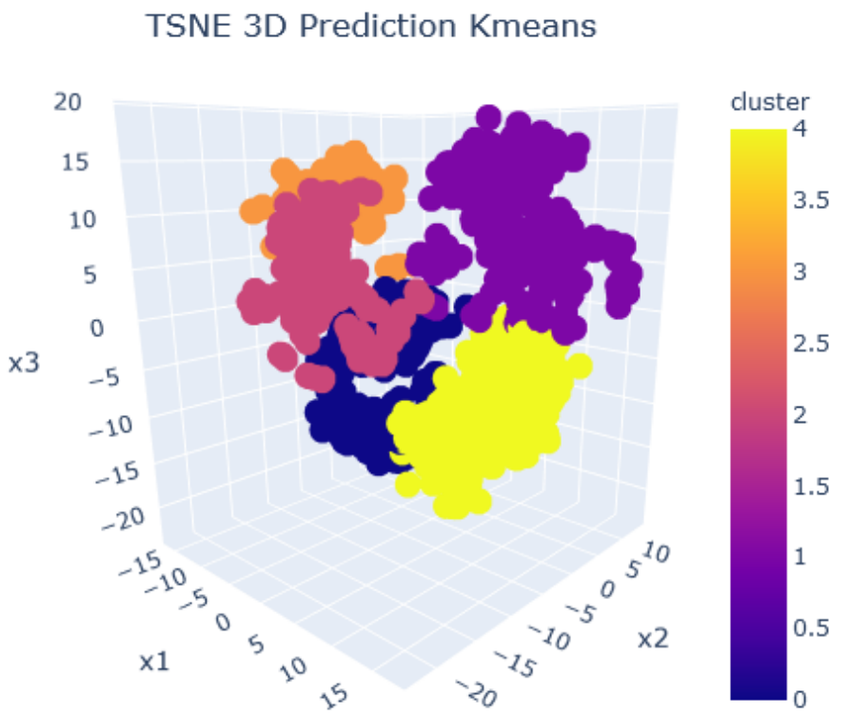


Abb. A.8: 3D TSNE mit K-Means, XRD Auflösung 1725x16000

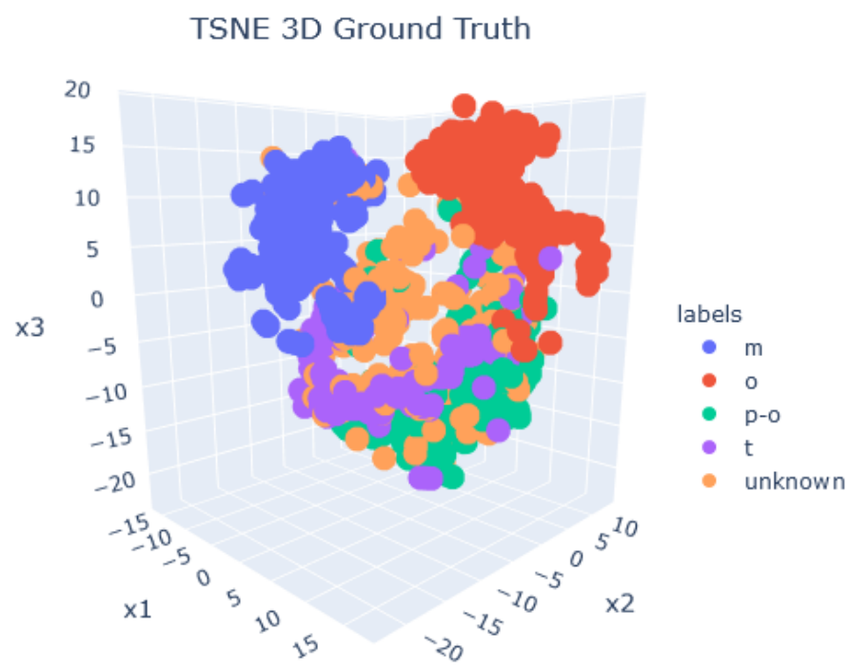


Abb. A.9: 3D TSNE mit Ground-Truth-Labels, XRD Auflösung 1725x16000

B. ALGORITHMEN

```
"""
@author: Maxi

Clustering with functional data in xrd pattern and skfda bib
"""

from ase.io import read
from crystals import Crystal
from skued import powdersim
import os
from skfda import FDataGrid
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import (train_test_split, GridSearchCV)
from sklearn.model_selection import StratifiedShuffleSplit
from skfda.ml.classification import KNeighborsClassifier
import skfda.preprocessing.smoothing.kernel_smoothers as ks
import skfda.preprocessing.smoothing.validation as val
from sklearn.model_selection import cross_val_score

def load_datagrid_sim(folder, arr, x):
    '''
    Load cif files from folder into ase crystal objects, simulate crystal
    diffraction and represent data in timeseries
    Args:
        folder (string): Where we load the data from.
        arr (list): List of cif filenames in folder.
        x (nparray): representation of 1/angström for powder simulation.

    Returns:
        fd (datagrid): representation of intensity values and 1/angström
        values in datagrid object.

    '''
    filename_arr = []
    crys_arr = []
    for file in arr:
        filename = os.sep.join([folder, file])
        filename_arr.append(file)
        crys = read(filename)
        crys = Crystal.from_ase(crys)

        diff = powdersim(crys, x, fwhm_l=50)
        diff_norm = diff / diff.max()
        crys_arr.append(diff_norm)

    crys_arr = np.array(crys_arr)
    fd = FDataGrid(crys_arr, x,
                   dataset_name='Diffraction Curves',
```

```
                    argument_names=[r'$q (1/\AA)$'],
                    coordinate_names=['Diffracted intensity (A.u.)'])
    print(fd)

    return fd


def smooth_datagrid(fd):
    '''
    Smoothes the values in the datagrid object

    Args:
        fd (datagrid): Includes data_matrix, x vector for contionous data
        representation.

    Returns:
        knn_fd (datagrid): datagrid values smoothed with an kN Kernel.

    '''
    # smooth the values with kNeighbors kernel
    param_values = np.linspace(start=2, stop=25, num=24)
    knn = val.SmoothingParameterSearch(ks.KNeighborsSmoother(), param_values)
    knn.fit(fd)
    knn_fd = knn.transform(fd)

    return knn_fd
```

Listing 13: Skript zur Simulation und Aufbereitung des XRD-Muster

```
"""
@author: Maxi

Build a dataframe with the crystal structures from the cif files for easy handling
    with the matminer lib.
"""

import pandas as pd
import numpy as np
import os

def load_data(folder, arr):
    '''
    Load cif files from folder into pymatgen crystal objects and parse them
    to structures.
    Args:
        folder (string): Where we load the data from.
        arr (list): List of cif filenames in folder.

    Returns:
        df (dataframe): representation of structures in pymatgen composition object
    .
    '''
```

```
    from pymatgen.io.cif import CifParser

    file_arr = []
    for file in arr:
        filename = os.sep.join([folder, file])
        parser = CifParser(filename)
        structure = parser.get_structures()[0]
        data = [structure]
        file_arr.append(data)

    df = pd.DataFrame(file_arr, columns=['structure'])
    return df

# specify abs path of folder, where you want to load the data from
folder = r"C:\Python\Projects\crystal-phase-prediction\crystal_data\zro2_del"
arr = [f for f in os.listdir(folder) if not f.endswith('.ini')]  # not include .ini
    files

df = load_data(folder, arr)
df['name'] = arr # create column with the filenames in folder

# change name to dataset that you created
pd.to_pickle(df, r"C:\Python\Projects\crystal-phase-prediction\pkl_files\
   structure_df_zro2_del.pkl")
```

Listing 14: Skript zur Vorbereitung der Deskriptoren mit Datentyp Composition

```
"""
@author: Maxi

We want to change the hfo2 or zro2 oxigen atoms with Hf or Zr, so that we can
detect them better in an XRD scan.
"""

from ase.io import read, write
import os


def change_spec_elements(folder, target_folder, arr, element):
    '''
    Load cif files from folder into ase crystal objects, change all elements
    O with specific other elements.
    '''
    filename_arr = []
    j = 0
    for file in arr:
        filename = os.sep.join([folder, file])
        filename_arr.append(file)
        structure = read(filename)
        symbols = structure.get_chemical_symbols()
        i = 0
        for symbol in symbols:
            if symbol == 'O':
                structure.symbols[i] = element
```

```
            i = i + 1

        print(structure.get_chemical_symbols())
        print(j)
        target_path = os.sep.join([target_folder, file])
        write(target_path, structure)
        j = j + 1

# select the element you want to switch in crystal
element = 'Zr'
folder = r'C:\Python\Projects\crystal-phase-prediction\hfo2' # load data from this
    folder
arr = [f for f in os.listdir(folder) if not f.endswith('.ini')] # ignore hidden
    files in folder
target_folder = r'C:\Python\Projects\crystal-phase-prediction\hfo2_del' # file
    where to store the new cif files

change_spec_elements(folder, target_folder, arr, element)
```

Listing 15: Ändern der Sauerstoffatome zu höherwertigen Elementen Zr und Hf

```
"""
@author: Maxi

We want to change the hfo2 or zro2 angles to 90/80 deg. and see if it has an impact
     on the clearness of
clustering.
"""

from ase.io import read, write
import os


def change_angles(folder, target_folder, arr):
    '''
    Load cif files from folder into ase crystal objects, change
    all elements angles with 80 or 90 deg.
    '''
    filename_arr = []
    j = 0
    for file in arr:
        filename = os.sep.join([folder, file])
        filename_arr.append(file)
        crystal = read(filename)
        unit_cell = crystal.get_cell_lengths_and_angles()
        i = 3
        for angle in unit_cell[3:6]:
            if(75.0 <= angle <= 85.0):
                unit_cell[i] = 80.0
            else:
                unit_cell[i] = 90.0
            i = i + 1

        crystal.set_cell(unit_cell)
```

```
        print(crystal.get_cell_lengths_and_angles())
        print(j)
        target_path = os.sep.join([target_folder, file])
        write(target_path, crystal)
        j = j + 1


folder = r'C:\Python\Projects\crystal-phase-prediction\crystal_data\hfo2' # load
    data from this folder
arr = [f for f in os.listdir(folder) if not f.endswith('.ini')] # ignore hidden
    files in folder
target_folder = r'C:\Python\Projects\crystal-phase-prediction\crystal_data\
    hfo2_90_80_deg' # file where to store the new cif files

change_angles(folder, target_folder, arr)
```

Listing 16: Ändern der Winkel zu festen Werten von 80 oder 90 Grad zur Verbesserung der XRD

```
import numpy as np
import pandas as pd

from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn.decomposition import NMF
from sklearn.manifold import TSNE

def kmeans_algo(df, n_clusters):
    '''Return the dataframe with predicted kmeans labels.'''
    kmeans = KMeans(n_clusters=n_clusters)
    labels = kmeans.fit_predict(df)
    df['cluster'] = labels
    return df


def pca_algo(X, comp):
    '''Return the PCA from Descriptor X.'''
    pca = PCA(n_components=comp)
    dfpca = pca.fit_transform(X)
    dfpca = pd.DataFrame(dfpca)
    return dfpca


def nmf_algo(X, comp):
    '''Return the Matrix Factorisation from Descriptor X.'''
    model = NMF(n_components=comp, init='random', random_state=0, max_iter = 8000)
    nmf_features_W = model.fit_transform(X)
    nmf_componentes_H = model.components_
    # nmf_df = pd.DataFrame(nmf_componentes_H.T)
    W_df = pd.DataFrame(nmf_features_W) # weights represent abundence of phase at a
    given nominal composition
    return W_df

```

```
36
37 def tsne_algo(X, perplex):
38         '''Return the TSNE from Descriptor X.'''
39     Xtsne = TSNE(n_components=3, perplexity=perplex).fit_transform(X)
40     dftsne = pd.DataFrame(Xtsne)
41     return dftsne
```

Listing 17: Pythonskript mit Funktionen zu häufig benötigten Algorithmen

```
1  import numpy as np
2  import pandas as pd
3  from sklearn.cluster import KMeans
4  from sklearn.decomposition import PCA
5  from sklearn.metrics import accuracy_score
6  import seaborn as sns
7  import matplotlib.pyplot as plt
8  import plotly.express as px
9  from algorithms import pca_algo, kmeans_algo, nmf_algo, tsne_algo
10
11 def sort_clusterlabels(df_kmeans):
12     """ Match the random cluster label 0, 1, 2,.. with the labels from phases
13     m, po, o, t, unk., by conting the occurences and match them to the new labels.
14     Parameters:
15     df_kmeans (dataframe): Dataframe with descriptor, labels and cluster columns
16
17     Returns:
18     df_kmeans: Matched cluster numbers together in labels and cluster column
19
20    """
21     df_times = df_kmeans.groupby(["labels", "cluster"]).size().reset_index(name="
       Time")
22
23     unknown_df = df_times[df_times['labels'].str.match('unknown')]
24     m_df = df_times[df_times['labels'].str.match('m')]
25     po_df = df_times[df_times['labels'].str.match('p-o')]
26     o_df = df_times[df_times['labels'].str.match('o')]
27     t_df = df_times[df_times['labels'].str.match('t')]
28
29     unk_newlabel = unknown_df.loc[unknown_df['Time'] == unknown_df.Time.max(), '
       cluster'].values[0]
30     m_newlabel = m_df.loc[m_df['Time'] == m_df.Time.max(), 'cluster'].values[0]
31     po_newlabel = po_df.loc[po_df['Time'] == po_df.Time.max(), 'cluster'].values[0]
32     o_newlabel = o_df.loc[o_df['Time'] == o_df.Time.max(), 'cluster'].values[0]
33     t_newlabel = t_df.loc[t_df['Time'] == t_df.Time.max(), 'cluster'].values[0]
34
35     df_kmeans = df_kmeans.replace({'m': m_newlabel,
36                                    'p-o': po_newlabel,
37                                    'o': o_newlabel,
38                                    't':t_newlabel ,
39                                    'unknown': unk_newlabel})
40     return df_kmeans
41
42
43 def name_change(df):
```

```
    """Change names of columns for plot2d and plot3d"""
    df3 = df.iloc[:, [0, 1, 2]].copy()
    df3 = df.rename(columns={0: 'x1', 1: 'x2', 2:'x3'})
    df3['cluster'] = df[['cluster']]
    df3['labels'] = df[['labels']]
    return df3

def plot2d(df3, title, clus_lab):
    """Visual 2d DR method with cluster or labels
    Parameters:
    df3 (Dataframe): name_changed dataframe with descriptor, cluster and labels
    title (string): tiles name
    clus_lab (string): select cluster or labels as keyword
    """
    lm = sns.lmplot(data=df3, x='x1', y='x2',hue=clus_lab, fit_reg=False, legend=
   True, legend_out=True)
    fig = lm.fig
    fig.suptitle(title, fontsize=10)

def plot3d(df3, title, clus_lab):
    """Visual 3d DR method with cluster or labels
    Parameters:
    df3 (Dataframe): name_changed dataframe with descriptor, cluster and labels
    title (string): tiles name
    clus_lab (string): select cluster or labels as keyword
    """
    fig = px.scatter_3d(
    df3, x='x1', y='x2', z='x3', color=df3[clus_lab],
    labels={'x1': 'C 1', 'x2': 'C 2', 'x3': 'C 3'}, title=title)
    fig.show()

def hyperparameter_testing(X, reduction_methode, hyperpar, labels_true):
    """ specific hyperparameter testing by selection on DR method
    Parameter:
    reduction_methode (string): select pca, nmf or tsne
    hyperpar (list): select choose of hyperparameter
    labels_true (array): string array with ground truth labels
    """
    performance = []
    i = 1
    for par in hyperpar:
        if(reduction_methode == 'pca'):
            df = pca_algo(X, par)
        elif(reduction_methode == 'nmf'):
            df = nmf_algo(X, par)
        elif(reduction_methode == 'tsne'):
            df = tsne_algo(X, par)

        df_kmeans = kmeans_algo(df, 5)
        df_kmeans['labels'] = labels_true
        df_kmeans = sort_clusterlabels(df_kmeans)
        acc = accuracy_score(df_kmeans['labels'], df_kmeans['cluster'])
        performance.append(acc)
        print("Round: " + str(i))
```

```
97          i = i + 1
98
99      perf_dic = dict(zip(hyperpar, performance))
100     print('Best value of performance: ' + str(max(perf_dic.values())) + '
    Hyperparameter = ' + str(max(perf_dic, key=perf_dic.get)))
101     print('Overview: ' + str(perf_dic))
```

Listing 18: Pythonskript mit Funktionen zu häufig benötigten Validierungs- und Testmethoden