Informatik
Hauptcampus

H O C H
S C H U L E
T R I E R

KI-gestützte Clusterung von studentischen Programmierlösungen zur Verbesserung automatisierter Feedbackprozesse

AI-supported clustering of student programming solutions to improve automated feedback processes

Gregor Germerodt

Bachelor-Abschlussarbeit

Betreuer: Prof. Dr. Michael Striewe

Trier, 07.07.2025

Kurzfassung

Der zunehmende Einsatz von Programmieraufgaben in der Hochschullehre stellt Lehrkräfte vor die Herausforderung, eine große Anzahl an studentischen Einreichungen effizient und gleichzeitig qualitativ hochwertig zu bewerten. Insbesondere der zeitliche Aufwand für individuelles Feedback nimmt mit wachsender Teilnehmerzahl erheblich zu. In dieser Arbeit wurde ein Verfahren zur KI-gestützten Clusterung von studentischen Programmierlösungen entwickelt, das den Bewertungsprozess entlasten und die Feedbackqualität verbessern soll.

Ziel war es, eine Pipeline zu implementieren, die Java-Dateien automatisiert einliest, in numerische Vektorrepräsentationen überführt und anschließend mittels Dimensionsreduktion und Clustering gruppiert. Durch diese Clusterungen sollen ähnliche Lösungen identifiziert werden, um repräsentative Beispiele pro Cluster für ein standardisiertes Feedback auszuwählen.

Im Verlauf der Arbeit wurden verschiedene Clustering- und Dimensionsreduktionsalgorithmen getestet und hinsichtlich ihrer Eignung bewertet. Besonderes Augenmerk lag auf der Konsistenz der Cluster bei großen und ungleichen Datensätzen sowie der Einbeziehung von Punktzahlen als ergänzendes Kriterium zur Verbesserung der Clusterqualität. Die entwickelte Lösung wurde modular umgesetzt und anhand eines beispielhaften Experiments von rund 1000 studentischen Einreichungen in Java evaluiert.

Die Ergebnisse zeigen, dass sich die entwickelte Pipeline prinzipiell eignet, Programmierlösungen automatisiert zu clustern und somit den Korrekturaufwand zu reduzieren. Gleichzeitig konnte aufgezeigt werden, dass eine rein syntaktisch-semantische Clusterung kontextabhängig nicht ausreicht, um homogene Gruppen für Feedbackzwecke zu bilden. Ergänzend wurden dafür Punktzahlenintervalle und die Konkatenation von Einreichungen eingebaut, um Clusterungen weiter einzugrenzen. Die Arbeit legt damit eine fundierte Grundlage für weiterführende Entwicklungen im Bereich automatisierter Feedbacksysteme für die Hochschullehre.

Abstract

The increasing use of programming tasks in university teaching presents teachers with the challenge of evaluating a large number of student submissions efficiently and at the same time ensuring high quality. In particular, the time required for individual feedback increases significantly as the number of participants grows. In this thesis, a method for AI-supported clustering of student programming solutions was developed to reduce the burden on the evaluation process and improve the quality of feedback.

The goal was to implement a pipeline that automatically reads Java files, converts them into numerical vector representations, and then groups them using dimension reduction and clustering. These clusters are intended to identify similar solutions in order to select representative examples per cluster for standardized feedback.

In the course of the work, various clustering and dimension reduction algorithms were tested and evaluated for their suitability. Particular attention was paid to the consistency of the clusters in large and unequal data sets and the inclusion of scores as a supplementary criterion for improving cluster quality. The developed solution was implemented in a modular fashion and evaluated using an exemplary experiment of around 1,000 student submissions in Java.

The results show that the developed pipeline is fundamentally suitable for automatically clustering programming solutions, thereby reducing the amount of correction work required. At the same time, it was shown that purely syntactic-semantic clustering is not sufficient in context-dependent cases to form homogeneous groups for feedback purposes. To this end, score intervals and the concatenation of submissions were incorporated to further narrow down the clusters. The work thus lays a solid foundation for further developments in the field of automated feedback systems for university teaching.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1

Einleitung und Problemstellung

Für Lehrkräfte an Hochschulen und Universitäten stellt das Prüfen und Bewerten studentischer Einreichungen eine zentrale, oftmals sehr zeitaufwändige Aufgabe dar. Insbesondere bei einer hohen Anzahl an Abgaben steigt der Korrekturaufwand erheblich, was den zeitlichen Rahmen für individuelles und qualitativ hochwertiges Feedback stark einschränken kann. Eine Untersuchung zeigt, dass vor allem die Bewertung studentischer Arbeiten als Hauptfaktor für Arbeitsbelastung und Beeinträchtigung des Wohlbefindens von Lehrkräften genannt wird (vgl. [JS21]). Im Bereich der Informatik betrifft dies insbesondere die Korrektur von Programmieraufgaben, bei denen jede Einreichung in Syntax und Semantik individuell gestaltet ist und dennoch konsistent und objektiv bewertet werden muss.

Einen Lösungsansatz bieten automatisierte Systeme zur Bewertung von Programmieraufgaben. Systematische Übersichtsarbeiten zeigen jedoch, dass viele dieser Systeme vorwiegend auf Unit-Tests oder statische Analysen setzen und meist lediglich generisches Feedback generieren (vgl. [MBKS]). Ein praktisches Beispiel hierfür ist das an der Hochschule Trier eingesetzte System ASB - Automatische Software-Bewertung[1]. Studierende laden dabei ihre bearbeiteten Aufgaben hoch, woraufhin das System prüft, ob alle erforderlichen Dateien vorhanden sind und die Benennungs- und Strukturvorgaben eingehalten wurden. Anschließend erfolgt eine dynamische Analyse durch Ausführung des Programms mit vordefinierten Testdaten. Entspricht die Ausgabe nicht den erwarteten Ergebnissen, wird eine Fehlermeldung generiert, die über die Funktionsfähigkeit einzelner Module oder des gesamten Programms informiert.

Das von solchen Systemen erzeugte Feedback dient in erster Linie der Orientierung, bietet jedoch meist wenig Unterstützung bei der Fehlersuche oder einer tiefergehenden Analyse der Lösung. Um die Qualität und Aussagekraft des Feedbacks zu erhöhen, könnten KI-gestützte Verfahren eingesetzt werden, die über rein statische oder testbasierte Ansätze hinausgehen.

Mit einem Schritt vor der Feedbackgenerierung befasst sich diese Arbeit. Ziel ist ein Verfahren zur KI-gestützten Clusterung studentischer Programmierlösungen zu entwickeln. Dabei sollen ähnliche Einreichungen anhand syntaktischer und semantischer Merkmale geclustert bzw. gruppiert werden, um es Lehrenden oder

[1] `https://www.hochschule-trier.de/informatik/forschung/projekte/asb`

automatisierten Systemen zu ermöglichen, ein repräsentatives Beispiel pro Cluster auszuwählen. Dieses kann als Grundlage für ein ausführliches Feedback dienen, das anschließend an alle anderen Teilnehmer des Clusters weiterleitet wird. Auf diese Weise ließe sich der Korrekturaufwand deutlich reduzieren und gleichzeitig mehr Spielraum für individualisiertes und qualitativ hochwertiges Feedback schaffen.

Das folgende Kapitel gibt eine Einführung in das Thema, indem es grundlegende Begriffe klärt, die in dieser Arbeit benutzt wurden. Der größte Teil des Kapitels Vorgehensweise und Implementierung besteht in der Beschreibung wie das Programm implementiert wurde. Der Verlauf wird in vier Abschnitte beschrieben, wobei der Letzte das fertige Programm vorstellt. Der benutzte Code dazu ist im Anhang zu finden. Im anschließenden Kapitel wurden mittels dieses Programms Experimente mit gestellten Testdatensätze durchgeführt. Zum Schluss werden die Resultate der Arbeit und des Arbeitsverlauf und der Ausblick diskutiert.

Begriffe wie "Verfahren" und "Algorithmen", sowie "Einreichungen" und "Lösungen" werden im Projekt gleichbedeutend behandelt.

2

Theoretische Grundlagen

2.1 Künstliche Intelligenz

Künstliche Intelligenz (KI) kann vielfältig definiert werden. Im Zusammenhang dieser Arbeit bezeichnet sie den Teilbereich der Informatik, der sich mit der Entwicklung von Systemen befasst, die in der Lage sind, Aufgaben zu lösen, für die normalerweise menschliche Intelligenz erforderlich ist. Dazu gehören unter anderem das Erkennen von Mustern, das Treffen von Entscheidungen, das Verstehen natürlicher Sprache und das Lernen aus Erfahrungen (vgl. [BLWW21]). Der Mensch hat sich damit Systeme geschaffen, um die Kapazitäten menschlicher Intelligenz für bestimmte Aufgaben zu schonen oder zu erweitern. Dabei übernimmt Künstliche Intelligenz im Rahmen intelligenter Automatisierung Aufgaben, die zuvor von Menschen durchgeführt wurden, indem sie diese selbstständig und konsistent ausführt und dabei über die Zeit hinweg lernen, sich anpassen und verbessern kann, um Prozesse effizient und fehlerarm zu gestalten (vgl. [CHTB20]). Während sich solche Systeme auf das Bildungswesen und hier speziell auf die Bewertung und Rückmeldung (Feedback) zu studentischen Programmierlösungen übertragen? Wie kann eine intelligente Automatisierung Lehrkräfte dabei unterstützen, qualitativ hochwertiges und individualisiertes Feedback zu generieren, ohne jede Lösung einzeln manuell prüfen zu müssen?

In dieser Arbeit wird KI anhand verschiedener Open-Source-Bibliotheken genutzt. Das System kombiniert mehrere KI-Techniken wie

- Deep Learning - ein Teilbereich des Machine Learnings (ML), der künstliche neuronale Netze mit vielen Schichten verwendet, um komplexe Muster in Daten zu erkennen,
- Unsupervised Machine Learning - ein Verfahren, bei denen Modelle ohne beschriftete Trainingsdaten Muster oder Strukturen in den Daten erkennen, z.B. durch Clustering, und
- andere verschiedene Machine Learning Methoden, bei denen Computer aus Beispieldaten eigenständig Muster und Zusammenhänge erkennen, um daraus Vorhersagen oder Entscheidungen abzuleiten.

2.2 Verwendete Algorithmen

2.2.1 Einbettung (Embedding)

Einer der ersten Schritte des Programms ist das Einbetten von Quellcode-Text der Programmierlösungen, dessen Erstellung im späteren Verlauf der Arbeit erklärt wird. Der Quellcode-Text wird in numerische hochdimensionale Vektor-Repräsentationen (Embeddings) umgewandelt. Diese numerischen Vektoren fassen semantische und strukturelle Merkmale des Codes zusammen und machen die Daten für anschließende Verfahren wie Dimensionsreduktion, Clustering und Visualisierung nutzbar. Hier wurde eine erste Erwähnung solch eines Verfahrens in der Arbeit von Orvalho et al. (2022, [OJM]) erfasst. Das dort beschriebene Verfahren CodeBERT stellte sich durch weitere Recherche für diese Arbeit als geeignet heraus. CodeBERT ist ein auf die Transformer-Architektur[1] basiertes Modell, das gleichzeitig mit natürlicher Sprache und Programmiercode (u.a. Java und Python) trainiert wurde. Es verwendet dabei machine learning (ML) wie Masked Language Modeling[2] und Replaced Token Detection[3], um inhaltlich sinnvolle und strukturierte Vektorrepräsentationen (Embeddings) für Code zu erzeugen (vgl. [FGT+]).

2.2.2 Dimensionsreduktion

Weiterhin wurden Verfahren eingebunden, die die durch das Embedding erstellten, hochdimensionale Vektoren in ihren Dimensionen reduzieren, um sie ebenso für weiterführende Prozesse wie z. B. zur Clusterung und besonders zur Visualisierung in den zwei- oder dreidimensionalen Raum nutzbar zu machen. In dieser Arbeit wurden folgende Verfahren benutzt:

- Principal Component Analysis (PCA) - projiziert hochdimensionale Daten in einen linearen Unterraum mit geringerer Dimension, indem neue Achsen, entlang derer die Daten am stärksten streuen, berechnet werden und stellt die Daten entlang dieser Achsen dar (vgl. [Kar01]),
- t-Distributed Stochastic Neighbor Embedding (t-SNE) - visualisiert hochdimensionaler Daten, indem es berechnet, wie ähnlich Punkte zu ihren Originaldaten sind, um sie entsprechend weit auseinander oder nahe zusammen zu platzieren (vgl. [Lau08]), und
- Uniform Manifold Approximation and Projection (UMAP) - nutzt Topologie und Geometrie, um skalierbare, strukturtreue Einbettungen in niedrigere Dimensionen zu erreichen (vgl. [MHM]).

Diese Algorithmen wurden bevorzugt, da sie aufgrund ihrer Verfügbarkeit in öffentlichen Bibliotheken in das vorgestellte Python Projekt einfach eingebunden werden konnten.

[1] Neuronales Netzwerkmodell, das mithilfe von Self-Attention-Mechanismen die Beziehungen zwischen Elementen in einer Sequenz erfasst und dadurch besonders leistungsfähig für Aufgaben mit Text- oder Code-Daten ist

[2] Trainingsmethode, bei der zufällig Wörter im Text verdeckt werden und das Modell lernen soll, diese fehlenden Wörter richtig vorherzusagen

[3] Trainingsmethode, bei der das Modell lernt zu erkennen, welche Wörter im Text durch andere ersetzt wurden, um so bessere Sprachrepräsentationen zu entwickeln.

2.2.3 Gruppierung (Clustering)

Das zentral angesprochene Verfahren ist das Clustering. Inspiration für diese Arbeit wurde aus aktuellen Werken entnommen, wie Orvalho et al. (2022), die mit InvAASTCluster ein Verfahren zur Clusterung von Programmierlösungen mittels dynamischer Invarianten-Analyse vorstellen (vgl. [OJM]); aus Paiva et al. (2024) die AsanasCluster, ein inkrementelles k-Means-basiertes Verfahren, zur Clusterung von Programmierlösungen für automatisiertes Feedback entwickelt haben (vgl. [PLF24]); und Tang et al. (2024) die Large Language Models[4] (LLMs) und Clustering kombinieren, um personalisiertes Feedback in Programmierkursen zu skalieren (vgl. [TWH+]).

Die Algorithmen dieser Quellen und weitere Recherche dienten zum Kennenlernen und zum späteren Einbinden von Algorithmen, die aufgrund ihrer besonderen Eignung zur Clusterung von Programmieraufgaben herausstachen wie

- k-Means - teilt N Beobachtungen in k Cluster auf, wobei jede Beobachtung zu dem Cluster mit dem nächstgelegenen Mittelwert gehört, der als Prototyp des Clusters dient (vgl. [Mac67]), und
- HDBSCAN - erweitert des Density-Based Spatial Clustering of Applications with Noise (DBSCAN) Algorithmus, indem es eine Hierarchie von Clustern aufbaut und die stabilen Cluster über unterschiedliche Dichteebenen hinweg extrahiert (vgl. [CMS]),

2.2.4 Visualisierung

Zur Visualisierung der zuvor geclusterten studentischen Programmierlösungen wurden die Python-Bibliotheken pandas und plotly.express verwendet:

- pandas: Dient zur effizienten Verarbeitung und Analyse von tabellarischen Daten. In diesem Fall wurden damit die Cluster-Zuordnungen und die zugehörigen Punktkoordinaten in einer DataFrame-Struktur verwaltet.
- plotly.express: Eine High-Level-Bibliothek für interaktive Diagramme. Sie wurde genutzt, um die studentischen Lösungen als farbige Punkte in einem Streudiagramm darzustellen, wobei die Farbe jeweils das zugehörige Cluster repräsentiert.

So konnte die Qualität und Trennschärfe der Clusterung visuell überprüft werden.

2.2.5 Evaluierung

Das Projekt wurde so erstellt, dass für ein beliebigen Clustering-Algorithmus ein Diagramm erstellt wird, in denen farbige Punkte die entsprechenden studentischen Lösungen repräsentieren. Um diesen Prozess zu bewerten wurden Evaluierungsverfahren etabliert. Nach Halkidi et al. 2001 bewerten interne Clustering-Evaluierungsverfahren die Qualität einer Clusterlösung anhand der Dichte innerhalb der Cluster und der Trennung zwischen den Clustern, ohne dabei externe

[4] auf Textdaten trainierte KI-Modelle, die natürliche Sprache verarbeiten und generieren

Referenzdaten heranzuziehen (vgl. [HBV01]). In dieser Arbeit wurden erstmalig Erwähnungen solcher Verfahren in [You24] entdeckt, wobei daraus nur zwei der benutzen Verfahren und erst durch weitere Recherche ein drittes hier eingebunden wurde. Folgende Auflistung beschreibt die benutzten Verfahren:

- Silhouette Score - berechnet für jeden Datenpunkt einen Silhouette-Wert, der die Qualität der Clusterzuordnung anhand der Abstände innerhalb und zwischen Clustern bewertet (vgl. [Rou87])
- Caliński-Harabasz Index - bewertet die Clusterqualität anhand des Verhältnisses von Streuung zwischen und innerhalb der Cluster. Das Verfahren wurde ursprünglich von Calinski und Harabasz (1974, [**?**]) eingefügt, eine Beschreibung findet sich in [HBV01].
- Davies-Bouldin Index - bewertet die Clusterqualität anhand des Verhältnisses von Intra-Cluster-Distanzen zu den Distanzen zwischen den Clustermittelpunkten. Das Verfahren wurde ursprünglich von Davies und Bouldin (1979, [DB79]) eingeführt, eine Beschreibung findet sich beispielsweise in der scikit-learn-Dokumentation[5].

[5] `https://scikit-learn.org/stable/modules/clustering.html#clustering-performance-evaluation`

3

Vorgehensweise und Implementierung

3.1 Themenfindung

Die zunehmende Digitalisierung und der technologische Fortschritt eröffnen vielfältige Einsatzmöglichkeiten für Methoden der künstlichen Intelligenz (KI). Besonders im Bildungsbereich entsteht dadurch die Chance, Prozesse zu optimieren und Lehrkräfte bei Routineaufgaben zu entlasten.

Vor diesem Hintergrund wurde das Thema dieser Arbeit gewählt. Ziel ist es, das Potenzial von KI-gestützten Verfahren im Kontext der automatisierten Auswertung studentischer Programmierlösungen zu untersuchen. Durch die Clusterung ähnlicher Lösungen können neue Ansätze für Feedbackprozesse entwickelt werden, die eine effizientere und gezieltere Betreuung von Studierenden ermöglichen.

3.2 Recherche und Vorbereitung

Zu Beginn wurden diverse Quellen herausgesucht, die sich im Rahmen dieses Themas bewegen. Besonders oft stach dabei der k-Means Algorithmus heraus oder wie dieser als Grundlage für erweiternde Algorithmen wie den InvAASTCluster (vgl. [OJM]) oder AsanasCluster (vgl. [PLF24]) benutzt wurde, um bessere Ergebnisse zu liefern. Anhand eines Rankings wurde die Relevanz der Quellen festgelegt, um das weitere Vorgehen einzugrenzen.

Andere Methoden wie der Caliński-Harabasz Index oder der Silhouette Score wurden erwähnt, die zur Evaluation des Clusterings dienen. Es wurde klar, dass der Verlauf von studentischen Programmierlösungen bis zu nutzbaren Daten zur Feedbackgenerierung ein mehrschrittiger Prozess ist.

Die folgenden Abschnitte, beschreiben die Entstehung des Programms bzw. einer Pipeline, die die benötigten Prozessschritte beinhaltet, um dies zu erreichen.

3.3 Entwicklungswerkzeuge

Das Projekt wurde über die frei verfügbare Entwicklungsumgebung Visual Studio Code (VSC) implementiert und über Git verwaltet. Aufgrund der weltweiten Etablierung und die dadurch gegebene vielfältige Auswahl an Bibliotheken

und online verfügbare Hilfestellungen, fiel die Wahl der Programmiersprache auf Python. Weiterhin wurde das Textsatzsystem LaTeX unter einer von der Hochschule Trier bereitgestellten Vorlagen zum Verfassen wissenschaftlicher Arbeiten[1] genutzt. Folgend eine Auflistung von Erweiterungen die VSC notwendigerweise oder unterstützend hinzugefügt wurden.

- Notwendig:
  - LaTeX Workshop (Kompilierung, Vorschau, Autovervollständigung)
  - Python
  - Pylance (effizienter language Server für Python)
- Unterstützend:
  - LaTeX language support (Syntax-Highlighting und Sprache für .tex-Dateien)
  - Python Debugger
  - isort (automatische Sortierung von Python-Imports)
  - Git Graph (Visualisierung von Arbeitsverlauf)

3.4 Implementierungsverlauf

3.4.1 Erster Implementierungsabschnitt

Um einen mehrschrittigen Prozess zu vereinen, eignete sich das Prinzip einer Pipeline, in der nacheinander Algorithmen ablaufen, dessen Eingabewerte vom vorherigen Algorithmus abhängen und dessen Ausgabewerte dem Nächsten folglich wieder als Eingabe dienen.

Dateien laden

Als erstes musste es eine Möglichkeit geben die Java-Dateien der studentischen Programmierlösungen einlesen zu können. Diese wurden als Datensätze durch den betreuenden Prof. Dr. Striewe bereitgestellt.

Der Datensatz an denen das Programm fortlaufend getestet wurde, bestand aus mehreren Überordnern und final aus drei Java-Dateien und einer Text-Datei, welche den Studierenden vorgegeben waren und vervollständigt werden mussten. Jedoch soll die Menge der Java-Dateien keine überwiegende Rolle spielen.

Das Programm musste also in der Lage sein Ordner zu durchsuchen und Java-Dateien zu erkennen. Dies ermöglichte eine Methode des ersten implementierten Moduls `data_loader.py`. Es extrahierte Quellcode-Text und speicherte pro Datei Code-Schnipsel (Code Snippets) in eine Liste und gab diese an die Pipeline zurück.

Anfangs entstanden durch problematische Zeichen innerhalb der Java-Dateien Fehlermeldungen, welche jedoch einfachheitshalber ignoriert werden.

[1] `https://www.hochschule-trier.de/informatik`

Vektorrepräsentation

Als nächstes folgte ein Modul zum Einbetten dieser Code Snippets bzw. zum dessen repräsentieren durch hochdimensionale Vektoren. Dies war notwendig, um sie für Clustering-Algorithmen vorzubereiten.

Dafür wurde das Modul `embedding_model.py` erstellt, welches vortrainierte Sprachmodelle aus der Transformers-Bibliothek[2] von Python (hier CodeBERT) und passende Tokenizer, die den Code vorbereitend für den Transformer in Tokens zerlegt, importiert.

Anhand einer Methode werden die Code Snippets in numerische Vektoren umwandelt (Embeddings). Alle entstandenen Embeddings wurde anschließend jeweils an die Pipeline zurückgegeben und in eine Liste abgespeichert.

Clustering

Nun mussten die Embeddings geclustert werden. Das entsprechend erstellte Modul `clustering_engine.py` importierte dafür die beiden Cluster Algorithmen k-Means aus der scikit-learn Bibliothek[3] für maschinelles Lernen und HDBSCAN aus der separaten HDBSCAN Python-Bibliothek[4]. An die Pipeline werden schließlich mit Markierungen (Labels) versehene Cluster zurückgegeben.

Evaluationsmetriken

Anschließend wurde ein Evaluationsverfahren eingebaut, um die Qualität der Cluster zu bewerten. Dafür wurde das Modul `evaluation_metrics.py` implementiert. Darin wird jeder Cluster nach den Evaluationsmetriken Silhouette Score[5], Caliński-Harabasz Index[6] und den Davies-Bouldin Index[7] getestet, welche ebenfalls aus der scikit-learn Bibliothek importiert wurden. Zurückgegeben wird ein Dictionary mit den drei Zahlenwerten der Evaluationsmetriken.

Tests, Optimierungen und Dokumentation

Jedes implementierte Modul wurde einzeln getestet. Dazu wurden das Ergebnis und die Zeit, welche für den jeweiligen Prozessschritt notwendig waren durch print-Anweisungen ausgegeben. Dabei stellte sich heraus, dass Embedding und besonders Imports relativ viel Zeit in Anspruch nahmen.

[2] `https://huggingface.co/docs/transformers`
[3] `https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html`
[4] `https://hdbscan.readthedocs.io/en/latest/api.html#hdbscan.hdbscan.HDBSCAN`
[5] `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html`
[6] `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.calinski_harabasz_score.html`
[7] `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.davies_bouldin_score.html`

Da die zu testenden Datensätze teilweise aus mehreren hunderten Dateien bestanden, wurde dazu Caching der Embeddings eingeführt, um das Testen des Zusammenspiels der einzelnen Module zu beschleunigen. Versuche die Imports durch Lazy Loading zu beschleunigen waren in diesem Fall nur geringfügig merkbar.

Des Weiteren wurde eine Requirements.txt-Datei erstellt. Diese beinhaltet sämtliche Information über die aktuell im Projekt benutzen Versionen der importierten Bibliotheken. Sie sorgt, dass für andere Nutzende gleiche Bedingungen wie auch in der Entwicklung herrschen, um ein lauffähiges Programm zu gewährleisten.

Die noch später hinzugefügte Datei HowToInstall.txt dient zusätzlich als Schritt-für-Schritt-Anleitung. Weiterhin wurde eine config.yaml-Datei hinzugefügt. Diese diente als zentrale Ansprechquelle der Pipeline für Parameter.

Dimensionsreduktion

Zu diesem Zeitpunkt war ein Visualisierungs-Modul geplant, um die Clusterungen sichtbar zu machen, doch bevor es sinnvoll eingesetzt werden konnte, wurde noch ein weiterer Prozessschritt eingebunden, das Dimensionsreduktionsverfahren bzw. dessen Algorithmen. Das entsprechende Modul `dimensionality_reducer.py` importiert auch hier Algorithmen aus der scikit-learn Bibliothek, die Principal Component Analysis (PCA)[8] und t-Distributed Stochastic Neighbor Embedding (t-SNE)[9]. Der dritte Algorithmus Uniform Manifold Approximation and Projection (UMAP)[10] stammt aus der eigenständigen UMAP-learn Bibliothek.

Dieser Prozessschritt findet zwischen Embedding und Clustering statt. Er reduziert die Embeddings bzw. die hochdimensionalen Vektoren in ihren Dimensionen (hier zu 2D oder 3D), welche danach zur Clusterung weitergereicht und dadurch auch besser als menschlicher Betrachter in einem Diagramm beobachtet werden können.

Visualisierung

Schließlich wurde noch das Modul `cluster_plotter.py` implementiert, welches anhand der dimensionsreduzierten Embeddings und der aus dem Clustering hervorgegangenen Labels ein statisches Diagramm (engl. plot) in einem separaten Fenster erstellt (Beispiel-Clusterung in Abbildung 3.1). Hierfür wurden aus der Matplotlibs Bibliothek die Plotting-Module pyplot[11] und `mpl_toolkits.mplot3d`[12] importiert, die für 2D- und 3D-Visualisierung (falls gewünscht) zuständig sind.

Die zuvor erstellten Module glichen sich durch ihre einheitlichen Klassenstruktur, da jedoch für dieses Modul keine Zwischenspeicherung von Zuständen anhand einer Instanz notwendig war, wurden dessen Methoden statisch definiert.

[8] `https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html`
[9] `https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html`
[10] `https://umap-learn.readthedocs.io/en/latest/`
[11] `https://matplotlib.org/stable/api/pyplot_summary.html`
[12] `https://matplotlib.org/stable/gallery/mplot3d/2dcollections3d.html#sphx-glr-gallery-mplot3d-2dcollections3d-py`

Pipeline

Die Pipeline besteht zu diesem Zeitpunkt aus folgendem Ablauf:
Daten laden → Einbetten → Dimensionen reduzieren → Clustern → Evaluieren → Visualisieren

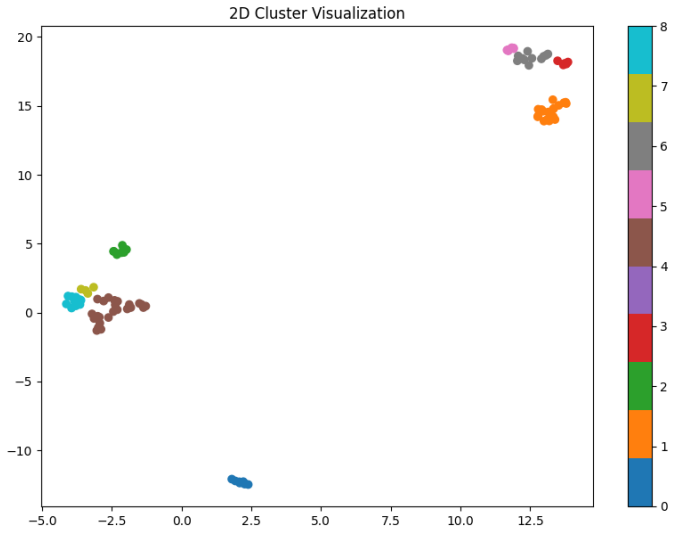

Abbildung 3.1: Zweidimensionales Clustering-Diagramm einer Clusterung von 147 Java-Dateien. Die unterschiedlichen Farben kennzeichnen die Zugehörigkeit der Punkte zu einem Cluster. Am rechten Rand sind die benutzten Farben in einer Farbskala angeordnet.

3.4.2 Zweiter Implementierungsabschnitt

Interaktive Diagramme

In Abbildung 3.2 sind zwar farbige Punkte in unterschiedlicher Anzahl pro Cluster zu erkennen, jedoch wurden keine Labels angezeigt, die Informationen über die Punkte anzeigen sollten. Als vorbereitender Schritt für weiterführende Prozesse zur automatischen Feedbackgenerierung, müssen sie zumindest zum Testen visuell zuordenbar sein.

Aus diesem Grund wurde ein weiteres Modul zur Visualisierung implementiert. Das entstandene Modul `interactive_plot.py` nahm dafür wieder Embeddings, Labels und zusätzlich noch den Namen der aktuellen Java-Datei entgegen, die im `data_loader.py`-Modul gespeichert wurde.

Dazu wurden aus der Python-Bibliothek die Module Pandas[13] und Plotly Express[14] importiert, die einerseits zur Erstellung von Tabellenstrukturen (DataF-

[13] `https://pandas.pydata.org/docs/`
[14] `https://plotly.com/python/plotly-express/`

rames) und andererseits für einfache und interaktive Diagramme zuständig sind. Ausgeführt, öffnete sich ein Fenster im Webbrowser mit einem von der Gestaltung her ähnlichem Diagramm wie in der statischen Variante (3.1).

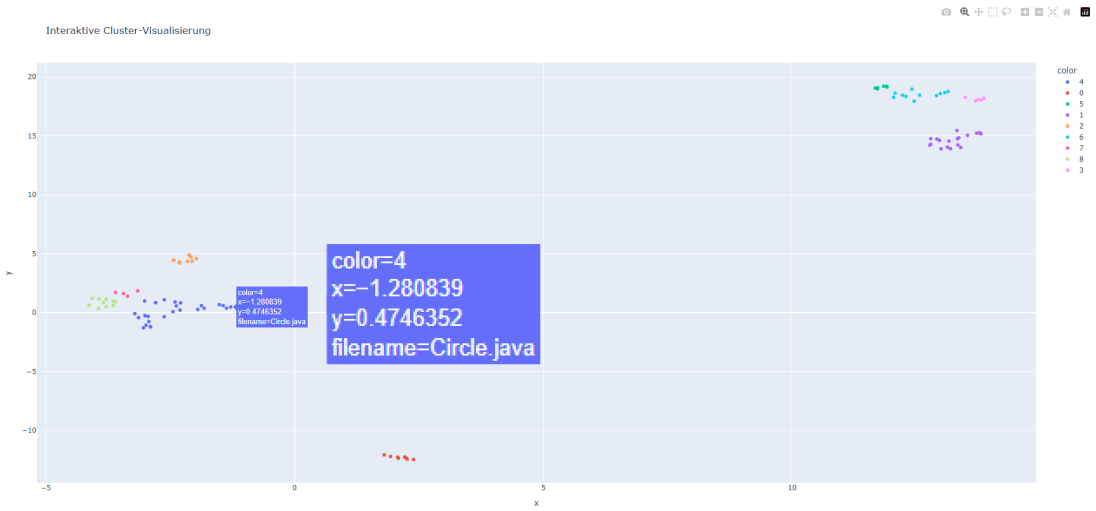

Abbildung 3.2: Interaktives Clustering-Diagramm einer Clusterung von 147 Java-Dateien. Durch das Halten der Maus über die Punkte werden Informationen über sie angezeigt (im Bild wiederholt vergrößert dargestellt). Am rechten Rand sind die Mengen der Cluster angezeigt.

Probleme bei der Cluster-Konsistenz und getroffene Maßnahmen

Beim mehrmaligen Austesten verschiedener Dateimengen und Überprüfen der ausgegebenen Punkte, fiel auf, dass Cluster bei größeren Datenmengen nicht mehr konsistent sind, obwohl akzeptable Ergebnisse im Evaluationsmetriken erreicht wurden. Genauer genommen wurden vereinzelt unterschiedliche Dateien in einem gemeinsamen Cluster platziert (gezeigt in 4.1.2 in Abschnitt Forschungsergebnisse). So kamen beispielsweise Point.java-Dateien in einem Cluster mit sonst nur Circle.java-Dateien vor.

Da Clustering-Algorithmen nach ähnlicher Syntax und Semantik clustern, kann solch ein Verhalten durchaus vorkommen, ist hier jedoch nicht von praktischem Nutzen. Andererseits könnten ebenso Fehler in den Algorithmen oder Inkonsistenzen in den gegebenen Datensätzen die fehlerhafte Clusterung verursachen. Selbst Dateien die abhängig von der gestellten Aufgabe zu den Einreichungen bereits gegeben waren, nicht bearbeitet werden sollten und überall identisch waren, wurden in verschiedenen Clustern angeordnet.

Als erste Maßnahme gegen dieses Problem, wurden die nicht zu bearbeitenden Dateien ignoriert, indem nicht mehr allgemein nach.java-Dateien gesucht wird, sondern nur noch nach bestimmten Namen.

Im weiteren Verlauf wurde zudem in der Pipeline eine Schleife ergänzt, die die Prozesse ab Embedding bis zur Visualisierung für die gewünschten, in der Konfigurationsdatei festgelegten Namen bzw. Java-Dateien wiederholt. Dadurch werden gemischte Cluster verhindert und für jede unterschiedene Java-Datei ein Diagramm erstellt.

Weiterhin ist beim Testen einer niedrigen Anzahl von Dateien ($n \leq 4$) aufgefallen, dass der Embedding-Algorithmus nicht mehr funktioniert, jedoch ist solche eine Clusterung ohnehin nicht von Nutzen.

Experimentierungspipeline

Um die Evaluationsmetriken einfach testen zu können, wurde eine separate Experimentierungspipeline erstellt. Der Vorteil bestand darin, dass die verschiedenen Kombinationen der Dimensionsreduktions- und Clustering-Algorithmen mit ihren Parametern über Dictionaries innerhalb der Pipeline definiert wurden. Die Evaluationsergebnisse wurden anschließend in einer CSV-Datei im Projektverzeichnis festgehalten.

In den Tabellen 4.1 und 4.2 im Abschnitt Forschungsergebnisse wurde gezeigt, welche Algorithmen-Kombination für verschiedene Dateimengen geeignet sind.

Visuelle Erweiterung

Um den Punkten im Diagramm mehr Informationen entnehmen zu können, wurde neben den Dateinamen nun noch der Name des Einreichungsordners hinzugefügt. Da die erreichten Punktzahlen der Einreichungen Teil des Ordnernamens sind, konnten die Clusterungen jetzt besser nachvollzogen und überprüft werden. Weiterhin wurde das `interactive_plot.py`-Modul um die Option das Diagramm als dreidimensionale Umgebung darzustellen erweitert. Sollten mehrere Cluster im 2D-Diagramm aufeinanderliegen, so kann die dritte Achse bessere Einsicht gewährleisten.

3.4.3 Dritter Implementierungsabschnitt

Konkatenation gesuchter Dateien

Auch wenn für jede durch Namen getrennte Art Datei ein separates Diagramm erstellt wird, besteht die Möglichkeit, dass die vollständige Einreichung bzw. Lösung zu den gestellten Aufgaben gesamt betrachtet, werden sollte, da es sonst zu verminderter Information führen könnte. Um die Dateien als ein Ganzes zu betrachten, wurde das `data_loader.py`-Modul um eine Konkatenationsfunktionalität ergänzt. Die entstandene Methode konkateniert alle gesuchten Java-Dateien eines Einreichungsordners. Der im Diagramm gezeigte Dateiname eines Punktes, setzt sich nun aus den konkatenierten Namen der Dateien zusammen. Die Schleife in der Pipeline die die Prozessschritte für jede gesuchte Art Datei wiederholte, wurde daraufhin entfernt.

Probleme bei der Cluster-Visualisierung und Punktzuordnung

Theoretisch wäre das Programm in der Lage, Embeddings ohne Dimensionsreduktionsalgorithmen zu verarbeiten. Zum Testen wird jedoch weiterhin eine geeignete Visualisierung verwendet, bei der allerdings durch die Anzeige der Ordnernamen pro Punkt ein weiteres Problem auffiel. So werden konkatenierte Dateien in einem Cluster gesteckt, dessen Ordnernamen sich durch stark variierende Punktzahlen unterscheiden, wie in Abbildung 3.3 zu sehen ist. Auch hier kann solch ein Verahlten durchaus vorkommen, ist aber nicht von praktischem Nutzen, da sich das zu generierende Feedback nach den erreichten Punktzahlen unterscheiden sollte.

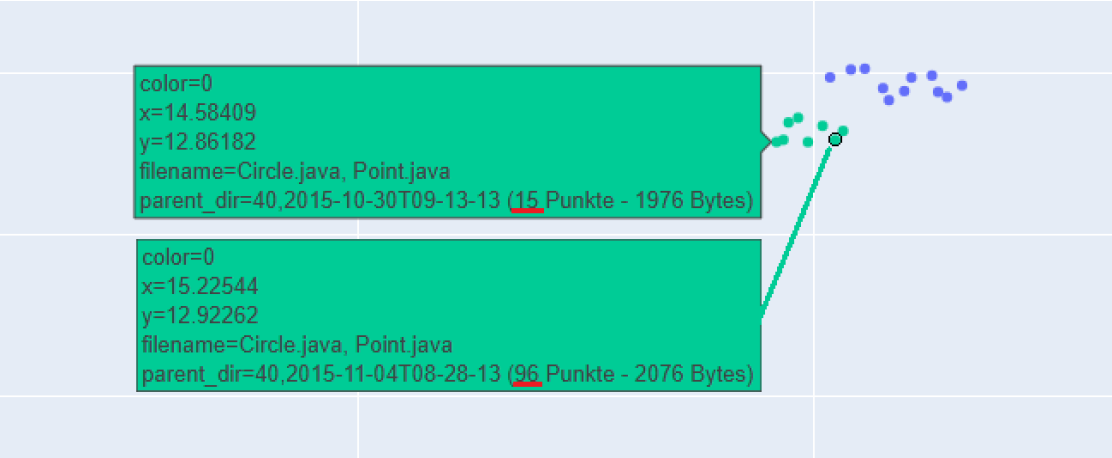

Abbildung 3.3: Interaktives Clustering-Diagramm einer Clusterung von 80 bzw. 40 konkatenierten Java-Dateien. Die rote Unterstreichung zeigt das Problem gemischter Cluster mit unterschiedlichen Punktzahlen.

Punktzahlenintervalle

Um diesem Problem entgegenzuwirken, wurde erneut die Vorgehensweise der wiederholten Prozessschritte mittels einer Schleife angewendet, indem nach Ebenen bzw. Punktzahlenintervallen (Score-Bins) geclustert wird. Es können beliebig viele Intervalle definiert werden (z. B. 0-49, 50-79, 80-89, 90-100), um das Clustering weiter zu präzisieren. Dafür wurde dem `data_loader.py`-Modul eine Methode hinzugefügt, die die Punktzahlen aus den Ordnernamen, in denen die Java-Dateien enthalten sind, ausliest.

Weiterhin wurde das Modul `score_binning.py` entwickelt, die den ausgelesenen Punktzahlen ein Intervall zuordnet. Anschließend werden in der Pipeline für jedes Intervall bzw. für alle Dateien, die zu diesem Bereich gehören, die Prozessschritte ab Embedding wiederholt. Anstatt daraufhin pro Intervall ein Diagramm zu erstellen, wird zur besseren Übersichtlichkeit alle separaten Clusterungen in einem Diagramm visualisiert. Die Ergebnisse der Evaluationsmetriken könnten dadurch leicht verfälscht.

Verarbeitung von Cluster-Ergebnissen

Um die Dateimenge weiter einzugrenzen, wurde ein zusätzlicher Parameter der Konfigurationsdatei hinzugefügt, der zu exkludierende Dateinamen speichert. Zudem wurden die Cluster-Ergebnisse mittels eines `report_generator.py`-Moduls für weiterführende Prozesse vorbereitet, indem sie nach Score-Bins und Cluster sortiert, in eine CSV-Datei abgespeichert werden. Da es pro Studierenden mehrere Einreichungen geben kann, wurde zudem der Überordnername ergänzt. Folgende Auflistung zeigt diese Struktur. Ein Element eines Clusters besteht aus: Überordner - Einreichungsordner - (konkatenierte) Java-Dateien.

Score-Bin: 0-49

- **Cluster 0:**
  - 0D01AAB9 – 2015-11-03T19-55-35 (28 Punkte – 1485 Bytes) – Circle.java, Point.java
  - 0D1C6395 – 2015-10-30T09-10-34 (15 Punkte – 1976 Bytes) – Circle.java, Point.java
- **Cluster 1:**
  - 0B87094E – 2015-11-03T13-16-47 (0 Punkte – 1983 Bytes) – Circle.java, Point.java
  - ...

  ...

Im folgenden Abschnitt wird die finale Implementierung bzw. dessen Module vorgestellt.

3.5 Finale Implementierung

3.5.1 Pipeline

Die Pipeline besteht zu diesem Zeitpunkt aus folgenden Hauptbestandteilen:

Daten laden → Punktzahlenintervalle → Einbetten → Dimensionen reduzieren → Clustern → Evaluieren → Visualisieren → Bericht erstellen

3.5.2 Ablauf in Pipeline

Zuerst wird die Konfigurationsdatei eingelesen (Abschnitt 3.5.3, # load config).

Danach werden für die Funktionalität des Programms unwichtige Warnungen ignoriert, um die Ausgabe übersichtlich zu halten (# ignore warnings). Sie beziehen sich auf veraltete Parameter, eingeschränkte Parallelverarbeitung durch feste zufällige Status (Random States) und automatische Anpassung von Nachbarparametern bei kleinen Datenmengen.

Als nächstes wird der Eingabepfad bestimmt, indem der Pfad der studentischen Einreichungen mit dem des aktuellen Projekts vereint wird (# dynamically determine path).

Dieser Pfad wird daraufhin neben anderen Parametern zum Laden der studentischen Einreichungen genutzt und als Code-Schnipsel abgespeichert (Abschnitt 3.5.4, # load data).

Danach werden die Punktzahlenintervall vorbereitet (Abschnitt 3.5.5, # score bins). Die aus den Einreichungsorder extrahierten Punktzahlen werden einem Punktzahlenintervall eingeordnet und sortiert diese dann alphabetisch.

Nun werden vorbereitend ein Embedding-Objekt und der Name der zwischengespeicherten (cached) Embeddings erstellt (# Embedding object). Zudem werden Listen angelegt, die Informationen über die Einreichungen aus der darauffolgenden Schleife speichert, um sie später in einem Diagramm darstellen zu können (# save all information ...).

Folgend wird über alle Punktzahlenintervalle iteriert (# filter solutions by ...). Darin werden die Indizes aller Einreichungen über eine Liste gemerkt, dessen Punktzahl im aktuellen Punktzahlenintervall liegt. Bei einer niedrigen Dateimenge, kann nicht sinnvoll geclustert werden. Darum werden Punktzahlenintervalle ignoriert, zu denen weniger als vier Einreichungen zugeordnet werden konnten. Danach werden für die ausgewählten Indizes die passenden Code-Schnipsel, Datei- und Ordnernamen aus den jeweiligen Listen herausgefiltert und in neue Listen für diese Punktzahlenintervalle gespeichert.

Im nächsten Teil der Schleife werden die Embeddings berechnet (Abschnitt 3.5.6, # Embedding in loop). Es wird geprüft ob bereits ein Embedding-Cache vorliegt und ob dessen Größe die Anzahl der aktuell zu prüfenden Dateimenge einrahmt. Wenn ja, werden die Embeddings aus dem Cache geladen, ansonsten werden sie neu berechnet und im Cache abgespeichert.

Danach werden nacheinander die Embeddings in ihren Dimensionen reduziert (Abschnitt 3.5.7, # Dimension reduction), die reduzierten Embeddings geclustert (Abschnitt 3.5.8, # Clustering of the ...) und diese dann evaluiert (Abschnitt 3.5.9, # Evaluation). Anschließend werden alle Daten in der vor der Schleife definierten Listen angehängt (# save all information ...).

Nach der Schleife werden die gesammelten Daten in einem Diagramm visualisiert (Abschnitt 3.5.10, # Advanced interactive plot). Daneben kann noch auf das statische Diagramm zugegriffen werden (dient als Reserve und wird hier nicht genauer erklärt).

Als letztes wird der im Implementierungsverlauf erwähnte Bericht über das Clustering erstellt (Abschnitt 3.5.11, # Reporting).

Sämtliche Module inklusive der Pipeline sind im Anhand gegeben.

3.5.3 `config.yaml`

Hier sind alle Parameter gehalten, die in der Pipeline für die verschiedenen Algorithmen gebraucht werden. Es sind die drei vorgestellten Dimensionsreduktionsalgorithmen, sowie die beiden Clustering-Algorithmen mit vollständigen möglichen Parametern angegeben. Weitere Einstellungen werden durch den Parameter "data"getroffen, in den neben Punktzahlen-Intervalle (Score-Bins), die gesuchten Da-

teien, zu exkludierende Dateien, etc. angegeben werden können. Um Algorithmen zu wechseln, müssen sie entsprechend ent- und auskommentiert werden.

3.5.4 `data_loader.py`

Die Klasse nimmt beim Erstellen den Dateinamen, den Pfad zu den studentischen Einreichungen und die zu exkludierenden Datei- und Ordnernamen entgegen. Zudem werden Listen zum Speichern aller Dateinamen und Einreichungsordner gespeichert, um sie später im Diagramm anzeigen zu können. Folgende weitere Methoden sind gegeben:

- load_code_files(): Lädt alle passenden Dateien aus den Einreichungsordnern, liest deren Inhalt, speichert Datei-, Einreichungsordner- und Überordnernamen und entscheidet, ob die Inhalte der Dateien zusammengeführt werden sollen.
- get_scores(): Liest aus den Ordnernamen die Punktzahl heraus und gibt sie als Liste zurück.
- get_filenames() und get_parent_dirs(): Ermöglichen Zugriff auf die zwischengespeicherten Datei-, Einreichungsordner- und Überordnernamen.

3.5.5 `score_binning.py`

Die Klasse enthält die statische Methode bin_scores(), die eine Liste von Punktzahlen aus get_scores() des `data_loader.py`-Moduls nimmt und anhand der definierten score_bins aus der Konfigurationsdatei jeder Punktzahl ein passendes Label (z. B. „90-94“) oder „Unassigned“zuweist, falls sie in kein Intervall passt.

3.5.6 `embedding_model.py`

Die Klasse nimmt beim Erstellen den Namen des Embedding-Modells entgegen und speichert Tokenizer und Transformermodell (hier CodeBERT). Diese erhalten über die Methode get_embedding() Code-Schnipsel, zerlegen sie weiter in Einheiten und wandeln sie in einen numerischen Vektor (Embedding) um. Anschließend wird das Embedding in einem Numpy-Array zurückgegeben.

3.5.7 `dimension_reducer.py`

Ein Objekt der Klasse nimmt den Namen des Algorithmus und ein Dictionary mit Parametern entgegen, welche den Algorithmus manipulieren. Die Methode reduce() nimmt Embeddings entgegen, wendet dann den gewählten Algorithmus darauf an und gibt dimensionsreduzierte Embeddings zurück.

3.5.8 `clustering_engine.py`

Hier werden die dimensionsreduzierten Embeddings aus der Klasse des Moduls `dimension_reducer.py` geclustert und deren Cluster-Zugehörigkeit als Labels zurückgegeben. Der Aufbau ist zudem weitestgehend gleich (siehe Abschnitt 3.5.7).

3.5.9 `evaluation_metrics.py`

Die Klasse enthält die statische Methode evaluate(), welche die dimensionsreduzierten Embeddings und die Clustering-Labels entgegennimmt. Wenn mehr als ein Label bzw. Cluster vorhanden ist, werden die importieren Evaluationsmetriken darauf angewendet, dessen Ergebnisse in einem Dictionary gespeichert und anschließend zurückgegeben.

3.5.10 `advanced_iteractive_plot.py`

Die Klasse enthält eine statische Methode plot(), die die dimensionsreduzierten Embeddings, Clustering-Labels, Datei- und Ordnernamen und Punktzahlenintervalle entgegennimmt. Danach werden diese Daten in eine Tabellenstruktur (DataFrame) umgewandelt und damit dann ein Streudiagramm erstellt. Durch entsprechendes Ent- und Auskommentieren kann zwischen zwei- und dreidimensionalen Diagrammen gewechselt werden.

3.5.11 `report_generator.py`

Die Klasse enthält eine statische Methode generate_report(), welche die Datei- und Ordnernamen, Clustering-Labels, Punktzahlenintervalle und den Ausgabepfad entgegennimmt. Über eine Schleife wird für jedes Punktzahlenintervall ein neues Dictionary und darin für jeden Cluster eine Liste erstellt. Danach werden diese Container erneut durchgegangen, um damit eine CSV-Datei nacheinander zu füllen. Anschließend wird die Datei im Ausgabepfad abgespeichert.

4

Forschungsergebnisse

4.1 Ergebnisse - Zweiter Implementierungsabschnitt

4.1.1 Rangliste der Evaluationsmetriken

In den folgenden beiden Tabellen 4.1 und 4.2 sind die Ergebnisse der Evaluationsmetriken für den zweiten Implementierungsabschnitt enthalten, welche nach Rang bzw. aufsteigend nach bester Algorithmus-Kombination geordnet sind. Der Rang ergibt sich dabei aus dem Mittelwert der normalisierten Werte der Metriken (Davies-Bouldin Index invertiert normalisiert). Die optimalen Werte für die verschiedenen Verfahren sind wie folgt:

- Silhouette Score: 0,5 oder höher
- Caliński-Harabasz Index: höchster Wert aus allen Tests
- Davies-Bouldin Index: niedrigster Wert aus allen Tests (gut zwischen 0,3 und 0,7)

Tabelle 4.1 legt nahe, dass UMAP das geeignetste Dimensionsreduktionsverfahren ist, unabhängig vom Clustering-Algorithmus, wobei t-SNE und PCA mittelwertige Ergebnisse mit k-Means und schlechtere Ergebnisse mit HDBSCAN liefern. Tabelle 4.2 zeigt, dass sich die Eignung nicht großartig ändert. So sind sowohl für kleine, als auch große Dateimengen beide Clustering-Algorithmen zusammen mit UMAP geeignet.

Kombination	Silhouette	Calinski-Harabasz	Davies-Bouldin	Rang
umap_kmeans	0.763	3593.558	0.317	1
umap_hdbscan	0.728	3248.739	0.425	2
tsne_kmeans	0.640	173.360	0.273	3
pca_kmeans	0.609	168.268	0.444	4
pca_hdbscan	0.573	85.872	0.777	5
tsne_hdbscan	0.600	2.318	5.856	6

Tabelle 4.1: Zweiter Implementierungsabschnitt - Evaluationsergebnisse mit 40 Dateien.

Kombination	Silhouette	Calinski-Harabasz	Davies-Bouldin	Rang
umap_kmeans	0.595	6554.361	0.390	1
umap_hdbscan	0.671	1398.606	0.608	2
tsne_kmeans	0.579	490.225	0.609	3
pca_kmeans	0.716	36.378	0.863	4
pca_hdbscan	0.408	201.754	0.780	5
tsne_hdbscan	0.421	267.712	0.814	6

Tabelle 4.2: Zweiter Implementierungsabschnitt - Evaluationsergebnisse mit 320 Dateien.

4.1.2 Clustern unterschiedlicher Dateien in einem Diagramm

Folgende Abbildungen zeigen Clusterungen mit steigender Anzahl Dateien, die sich auf den zweiten Implementierungsabschnitt beziehen. Dabei trat das Problem auf, dass bei größeren Dateimengen die Wahrscheinlichkeit für gemischte Cluster, also mit unterschiedlichen Dateien in einem Cluster anstieg. In den Abbildungen 4.1 und 4.2 ist die Clusterung von 160 Point.java- (orangene Punkte) und 160 Circle.java-Dateien (blaue Punkte) und das beschriebene Problem zu sehen. Die genutzte Algorithmen-Kombination war UMAP mit HDBSCAN. Andere Kombinationen wie PCA mit k-Means ergaben deutlich andere Ergebnisse, jedoch stieg hier die Anzahl der gemischten Cluster ebenso an.

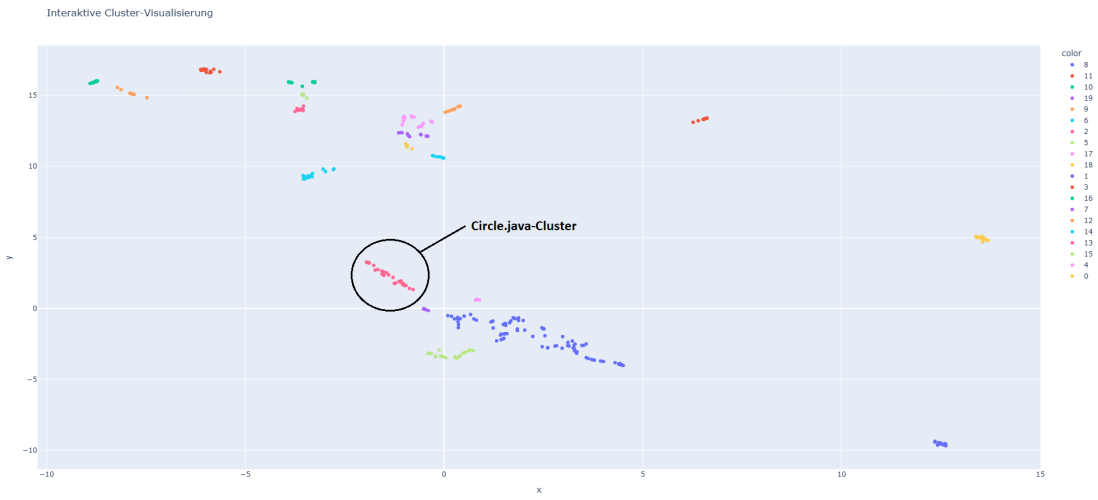

Abbildung 4.1: Zweiter Implementierungsabschnitt - Clustering-Diagramm einer Clusterung von 320 Java-Dateien. Der eingekreiste Cluster ist ein Circle.java-Cluster.

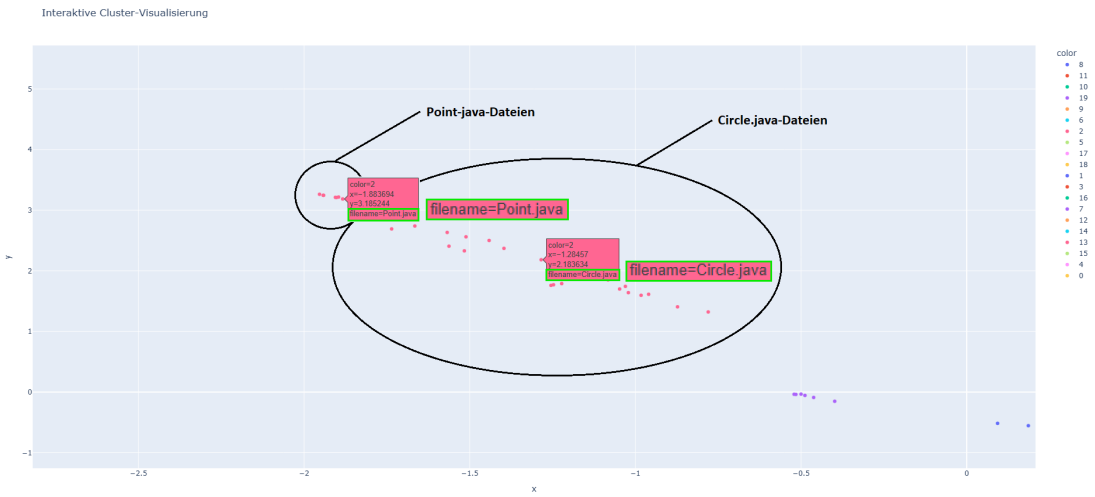

Abbildung 4.2: Zweiter Implementierungsabschnitt - Vergrößerter Circle.java-Cluster aus Abbildung 4.1

4.2 Ergebnisse - Finale Implementierung

Anhand der finalen Implementierung kann nun ein realistisches Szenario getestet werden. Die Abbildungen 4.3 und 4.4 beschreibt eine beispielhafte Aufgabenstellung:

Die zu bearbeitenden Java-Dateien waren Miniprojekt1, Point und Circle. Miniprojekt1-Dateien wurden ignoriert, da sie nicht bearbeitet werden sollten. Gefordert waren die Implementierung verschiedener Methoden der Point und Circle Klassen. Zunächst wurden die Einreichungen anhand der Experimentierungspipeline auf die beste Algorithmen-Kombination anhand Punktzahlenintervalle getestet. Sie wurden an das Notensystem angelehnt definiert (Note 1.0 bis 1.3 $\hat{=}$ Punktzahl $\geq$ 90, Note 1.7 bis 2.3 $\hat{=}$ Punktzahl $\geq$ 75, etc.), wobei perfekte Ergebnisse (mit 100 Punkten bewertet) extra geclustert werden. Wenn die Dateimengen oder die Punktzahlenintervalle klein gewählt sind, treten dabei vermehrt Probleme der Algorithmen auf, die mit geringen Datenmengen nicht funktionieren. Deshalb wurde eine Menge von rund 1000 studentischen Einreichungen/Lösungen zum Testen verwendet (entspricht etwa 2000 einzelnen bzw. 1000 konkatenierten Dateien). Tabelle 4.3 zeigt die Ergebnisse der Evaluationsmetriken ohne und Tabelle 4.4 mit Punktzahlenintervalle. Trotz der Verfälschung in der zweiten Tabelle durch die Punktzahlenintervalle, stellte sich die Kombination UMAP mit k-Means als am besten heraus.

Als nächstes wurde die Pipeline verwendet, um die Clusterungen zu visualisieren. Abbildung 4.5 zeigt das entstandene Diagramm, wobei die unterschiedlichen Farben und Überlappung der Cluster durch die Punktzahlenintervalle zustande kamen und damit visuell weniger wertvoll sind. Aus diesem Grund wurde auf die erstellte CSV-Datei aus dem cluster_report zurückgegriffen. Nun galt es zu überprüfen ob für die Cluster jeweils ein identisches Feedback ausreicht. Dafür wur-

paluno - The Ruhr Institute for Software Technology

Prof. Dr. Michael Goedicke

Miniprojekt 1

Abgabefrist: Dienstag, 3. November 2015, 23:59 Uhr

In diesem Miniprojekt befinden sich zwei Klassen, `Point` und `Circle`, die von Ihnen bearbeitet werden sollen. Die Ausgaben der Klasse `Miniprojekt1` können Sie verwenden, um Ihre Lösung zu testen.

Die Klassen-, Variablen- und Methodennamen dürfen nicht verändert werden!

Bitte beachten Sie, dass für die jeweilige Testat- sowie die Klausurzulassung die Abgabe einer (nicht notwendigerweise korrekten) Lösung zu jedem Miniprojekt erforderlich ist!

Aufgabe 1: Klasse Point

Die Klasse `Point` definiert einen Punkt im zweidimensionalen Koordinatensystem über einen x- und einen y-Wert.

- Der Standardkonstruktor `Point()` muss nicht verändert werden
- Implementieren Sie den Konstruktor `Point(double initX, double initY)` so, dass dem x- und y-Wert die als Parameter übergebenen Werte zugewiesen werden
- Implementieren Sie die Methoden `getX()`, `setX(double newX)`, `getY()` und `setY(double newY)`, welche den Wert der Variablen `Point.x` und `Point.y` setzen bzw. zurückgeben
- Implementieren Sie die Methode `getDistance(Point point)`. Diese Methode erhält ein `Point`-Objekt `point` als Parameter und gibt den Abstand des aktuellen Objekts zu genau diesem Punkt zurück. Der Abstand zwischen zwei Punkten P1(x1 | y1) und P2(x2 | y2) berechnet sich wie folgt:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Hinweis: Die Methode `Math.sqrt(double a)` aus der Java-Standardbibliothek gibt die Quadratwurzel für einen Wert `a` zurück.

Aufgabe 2: Klasse Circle

Die Klasse `Circle` definiert einen Kreis über einen Mittelpunkt im zweidimensionalen Koordinatensystem sowie einen Radius.

- Der Standardkonstruktor `Circle()` muss nicht verändert werden
- Implementieren Sie den Konstruktor `Circle(Point initLocation, double initRadius)` so, dass den Variablen `location` und `radius` die als Parameter übergebenen Werte zugewiesen werden
- Implementieren Sie die Methoden `getRadius()`, `setRadius(double newRadius)`, `getLocation()` und `setLocation(Point newLocation)`
- Implementieren Sie die Methode `getDiameter()` so, dass sie den Durchmesser des Kreises zurückgibt. Der Durchmesser entspricht dem doppelten Radius.

Abbildung 4.3: Programmieraufgabe aus dem Jahr 2015 des paluno - The Ruhr Institute for Software Technology von Prof. Dr. Michael Goedicke, erste Seite

paluno - The Ruhr Institute for Software Technology

Prof. Dr. Michael Goedicke

- Implementieren Sie die Methode `getCircumfence()` so, dass sie den Kreisumfang zurückgibt. Der Umfang eines Kreises berechnet sich wie folgt:

$$U = 2\pi r = \pi d$$

- Implementieren Sie die Methode `getArea()` so, dass sie den Flächeninhalt des Kreises zurückgibt. Der Flächeninhalt berechnet sich wie folgt:

$$A = \pi r^2$$

- Implementieren Sie die Methode `containsPoint(Point point)` so, dass sie `true` zurückgibt, falls `point` innerhalb des Kreises liegt und `false`, falls dies nicht der Fall ist
- Implementieren Sie die Methode `fromPoints(Point center, Point p)` so, dass sie ein neues `Circle`-Objekt mit folgenden Eigenschaften zurückgibt:
  - Der Mittelpunkt des so erzeugten Kreises ist `center`
  - Der Punkt `p` liegt genau auf dem Kreisrand

Abbildung 4.4: Programmieraufagbe aus dem Jahr 2015 des paluno - The Ruhr Institute for Software Technology von Prof. Dr. Michael Goedicke, zweite Seite

Kombination	Silhouette	Calinski-Harabasz	Davies-Bouldin	Rang
umap_kmeans	0.479	2417.266	0.591	1
pca_kmeans	0.530	1872.106	0.495	2
pca_hdbscan	0.702	239.336	0.343	3
tsne_kmeans	0.373	1224.132	0.924	4
umap_hdbscan	0.110	845.411	0.473	5
tsne_hdbscan	0.530	40.382	1.750	6

Tabelle 4.3: Finale Implementierung - Evaluationsergebnisse mit rund 1000 Dateien (ohne Punktzahlenintervalle).

Kombination	Silhouette	Calinski-Harabasz	Davies-Bouldin	Rang
umap_hdbscan	-0.124	100.007	4.872	1
umap_kmeans	-0.017	25.222	6.107	2
pca_kmeans	-0.006	14.737	6.880	3
pca_hdbscan	0.067	17.322	9.449	4
tsne_kmeans	-0.033	11.893	9.075	5
tsne_hdbscan	-0.494	2.994	20.513	6

Tabelle 4.4: Finale Implementierung - Evaluationsergebnisse mit rund 1000 Dateien (mit Punktzahlenintervalle).

den aus jedem Cluster zwei Kandidaten ausgesucht und nach Fehlerarten gesucht, welche das jeweilige Feedback bestimmen würden. Die Tabellen 4.5, 4.6 und 4.7 zeigen die Ergebnisse. Die ersten beiden Tabellen stehen dabei für die Auswertung der Circle- und Point-Dateien und die letzte für die verwendeten Abkürzungen. Sie sind sortiert nach Punktzahlenintervalle (Sc.-B.), Cluster, Aufgabenart (C1-C10, P1-P6), Einreichungen (aus Platzgründen zu Punktzahl und "Päbgekürzt, z. B. "53 P") und ob die Antworten gleich oder ungleich waren. Die Ergebnisse zeigen, dass die Zahl der gleichen Antworten nach steigenden Punktzahlenintervallen entsprechend mitwächst. Identisches Feedback ist erst bei einer 100-prozentigen Übereinstimmung der Antworten sinnvoll. Hier wäre es also nur im Bereich 60-74 und 90-99 angebracht identisches Feedback zu generieren. Um mehr gewährleisten zu können, würde womöglich eine weitere Spezialisierung der Punktzahlenintervalle als Idee nahe erscheinen, jedoch wurden bei einer identischen Punktzahl (53 und 53 in Sc.-B. 50-59) nur ungleiche Antworten gegeben. Es ist also eine Verallgemeinerung der Antwortarten oder des Feedbacks notwendig. Weiterhin könnten Tests mit verschiedenen Algorithmen-Kombinationen für präzisere Ergebnisse sorgen.

Abbildung 4.5: Finale Implementierung - Clusterung von rund 1000 Dateien. Pro Farbe wird weiter durch Punktzahlenintervalle unterschieden

Sc.-B. 0-49	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	gl.	ungl.
Cluster 0:												
0 P	A1	A1	A1	A1	A1	A1	A1	A1	A2	A3		
49 P	A4	A5	A4	A6	A4	A5	A9	A5	A7	A4		
	0	0	0	0	0	0	0	0	0	0	0	10
Sc.-B. 50-59												
Cluster 3:												
53 P	A4	A5	A4	A6	A4	A5	A9	A5	A7	A4		
53 P	A1	A1	A1	A1	A1	A1	A1	A1	A1	A8		
	0	0	0	0	0	0	0	0	0	0	0	10
Sc.-B. 60-74												
Cluster 1:												
66 P	A1	A1	A1	A6	A1	A5	A9	A5	A7	A4		
70 P	A1	A1	A1	A1	A1	A5	A9	A5	A7	A4		
	1	1	1	0	1	1	1	1	1	1	9	1
Sc.-B. 75-90												
Cluster 0:												
75 P	A1	A1	A1	A1	A1	A10	A10	A10	A11	A4		
87 P	A1	A1	A1	A1	A1	A1	A1	A1	A7	A4		
	1	1	1	1	1	0	0	0	0	1	6	4
Sc.-B. 90-99												
Cluster 1:												
92 P	A1	A1	A1	A1	A1	A1	A1	A1	A1	A4		
96 P	A1	A1	A1	A1	A1	A1	A12	A1	A13	A1		

Tabelle 4.5: Finale Implementierung - Auswertung der Circle-Dateien

Sc.-B. 0-49	P1	P2	P3	P4	P5	P6	gl.	ungl.
Cluster 0:								
0 P	A1	A4	A4	A4	A4	A4		
49 P	A15	A1	A1	A1	A1	A1		
	0	0	0	0	0	0	0	6
Sc.-B. 50-59								
Cluster 3:								
53 P	A1	A1	A1	A1	A1	A1		
53 P	A4	A5	A4	A5	A4	A5		
	0	0	0	0	0	0	0	6
Sc.-B. 60-74								
Cluster 1:								
66 P	A1	A1	A1	A1	A1	A14		
70 P	A1	A1	A1	A1	A1	A14		
	1	1	1	1	1	1	6	0
Sc.-B. 75-90								
Cluster 0:								
75 P	A1	A1	A1	A1	A1	A10		
87 P	A1	A1	A1	A1	A1	A1		
	1	1	1	1	1	0	5	1
Sc.-B. 90-99								
Cluster 1:								
92 P	A1	A1	A1	A1	A1	A1		
96 P	A1	A1	A1	A1	A1	A1		
	1	1	1	1	1	1	6	0

Tabelle 4.6: Finale Implementierung - Auswertung der Point-Dateien

Aufgaben:	Abkürzungen:
Circle(Point initLocation, double initRadius)	C1
getRadius()	C2
setRadius(double newRadius)	C3
getLocation()	C4
setLocation(Point newLocation)	C5
getDiameter()	C6
getCircumfence()	C7
getArea()	C8
containsPoint(Point point)	C9
fromPoints(Point center, Point p)	C10
Point(double initX, double initY)	P1
getX()	P2
setX(double newX)	P3
getY()	P4
setY(double newY)	P5
getDistance(Point point)	P6
Antworten:	Abkürzungen:
richtig	A1
falsche Einrückung im else-Fall	A2
Point hat keine Methode getDiameter()	A3
fehlt	A4
return -1.0	A5
return new Point()	A6
return false	A7
center = circle.getLocation();	A8
return Math.PI	A9
return 0.0	A10
Quadrat- statt Kreisüberprüfung	A11
return Math.PI * getCircumference()	A12
if(point.getDistance(location)¿ getRadius())	A13
Es fehlt eine Klammer bei Math.sqrt()	A14
initY = y	A15

Tabelle 4.7: Finale Implementierung - Antwortabkürzungen

5

Fazit und Ausblick

Im Rahmen dieser Arbeit wurde ein Verfahren zur KI-gestützten Clusterung von studentischen Programmierlösungen entwickelt und prototypisch umgesetzt. Ziel war es, Lehrkräfte an Hochschulen bei der Bewertung und Feedbackgenerierung für Programmieraufgaben zu entlasten, indem ähnliche Einreichungen automatisch gruppiert und so der Korrekturaufwand reduziert werden kann. Die entwickelte modulare Python-Pipeline, liest Java-Dateien ein, überführt sie mittels CodeBERT in numerische Vektorrepräsentationen (Embeddings), überträgt diese durch Dimensionsreduktionsalgorithmen in niedrigere Dimensionen und clustert sie. Die Clustering-Ergebnisse werden abschließend visuell aufbereitet und durch Evaluationsmetriken bewertet.

Ein wesentlicher Schwerpunkt dieser Arbeit lag auf der technischen Umsetzung und dem iterativen Ausbau der Pipeline. Dabei wurde deutlich, dass insbesondere bei größeren und ungleichen Datensätzen Herausforderungen bei der Cluster-Konsistenz auftreten können. Durch die Einführung von Punktzahlenintervallen und der Möglichkeit zur Konkatenation von Einreichungen konnte die Clusterqualität verbessert werden.

Erkenntnisse

Die Evaluationsergebnisse zeigen, dass die Kombination aus UMAP als Dimensionsreduktionsverfahren und k-Means als Clustering-Algorithmus in den meisten Testszenarien die besten Ergebnisse lieferte. Dies deckt sich mit Erkenntnissen aus verwandten Arbeiten, in denen UMAP aufgrund seiner stabilen Strukturtreue und Skalierbarkeit für hochdimensionale Daten empfohlen wird (vgl. [MHM]). Gleichzeitig wurde angedeutet, dass experimentell die Clustering-Algorithmen bzw. deren Parameter an die Datenmenge angepasst werden müssen.

Ein weiteres zentrales Ergebnis ist, dass die alleinige Clusterung nach syntaktischen und semantischen Merkmalen nicht ausreicht, um vollständig homogen bewertbare Cluster zu bilden. Daher wurden Clusterung präzisierende Punktzahlenintervalle integriert, die mehr Spielraum und damit eine höhere Relevanz für weiterführendes Feedback ermöglichen sollen.

Ausblick und weiterführende Arbeiten

Die vorliegende Arbeit bildet die Grundlage für eine Reihe weiterführender Forschungs- und Entwicklungsarbeiten. Insbesondere folgende Aspekte bieten sich für eine Vertiefung an:

- **Automatisierte Feedbackgenerierung:** Aufbauend auf den erzeugten Clustern könnten künftig Verfahren zur automatischen Feedbackgenerierung entwickelt werden. Denkbar ist der Einsatz von Large Language Models (LLMs) wie GPT-4 oder Codex, um auf Basis der Cluster ein repräsentatives Feedback-Template zu erstellen und dieses auf die übrigen Cluster-Mitglieder zu übertragen. Erste Ansätze hierzu finden sich bei Tang et al. (2024) (vgl. [TWH+]).
- **Integration dynamischer Metriken:** Die verwendeten internen Evaluationsmetriken bewerten lediglich die Clusterstruktur. In einer praxisnahen Feedback-Pipeline könnten zusätzlich domänenspezifische Metriken wie Codequalität, Laufzeitverhalten oder Einhaltung von Programmierkonventionen berücksichtigt werden, wie es Paiva et al. (2024) mit AsanasCluster vorschlagen (vgl. [PLF24]).
- **Skalierung und Deployment als Web-Service:** Um das entwickelte System hochschulweit nutzbar zu machen, könnte eine Integration in bestehende Lernmanagementsysteme (z. B. Moodle, Stud.IP) über Online-Dienste erfolgen. Die Arbeit von Orvalho et al. (2022) mit InvAASTCluster zeigt, dass eine derartige Veröffentlichung die Akzeptanz und Nutzbarkeit in der Lehre deutlich erhöhen kann (vgl. [OJM]).

Rückblick

Rückblickend wurde ein erheblicher Teil der Arbeitszeit für die iterative Implementierung der Pipeline, Fehlerbehebung sowie das Testen der verschiedenen Algorithmen und Module verwendet, der in Relation zur restlichen Arbeit ungeplant mehr Zeit in Anspruch nahm als erwartet. Weiterhin kann hätte durch breitere Tests und die Integrierung weiterer Algorithmen, noch wesentlich mehr Forschung in diesem Kontext betrieben werden. Trotz dieser Herausforderungen konnte ein funktionales Programm entwickelt werden, die die wesentlichen Anforderungen des Themas erfüllt und als Grundlage für weiterführende Arbeiten dient.

6

Anhang

```
import os
import yaml
import warnings
import numpy as np
from utils.data_loader import DataLoader
from embeddings.embedding_model import EmbeddingModel
from dimReducer.dimension_reducer import DimensionReducer
from clustering.clustering_engine import ClusteringEngine
from utils.score_binning import ScoreBinner
from visualization.advanced_interactive_plot import AdvancedInteractivePlot
from evaluation.evaluation_metrics import EvaluationMetrics
from reporting.report_generator import ReportGenerator

# load config
def load_config(path="config.yaml"):
    with open(path, "r") as file:
        return yaml.safe_load(file)

def run_pipeline():
    config = load_config()

    # ignore warnings
    warnings.filterwarnings("ignore", category=FutureWarning)
    warnings.filterwarnings("ignore", category=UserWarning)

    # dynamically determine path
    base_path = os.path.dirname(os.path.abspath(__file__))
    input_path = os.path.join(base_path, config['data']['input_path'])

    # load data
    loader = DataLoader(config['data']['files_to_look_for'],
                        input_path,
                        config['data']['exclude_files'],
                        config['data']['exclude_folders'])
    code_snippets = loader.load_code_files(concat=True)

    # score bins
    scores = loader.get_scores()
    binned_scores = ScoreBinner.bin_scores(scores,
                                config['data']['score_bins'])
                                unique_bins = sorted(set(binned_scores))

    # Embedding object
    model = EmbeddingModel(config['embedding']['model'])
    cache_path = "cached_embeddings.npy"

    # save all info resulting in the loop to display it in one plot later
    all_embeddings, all_labels, all_filenames,
        all_parent_dirs, all_score_bins = [], [], [], [], []
```

```
51      # filter solutions by score bin and prepare data for clustering
52      for score_bin in unique_bins:
53          print(f"\n==== Processing score_bin: {score_bin} ====")
54          indices =
55              [i for i, b in enumerate(binned_scores) if b == score_bin]
57          if len(indices) < 4:
58              continue    # Not enough solutions in score_bin
60          snippets_bin = [code_snippets[i] for i in indices]
61          filenames_bin = [loader.get_filenames()[i] for i in indices]
62          parent_dirs_bin = [loader.get_parent_dirs()[i] for i in indices]
64          # Embedding
65          if os.path.exists(cache_path):
66              embeddings_cache = np.load(cache_path)
67              if len(embeddings_cache) >= len(snippets_bin):
68                  embeddings = embeddings_cache[:len(snippets_bin)]
69              else:
70                  embeddings = np.array(
71                      [model.get_embedding(code) for code in snippets_bin])
72                  np.save(cache_path, embeddings)
73          else:
74              embeddings = np.array(
75                  [model.get_embedding(code) for code in snippets_bin])
76              np.save(cache_path, embeddings)
78          # Dimension reduction
79          reducer = DimensionReducer(config['dim_reduction']['algorithm'],
80                                     config['dim_reduction']['params'])
81          reduced_embeddings = reducer.reduce(embeddings)
83          # Clustering
84          clusterer = ClusteringEngine(config['clustering']['algorithm'],
85                                       config['clustering']['params'])
86          labels = clusterer.cluster(reduced_embeddings)
88          # Evaluation
89          results = EvaluationMetrics.evaluate(reduced_embeddings, labels)
90          print("Evaluation results:", results)
92          # Save info for interactive plot
93          all_embeddings.extend(reduced_embeddings)
94          all_labels.extend(labels)
95          all_filenames.extend(filenames_bin)
96          all_parent_dirs.extend(parent_dirs_bin)
97          all_score_bins.extend([score_bin] * len(labels))
99      # Interactive plot
100     AdvancedInteractivePlot.plot(all_embeddings,
101                                  all_labels,
102                                  all_filenames,
103                                  all_parent_dirs,
104                                  all_score_bins)
105     # Reporting
106     ReportGenerator.generate_report(all_filenames,
107                                     all_parent_dirs,
108                                     all_labels,
109                                     all_score_bins,
110                                     config['data']['output_path'])
111     print(f"Report saved to {config['data']['output_path']}")
113 if __name__ == "__main__":
114     run_pipeline()
```

Listing 6.1: Pipeline

```yaml
embedding:
  model: "microsoft/codebert-base"  # embedding model for code snippets

dim_reduction:
# --- umap ---
  algorithm: "umap"
  params:
    n_components: 2
    n_neighbors: 15
    min_dist: 0.1
    metric: "euclidean"
    random_state: 42
    spread: 1.0
    learning_rate: 1.0
    init: "spectral"
# # --- pca ---
#   algorithm: "pca"
#   params:
#     n_components: 2
#     svd_solver: "auto"
#     random_state: 42
# # --- t-sne ---
#   algorithm: "tsne"
#   params:
#     n_components: 2
#     perplexity: 30.0
#     learning_rate: 200.0
#     n_iter: 1000
#     metric: "euclidean"
#     random_state: 42
#     init: "random"
#     early_exaggeration: 12.0
#     angle: 0.5
#     verbose: 0

clustering:
# --- hdbscan ---
  # algorithm: "hdbscan"
  # params:
  #   min_cluster_size: 2
  #   min_samples: 1
  #   cluster_selection_epsilon: 0.5
  #   metric: "euclidean"
  #   cluster_selection_algorithm: "eom"
  #   alpha: 1.0
  #   allow_single_cluster: false
# --- k-means ---
  algorithm: "kmeans"
  params:
    n_clusters: 5
    init: "k-means++"
    n_init: 10
    max_iter: 300
    random_state: 42

data:
  files_to_look_for: ".java"
  exclude_files: ["Miniprojekt1.java"]
  exclude_folders: ["100␣Punkte"]
  input_path: "C:/Users/grego/Desktop/Loesungen/000TestingMedium000"
  output_path: "z_Output"
  score_bins:
  [[100, 100], [95, 99], [90, 94], [50, 89], [0, 49]]
```

Listing 6.2: Konfigurationsdatei

```
import os

import os

class DataLoader:
    def __init__(self, file_name, data_path,
            exclude_files=None, exclude_folders=None):
        self.data_path = data_path
        self.file_name = file_name
        self.filenames = []
        self.exclude_files = exclude_files if exclude_files else []
        self.exclude_folders = exclude_folders if exclude_folders else []
        self.parent_dirs = []


    def load_code_files(self, concat=False):
        if concat:
            return self.load_and_concat_code_files()

        code_snippets = []
        for root, dirs, files in os.walk(self.data_path):
            if any(excl in root for excl in self.exclude_folders):
                continue  # skip folder if it matches exclusion
            for file in files:
                if file.endswith(self.file_name)
                and file not in self.exclude_files:
                    file_path = os.path.join(root, file)
                    self.filenames.append(file)
                    punktzahl_ordner = os.path.basename(root)
                    student_id = os.path.basename(os.path.dirname(root))

                    self.parent_dirs.append((student_id, punktzahl_ordner))
                    with open(file_path, "r", encoding="utf-8",
                        errors="ignore") as f:
                        code_snippets.append(f.read())
        return code_snippets


    def load_and_concat_code_files(self):
        concatenated_solutions = []
        for root, dirs, files in os.walk(self.data_path):
            if any(excl in root for excl in self.exclude_folders):
                continue  # skip folder if it matches exclusion
            java_files = [f for f in files if f.endswith(self.file_name)
                and f not in self.exclude_files]
            if java_files:
                solution_code = ""
                for file in java_files:
                    file_path = os.path.join(root, file)
                    with open(file_path, "r", encoding="utf-8",
                        errors="ignore") as f:
                        solution_code += f.read() + "\n"
                concatenated_solutions.append(solution_code)
                self.filenames.append(",␣".join(java_files))

                score_dir = os.path.basename(root)
                student_id = os.path.basename(os.path.dirname(root))

                self.parent_dirs.append((student_id, score_dir))
        return concatenated_solutions


    def get_scores(self):
        scores = []
        for parent_tuple in self.parent_dirs:
            punktzahl_ordner = parent_tuple[1]
            try:
```

```
68                  score = int(''.join(filter(str.isdigit,
69                      punktzahl_ordner.split('␣Punkte')[0].split()[-1])))
70              except ValueError:
71                  score = -1
72              scores.append(score)
73          return scores


76      def get_filenames(self):
77          return self.filenames

79      def get_parent_dirs(self):
80          return self.parent_dirs
```

Listing 6.3: data_loader.py

```
1  class ScoreBinner:
2      @staticmethod
3      def bin_scores(scores, bins):
4          binned_labels = []
5          for score in scores:
6              binned_label = None
7              for low, high in bins:
8                  if low <= score <= high:
9                      binned_label = f"{low}-{high}"  # label string for bin
10                     break
11             if binned_label is None:  # if no bin matched the score
12                 binned_label = "Unassigned"  # assign 'Unassigned' label
13             binned_labels.append(binned_label)
14         return binned_labels
```

Listing 6.4: score_binning.py

```
1  from transformers import AutoTokenizer, AutoModel
2  import torch

4  class EmbeddingModel:
5      def __init__(self, model_name):
6          self.tokenizer = AutoTokenizer.from_pretrained(model_name)
7          self.model = AutoModel.from_pretrained(model_name)

9      def get_embedding(self, code_snippet):
10         tokens = self.tokenizer(code_snippet, return_tensors="pt",
11                                 truncation=True, padding=True)
12         with torch.no_grad():
13             outputs = self.model(**tokens)
14         return outputs.last_hidden_state.mean(dim=1).squeeze().numpy()
```

Listing 6.5: embedding_model.py

```
1  class DimensionReducer:
2      def __init__(self, algorithm="umap", params=None):
3          self.algorithm = algorithm
4          self.params = params if params is not None else {}

6      def reduce(self, embeddings):
7          if self.algorithm == "pca":
8              from sklearn.decomposition import PCA
9              model = PCA(**self.params)
10         elif self.algorithm == "tsne":
11             from sklearn.manifold import TSNE
```

```
            model = TSNE(**self.params)
        elif self.algorithm == "umap":
            import umap
            model = umap.UMAP(**self.params)
        else:
            raise ValueError(f"Unsupported algorithm: {self.algorithm}")

        reduced_embeddings = model.fit_transform(embeddings)
        return reduced_embeddings
```

Listing 6.6: dimension_reducer.py

```
class ClusteringEngine:
    def __init__(self, algorithm="hdbscan", params=None):
        self.algorithm = algorithm
        self.params = params if params is not None else {}

    def cluster(self, reduced_embeddings):
        if self.algorithm == "kmeans":
            from sklearn.cluster import KMeans
            model = KMeans(**self.params)
        elif self.algorithm == "hdbscan":
            import hdbscan
            model = hdbscan.HDBSCAN(**self.params)
        else:
            raise ValueError(f"Unsupported algorithm: {self.algorithm}")

        labels = model.fit_predict(reduced_embeddings)
        return labels
```

Listing 6.7: clustering_engine.py

```
from sklearn.metrics import silhouette_score
from sklearn.metrics import calinski_harabasz_score
from sklearn.metrics import davies_bouldin_score

class EvaluationMetrics:
    @staticmethod
    def evaluate(reduced_embeddings, labels):
        results = {}
        if len(set(labels)) > 1:
            results['silhouette'] = silhouette_score(
                reduced_embeddings, labels)
            results['calinski_harabasz'] = calinski_harabasz_score(
                reduced_embeddings, labels)
            results['davies_bouldin'] = davies_bouldin_score(
                reduced_embeddings, labels)
        else:
            results['silhouette'] = None
            results['calinski_harabasz'] = None
            results['davies_bouldin'] = None
        return results
```

Listing 6.8: evaluation_metrics.py

```
import pandas as pd
import plotly.express as px

class AdvancedInteractivePlot:
    @staticmethod
    def plot(reduced_embeddings, labels,
```

```
7               filenames, parent_dirs, score_bins):
8           df = pd.DataFrame({
9               'filename': filenames,
10              'parent_dir': parent_dirs,
11              'cluster': labels,
12              'score_bin': score_bins,
13              'x': [e[0] for e in reduced_embeddings],
14              'y': [e[1] for e in reduced_embeddings],
15              # 'z': [e[2] for e in reduced_embeddings], # third dimension
16          })

18          # decomment for a 2D diagram
19          fig = px.scatter(df, x='x', y='y',
20                           color=df['cluster'].astype(str),
21                           hover_data=['filename',
22                                       'parent_dir', 'score_bin'],
23                           title="Interaktive Cluster-Visualisierung")

25          # # decomment for a 3D diagram
26          # fig = px.scatter_3d(
27          #     df, x='x', y='y', z='z',
28          #     color=df['cluster'].astype(str),
29          #     hover_data=['filename', 'parent_dir', 'score_bin'],
30          #     title="Interaktive 3D-Cluster-Visualisierung")

32          fig.show()
```

Listing 6.9: advanced_iteractive_plot.py

```
1  import os

3  class ReportGenerator:
4      @staticmethod
5      def generate_report(filenames, parent_dirs,
6                          labels, score_bins, output_path):
7          output_path = os.path.join(output_path, "cluster_report.csv")
8          grouped = {}
9          # group by score_bins and clusters
10         for b, c, p_dir, f in zip(score_bins, labels,
11                                   parent_dirs, filenames):
12             if b not in grouped:
13                 grouped[b] = {}
14             if c not in grouped[b]:
15                 grouped[b][c] = []
16             grouped[b][c].append((p_dir, f))
17         # write score_bins, clusters and file data to csv-file
18         with open(output_path, "w", encoding="utf-8") as f:
19             for b in sorted(grouped.keys()):
20                 f.write(f"Score-Bin: {b}\n")
21                 for c in sorted(grouped[b].keys()):
22                     f.write(f"Cluster {c}:\n")
23                     for (p_dir, subfolder), filename in grouped[b][c]:
24                         f.write(f"- {p_dir} - {subfolder} - {filename}\n")
25                 f.write("\n")
```

Listing 6.10: report_generator.py

Literaturverzeichnis

BLWW21. Bartneck, Christoph, Christoph Lütge, Alan Wagner und Sean Welsh: *What Is AI?* An Introduction to Ethics in Robotics and AI, Seiten 5–16, 2021.

CHTB20. Coombs, Crispin, Donald Hislop, Stanimira K. Taneva und Sarah Barnard: *The strategic impacts of Intelligent Automation for knowledge and service work: An interdisciplinary review.* The Journal of Strategic Information Systems, 29(4):101600, 2020.

CMS. Campello, Ricardo J. G. B., Davoud Moulavi und Joerg Sander: *Density-Based Clustering Based on Hierarchical Density Estimates.* Seiten 160–172.

DB79. Davies, D. L. und D. W. Bouldin: *A Cluster Separation Measure.* IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-1(2):224–227, 1979.

FGT+. Feng, Zhangyin, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang und Ming Zhou: *CodeBERT: A Pre-Trained Model for Programming and Natural Languages.*

HBV01. Halkidi, Maria, Yannis Batistakis und Michalis Vazirgiannis: *On Clustering Validation Techniques.* Journal of Intelligent Information Systems, 17(2-3):107–145, 2001.

JS21. Jerrim, John und Sam Sims: *When is high workload bad for teacher wellbeing? Accounting for the non-linear contribution of specific teaching tasks.* Teaching and Teacher Education, 105:103395, 2021.

Kar01. Karl Pearson: *On lines and planes of closest fit to systems of points in space.* Philosophical Magazine, 2(11):559–572, 1901.

Lau08. Laurens van der Maaten und Geoffrey Hinton: *Visualizing Data using t-SNE.* Journal of Machine Learning Research, 9:2579–2605, 2008.

Mac67. MacQueen, J.: *Some methods for classification and analysis of multivariate observations.* In: Le Cam, Lucien M. und Jerzy Neyman (Herausgeber): *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, Band 5.1, Seiten 281–298. University of California Press, 1967.

MBKS. Messer, Marcus, Neil C. C. Brown, Michael Kölling und Miaojing Shi: *Automated Grading and Feedback Tools for Programming Education: A Systematic Review.*

MHM. McInnes, Leland, John Healy und James Melville: *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction.*

OJM. Orvalho, Pedro, Mikoláš Janota und Vasco Manquinho: *InvAASTCluster: On Applying Invariant-Based Program Clustering to Introductory Programming Assignments.*

PLF24. Paiva, José Carlos, José Paulo Leal und Álvaro Figueira: *Clustering source code from automated assessment of programming assignments.* International Journal of Data Science and Analytics, Seiten 1–12, 2024.

Rou87. Rousseeuw, Peter J.: *Silhouettes: A graphical aid to the interpretation and validation of cluster analysis.* Journal of Computational and Applied Mathematics, 20:53–65, 1987.

TWH+. Tang, Xiaohang, Sam Wong, Marcus Huynh, Zicheng He, Yalong Yang und Yan Chen: *SPHERE: Scaling Personalized Feedback in Programming Classrooms with Structured Review of LLM Outputs.*

You24. Youssef Lahmadi, Mohammed Zakariae El Khattabi, Mounia Rahhali, Lahcen Oughdir: *Optimizing Adaptive Learning: Insights from K-Means Clustering in Intelligent Tutoring Systems.* International Journal of Intelligent Systems and Applications in Engineering, 12(3):1842–1851, 2024.

A

Eigenständigkeitserklärung

☐ Die vorliegende Arbeit wurde als Einzelarbeit angefertigt.

☐ Die vorliegende Arbeit wurde als Gruppenarbeit angefertigt. Mein Anteil an der Gruppenarbeit ist im untenstehenden Abschnitt *Verantwortliche* dokumentiert:

☐ Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne unzulässige Hilfe Dritter angefertigt habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche kenntlich gemacht. Darüber hinaus erkläre ich, dass ich die vorliegende Arbeit in dieser oder ähnlicher Form noch nicht als Prüfungsleistung eingereicht habe.

☐ Es ist keine Nutzung von KI-basierten text- oder inhaltgenerierenden Hilfsmitteln erfolgt.

☐ Die Nutzung von KI-basierten text- oder inhaltgenerierenden Hilfsmitteln wurde von der/dem Prüfenden ausdrücklich gestattet. Die von der/dem Prüfenden mit Ausgabe der Arbeit vorgegebenen Anforderungen zur Dokumentation und Kennzeichnung habe ich erhalten und eingehalten. Sofern gefordert, habe ich in der untenstehenden Tabelle *Nutzung von KI-Tools* die verwendeten KI-basierten text- oder inhaltgenerierenden Hilfsmittel aufgeführt und die Stellen in der Arbeit genannt. Die Richtigkeit übernommener KI-Aussagen und Inhalte habe ich nach bestem Wissen und Gewissen überprüft.

Datum

Unterschrift der Kandidatin/des Kandidaten

Nutzung von KI-Tools

KI-Tool	**Genutzt für**	**Warum?**	**Wann?**	**Mit welcher Eingabe-frage bzw. -aufforderung?**	**An welcher Stelle der Arbeit übernommen?**
ChatGPT	Recherche, Verständnisfragen, Code-Generierung und Textüberarbeitung	Schnellere Quellen-, Quellcode und Fehlersuche	Beim gesamten Verlauf der Arbeit	Suche mir eine Quelle um die Aussage ... zu bestätigen; Erkläre mir im Allgemeinem den Begriff ...; Schreibe mir einen Code ... für die Funktion ...	S. 9 ff.