Vergleichsalgorithmus um ähnliche Objekte zu finden

incognitive | Christoph Pförtner

HTWK Leipzig

Zusammenfassung

In diesem Paper wird der Vergleichsalgorithmus für Objekte aus dem Softwareprojekt kognitiver Speicher erläutert. Weiterhin wird in einem Abschnitt erklärt, wie der implementierte Vergleichsalgorithmus ersetzt werden kann.

I. Grundlegende Funktionsweise des Algorithmus

In der Klasse LevenshteinSimilarityFinder werden Objekte untereinander, sowie Objekte und Kategorien miteinander verglichen. Dies wird mit dem Levenshteinalgorithmus, der durch eine externe Biliothek bereitgestellt wird und String-Vergleichen realisiert. Für jedes eingelesene oder manuell erstellte Objekt wird ein SimilarityScore für jede Kategorie errechnet, das passiert zum einen durch das Vergleichen von dem einzuordnenden Objekt mit der Kategorie selbst, zum Anderen durch das Vergleichen des Objektes mit Objekten, die sich bereits in dieser Kategorie befinden. Nach dem Errechnen des SimilarityScores wird jede Kategorie mit dem zugehörigen Score in einer HashMap gespeichert. Die erstellte HashMap wird anschließend in anderen Programmteilen für die Zuordnung von Objekten zu Kategorien bereitgestellt.

II. Die Methoden im Detail

II.1 compareObjects

In der Methode werden die Matches zunächst mit 0 initialisiert. Mit Hilfe von zwei for-Schleifen werden alle Attribute des neuen Objektes mit allen Attributen eines Objektes verglichen. Dabei werden folgende Bedingungen geprüft:

Stimmen die Attributnamen überein

Enthält der Attributwert des neuen Objekts den Attributnamen des schon bestehenden Objektes

Enhält der Attributwert des bestehenden Objektes den Attributnamen des neuen Objektes

Für jedes Zutreffen dieser Bedingungen wird die Variable Matches um 1 erhöht. Mit Hilfe der bereits erwähnten Bibliothek, werden die Matches noch um den Wert erhöht, der aus der Anwendung der Similarity-Funktion auf die beiden Attributwerte resultiert. Um vergleichbare Werte zu erhalten, werden die Matches noch durch die Anzahl der Attribute des bereits bestehenden Objektes geteilt.

II.2 compareCategory

In dieser Methode wird Matches zunächst mit 0 initialisiert. Innerhalb von zwei for-Schleifen werden alle Attribute des neuen Objektes mit allen Tags einer Kategorie verglichen. Enthält der Attributwert den Tag der Kategorie, werden die Matches um eins erhöht. Weiterhin

wird mit Hilfe der bereits erwähnten Bibliothek die Ähnlichkeit zwischen dem Tag und dem Objektnamen, die Ähnlichkeit zwischen dem Attributnamen und dem Tag, sowie die Ähnlichkeit zwischen den Tags des Objektes mit dem Tag der Kategorie berechnet und zu den Matches addiert. Um vergleichbare Werte zu erhalten, werden die Matches noch durch die Anzahl der Kategorietags geteilt.

II.3 findSimilarObjects

Innerhalb einer for-Schleife werden alle Kategorien durchlaufen und für jede Kategorie, die sich noch nicht in der HashMap befindet, wird für alle Objekte in der Kategorie, die Ähnlichkeit zum einzugliedernden Objekt bestimmt und der HashMap hinzugefügt.

II.4 findSimilarCategories

Innerhalb einer for-Schleifen, in der alle Kategorie durchlaufen werden, wird überprüft, ob der MatchingScore, den compareCategory zurückliefert, über dem MindestScore liegt, der per statischer Variable festgelegt wurde. Ist dies der Fall, wird die Kategorie der HashMap hinzugefügt.

III. Das Interface SimilarityFinder

Für ein einfaches Austauschen der LevenshteinSimilarityFinder-Klasse exisitiert ein Interface SimilarityFinder, das in der neuen Klasse implementiert werden muss. Die erforderlichen Methoden lauten wie folgt:

getFindSimilaritys : Implementation für die Rückgabe des Tasks zum Finden von ähnlichen Kategorien

setFindSimilaritys : Implementation um den Task zu setzen

findSimilarCategories : Implementation zum Finden von ähnlichen Kategorien anhand von Eigenschaften der Kategorien

findSimilarObjects : Implementation zum Finden von ähnlichen Kategorien anhand der Objekte in einer Kategorie

getCategoryWithMatches : Implementation für die Rückgabe der HashMap, die aus den Kategorien und ihrem MatchingScore besteht

IV. Ersetzen des Algorithmus

Um den Algorithmus zu ersetzen, muss zunächst eine neue Klasse erstellt werden, die das Interface SimilarityFinder implementiert. Anschließend ist in der Klasse **ImportViewController** in der Methode parseHtml *(CommonElements-MC/src/main/java/incognitive/commonelements/mc/control/)*, in der Klasse **DataObjectView** in der Methode initPanel *(CommonElements-V/src/main/java/incognitive/commonelements/v/editdialog)* und in der Klasse **TextImport** in der Methode save *(CommonElements-V/src/main/java/incognitive/commonelements/v/)* die Zeile "new LevenshteinSimilarityFinder(dataObject)"durch einen Aufruf der neu erstellten Klasse zu ersetzen.