# Dokumentation Programmentwurf

## Entwurf

Nach einem ersten Test mit der Versionenraummethode stellt sich heraus, dass diese nicht geeignet ist, aus den vorliegenden Trainingsdaten ein Konzept zu erlernen. Das Ergebnis liefert lediglich (nil)(nil). Grund hierfür ist, dass der Algorithmus nicht konvergiert.

Da die Versionenraummethode die Menge der lernbaren Konzepte auf konjunktive Konzeptbeschreibungen beschränkt, ist die Erweiterung des Algorithmus, der sogenannte AQ-Algorithmus zu verwenden. Grundlage der AQ-Erweiterung ist das Stern-Konzept. Hierbei wird die Menge G der Versionenraummethode durch Vorlage eines positiven Trainingsdatensatzes, kombiniert mit allen negativen Beispielen erzeugt. Dies wird für jeden vorliegenden positiven Datensatz durchgeführt und jeweils die beste erzeugte Hypothese aus G, disjunktiv in das Konzept eingefügt.

Vor der Implementierung erfolgt ein Entwurf des Programmablaufs, hier visuell dargestellt durch ein Nassi-Shneiderman-Diagramm.

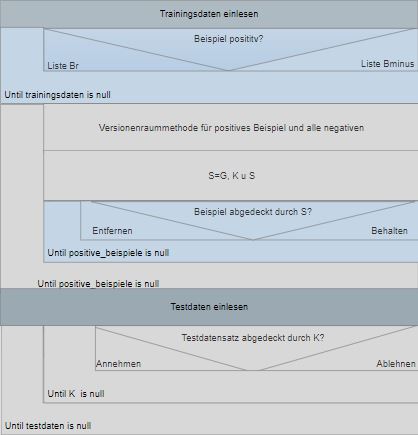


Abbildung 1 - Nassi-Shneiderman Diagramm zum Programmentwurf

## Umsetzung

Grundlage der Implementierung stellt eine durch den Dozenten zur Verfügung gestellte Umsetzung der Versionenraummethode dar. Die bereits enthaltenen Funktionen zum Einlesen von Testdaten aus Dateien werden hierbei wiederverwendet. Vorgegeben ist bereits der Aufruf des Algorithmus in der folgenden Form:

(classify (learn-concept \*path-to-training-data\*) \*path-to-testdata\*)

In unserer Implementierung, stellt die Funktion learn-concept den Startpunkt für die Versionenraummethode dar. Zunächst sind die vorliegenden Trainingsdaten einzulesen und anhand ihres Teacher-Attributes in entweder die Liste Bplus oder Bminus einzufügen. Der AQ Algorithmus sieht nun vor, durch Iteration über die Liste der positiven Beispiele Bplus, für jeden positiven Beipieldatensatz, die bereits vorhandene Implementierung der Versionenraummethode durchzuführen. Sie ist in der Art modifiziert worden, dass sie nicht Daten aus einer Datei einliest, sondern die bereits erstellten Listen Bplus und Bminus verwendet. Die entsprechenden Zeilen sehen nach der Modifikation wie folgt aus:

(defun version-space (examples)

(let ((dataset (length examples)))

Zurückgegeben wird hierbei die Liste G des Ergebnisses und das erste Element als der erlernte Stern für diese Iteration festgelegt. Der Stern ist nun dem Konzept hinzuzufügen. Aus Performance-Gründen sind nun aus den positiven Beispielen alle diejenigen zu entfernen, die durch diesen aktuellen Stern bereits abgedeckt werden. Unsere Funktion check-if-covered ist hierbei dafür zuständig, über jeweils ein positives Beispiel zu iterieren und zu prüfen, ob es durch den aktuellen Stern abgedeckt ist.

(defun check-if-covered (brelement s)

(cond ((OR (NULL brelement) (NULL s)) T)

((EQUAL (car s) \*star\*) (check-if-covered (cdr brelement) (cdr s)))

((EQUAL (car s) (car brelement)) (check-if-covered (cdr brelement) (cdr s)))

(T NIL)

)

)

Ist der positive Datensatz bereits abgedeckt, ist er aus der Liste der positiven Beispiele zu entfernen, damit er nicht erneut verarbeitet wird. In jeder Iteration wird der so erzeugte Stern in das Konzept K eingefügt. Dies resultiert in einer Liste K, die aus Listen der einzelnen Sterne besteht und zur Klassifizierung verwendet werden kann. Der vorgegebene Aufruf lautet classify. Diese Funktion liest zunächst die vorgegebenen Testdaten ein und verwendet anschließend Hilfsfunktionen, um die einzelnen Beispiele darauf zu prüfen, ob sie durch das erlernte Konzept K abgedeckt sind. Hierfür wird durch die einzelnen Attributwerte iteriert. Enthält das Konzept an einer Position einen Stern, ist der entsprechende Attributwert des Datensatzes unwichtig. Enthält das Konzept allerdings einen Wert, muss dieser mit dem Attributwert übereinstimmen. Der Testdatensatz wird akzeptiert, sobald ein Ausdruck im Konzept gefunden wird, der diesen Datensatz abdeckt, da es sich um Disjunktionen handelt.

## Test und Ergebnisbewertung

Ein Durchlauf des fertigen Algorithmus führt zu den folgenden Ergebnissen.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Anzahl der akzeptierten Datensätze |  |  |
| fälschlicherweise akzeptierte Datensätze |  |  |
| korrekt akzeptierte Datensätze |  |  |
| Anzahl der abgelehnten Datensätze |  |  |
| fälschlicherweise abgelehnte Datensätze |  |  |
| korrekt abgelehnte Datensätze |  |  |
| Genauigkeit Allgemein |  |  |
| falsch klassifizierte Datensätze |  |  |
| korrekt klassifizierte Datensätze |  |  |

Die erreichte Genauigkeit von \_% ist für eine zuverlässige Klassifizierung zu ungenau. Des Weiteren ist zu erkennen, dass im Allgemeinen die Klassifizierung negativer Beispiele zuverlässiger funktioniert, als die der positiven. Hier wurde eine korrekte Zuordnung von \_% erzielt, während es bei den positiven nur \_% sind.