# 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.

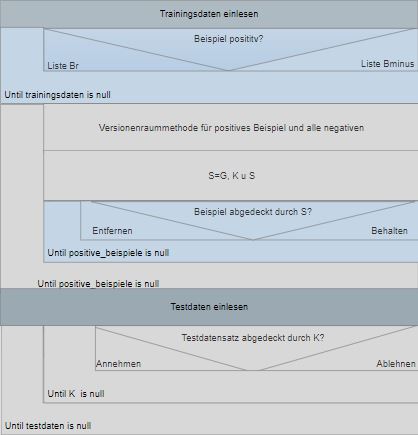


Abbildung 1 - Nassi-Shneiderman Diagramm zum Programmentwurf

## 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\*)

Wird unser Entwurf zunächst kompiliert kann er direkt ausgeführt werden, die Testdaten und Trainingsdaten müssen hierfür im gleichen Verzeichnis liegen. 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. Des Weiteren werden hier bereits leere Listen für das Konzept und den Stern erzeugt. Die zuständige Funktion aq-step ist rekursiver Natur und wird für jedes Element der Liste der positiven Beispiele aufgerufen. Der AQ Algorithmus sieht nun vor, durch Iteration über die Liste der positiven Beispiele Bplus, für jeden positiven Beispieldatensatz, 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 remove-covered-examples ist hierbei dafür zuständig, über jeweils ein positives Beispiel zu iterieren und zu prüfen, ob es durch den aktuellen Stern abgedeckt ist.

(defun remove-covered-examples (br s br-without-s)

(cond ((null br) br-without-s)

((is-included (car br) (list s)) (remove-covered-examples (cdr br) s br-without-s))

(T (remove-covered-examples (cdr br) s (cons (car br) br-without-s)))))

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.

(defun classify (k filename)

(let ((testdata (get-examplelist (load-exampleset filename))))

(classifier k testdata)

))

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. Die entsprechende Funktion ist zu umfangreich um hier abgebildet zu werden, an dieser Stelle wird auf die Kommentare im Code verwiesen. Die Funktion classifier enthält zusätzlich die Funktionalität direkt auszugeben, ob ein Beispieldatensatz korrekt oder fälschlicherweise positiv oder korrekt oder fälschlicherweise negativ eingestuft wurde. Dies erleichtert die anschließende Auswertung.

## Test und Ergebnisbewertung

Ein Durchlauf des fertigen Algorithmus führt zu den folgenden Ergebnissen.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Anzahl der akzeptierten Datensätze | 57 | 5.69 % |
| fälschlicherweise akzeptierte Datensätze | 49 | 85.96 % |
| korrekt akzeptierte Datensätze | 8 | 14.03 % |
| Anzahl der abgelehnten Datensätze | 944 | 94.3 % |
| fälschlicherweise abgelehnte Datensätze | 25 | 2.65 % |
| korrekt abgelehnte Datensätze | 919 | 97.35 % |
| Genauigkeit Allgemein |  |  |
| falsch klassifizierte Datensätze | 74 | 7.39 % |
| korrekt klassifizierte Datensätze | 927 | 92.61 % |

Die erreichte Genauigkeit von 92.6 % klingt für eine zuverlässige Klassifizierung eigentlich ausreichend. Allerdings ist zu beachten, dass die Genauigkeit bei der Klassifizierung der negativen Beispiele weitaus höher ist als bei den positiven Testdaten. Dies liegt daran, dass sehr viel mehr negative Trainingsdaten vorlagen als positive. Somit ist das gelernte Konzept genauer an die Klassifizierung negativer Daten angepasst. Damit dieser Algorithmus allgemein angewendet werden kann, ist es notwendig ihn mit einer größeren Zahl an positiven Daten zu trainieren, um auch hier die Genauigkeit zu erhöhen.