Theoriefragen zur Implementationsaufgabe

Küpper Joel, Ockenfels Malou, Schulz Daniel

January 23, 2017

1 5.3) Normalization and Tie-Breaking

1.1 a) Entscheidungsfunktion für getWinner

Problem: *Es kann vorkommen, dass mehr als eine Klasse am meisten Stimmen von den Nachbarn bekommen hat. Welche Klasse soll dann ausgewählt werden?*
Die Idee ist, dass falls mehr als eine Klasse am meisten Stimmen bekommen hat, jene klasse ausgewählt wird, die generell im Data-set am Meisten vorkommt. Somit ist die Wahrscheinlichkeit, richtig zu liegen, am höchsten. Den Fall, dass die beiden *gewinnenden* Klassen auch gleichzeitig gleich häufig im Data-set vorkommen wird hier in der Implementierung nicht beachtet, da die Wahrscheinlichkeit doch sehr gering ist, vorallem bei den vorliegenden Data-sets.
Für diesen Fall könnte jedoch so vorgegangen werden, dass jene Klasse vorhergesagt wird, die am nächsten (k=1, =2, ...) am Beispiel liegt. Dann gibt es aber Probleme bei zu hohem Rauschen in den Data-sets.
Eine weitere Möglichkeit wäre, dass man k+=1 inkrementiert und so einen weiteren Nachbarn (oder zwei, drei, ...) hinzuzieht, um eine eindeutigere Mehrheit zu finden. Dabei tritt aber das Problem der zu generrellen Vorhersage auf. Da in der Aufgabe keine Angabe über jene Priorisierung gemacht wurde, werden die Denkansätze hier nur erähnt und nicht implementiert.
Der folgende Pseudocode illustriert, wie das Zählen umgesetzt wurde.

Algorithm 1 getWinner Entscheidungsfunktion

```
function GETWINNER(data, votes)
    max ← votes.getHighestVote()
    classesWithMaxVotes ← null
    for each vote ∈ votes do
        if vote.numberOfVotes() == max then
            classesWithMaxVotes.add(vote);
    if classesWithMaxVotes.size == 1 then
        return ← classesWithMaxVotes.first();
    else
        counter.init(0);
        for each instance ∈ data do
            if classesWithMaxVotes.contains(instance.getClass()) then
                counter.incrementClass(instance.getClass());
    return ← counter.getBiggest().getClass();
```

1.2 c) Gleicher Abstand mehrerer Instanzen

Problem: *Es kann vorkommen, dass mehrere Instanzen den gleichen Abstand vom Beispiel haben. Wie soll dabei vorgegangen werden?*
Das ist prinzipiell nur ein Problem, wenn der k'te Nachbar den gleichen Abstand hat wie k+1, k+2, ... Wenn die Nachbarn eins und zwei bspw. bei k=4 den gleichen Abstand haben, gibt es kein Problem.
Erst wenn Nachbar k und Nachbar k+1 den gleichen Abstand und unterschiedliche Klassen haben, gibt es Entscheidungsbedarf. Nun, hier ist die Idee, wie oben schon beschrieben, aber nun unausweichlich, k zu inkrementieren und alle Nachbarn, die den gleichen Abstand haben wie der k'te Nachbar auch noch als nearst Neighbor zu bezeichnen und entsprechend zu berücksichtigen.
Auch hier der Pseudocode, wie es umgesetzt wurde:

Algorithm 2 getNearest Entscheidungsfunktion

```
function getNearest(sortedListOfNeighbors, k)
    sortedNeighborsByDistance ← null;
    for each neighbor ∈ sortedListOfNeighbors do
        sortedNeighborsByDistance.addAt(neighbor.getDistance(), neighbor);
    nearests ← null;
    for each setOfNeighborsWithSameDistance ∈ sortedNeighborsByDistance do
        nearests.addAll(setOfNeighborsWithSameDistance);
        if nearest.size() >= k then
            break;
    return ← nearests;
```