Report
EFME UE Exercise 1 - WS 2013

Gruppe B9

Florian Groh, 1168186
Felix Ledochowski, 1028318
Daniel Witurna, 1125818

1. Einleitung

Das Ziel dieser Übung war unseren ersten, eigenen Pattern Recognition Algorithmus zu implementieren. Zu diesem Zwecke mussten wir mehrere Stufen durchlaufen: Bilder einlesen, Features kalkulieren und mit eigenen Berechnungen erweitern, geeignete Merkmale bestimmen, trivialen Algorithmus (per Hand) und K-NN Algorithmus implementieren.

2. Umsetzung

2.1 Bilder einlesen

Um die Bilder und deren Eigenschaften (Solidity, Area, ...) aus der MPEG7 CE-Shape-1 database einzulesen, haben wir die Anleitung aus der gegebenen Angabe [1] unter Punkt 3.1.1 "How to proceed" verwendet. Als Bildmaterial haben wir nach dem Trial-and-Error Verfahren die Klassen "Watch", "Brick", "Fork, "Fountain" sowie "Apple" ausgewählt.

2.2 Features kalkulieren und mit eigenen Berechnungen erweitern

Zu allererst berechnen wir in getProps formfactor, roundness, compactness und aspectratio und fügen diese neuen Skalierungs-, Rotations- und Positionsinvarianten Eigenschaften zu den Ergebnissen, die wir per regionprops erhalten haben hinzu [Codeblock 1]. Die Formeln dafür haben wir den Vorlesungsfolien [2] entnommen.

```
props.formfactor  = 4*pi*props.Area/(props.Perimeter)^2;
props.roundness   = 4*props.Area/(pi*props.MajorAxisLength^2);
props.compactness = sqrt(props.roundness);
props.aspectratio = props.MinorAxisLength/props.MajorAxisLength;
```

Codeblock 1

2.3 Geeignete Merkmale bestimmen

Im nächsten Schritt haben wir unsere Berechnungen verwendet, um per Scatterplot zu erkennen, welche Eigenschaften sich am besten zur Unterscheidung verschiedener Datensätze eignen.

Generell konnten wir folgende Merkmale schon im Vorhinein ausschließen:

- Perimeter (nicht Skalierungsinvariant)
- Fläche (nicht Skalierungsinvariant)
- Centroid (nicht Positionsinvariant)
- Area (nicht Skalierungsinvariant)
- FilledArea (nicht Skalierungsinvariant)
- Orientation (nicht Rotationsinvariant)

Um einen Überblick zu erhalten, haben wir alle relevanten Merkmale in deren Kombinationen als Diagramm zeichnen lassen [Figure 1].

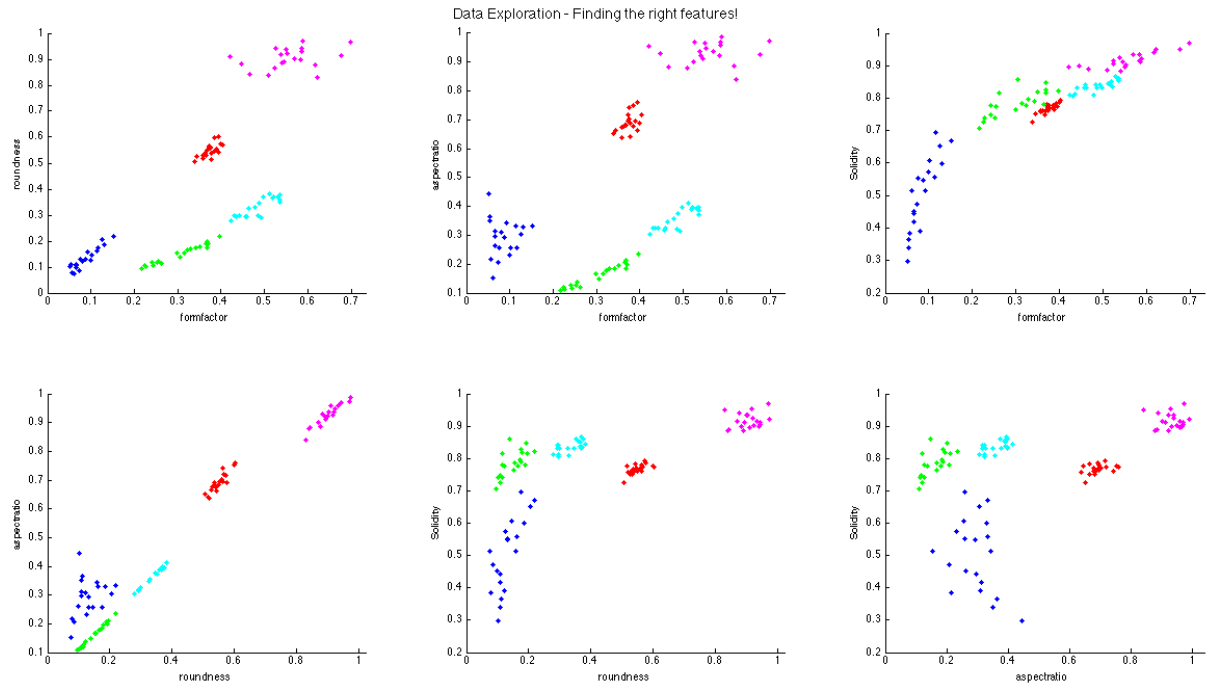

Figure 1

Dabei fiel uns auf, dass sich roundness und formfactor am besten eigneten um unsere ausgewählten Daten ("Watch", "Brick", "Fork", "Fountain" und "Apple") zu unterscheiden [Figure 1.1] Roundness sowie Formfactor sind Positions-, Rotations- sowie Skalierungsinvariant und daher für unsere Zwecke gut geeignet.

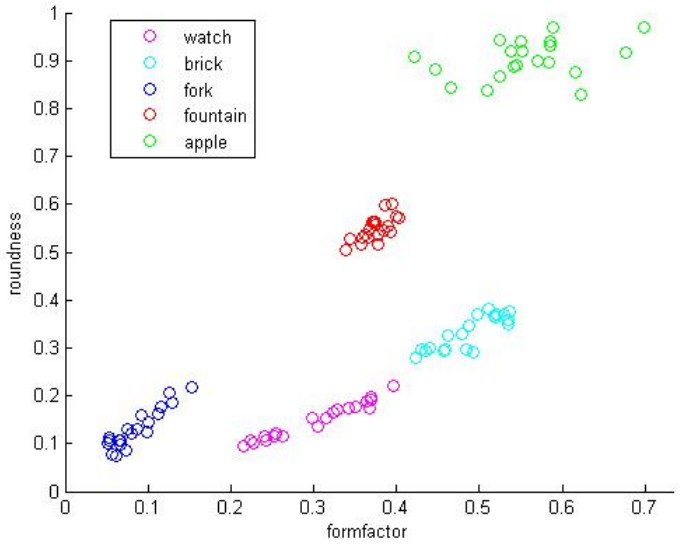

Figure 1.1

2.4 Klassifizierung per Hand Algorithmus

Der nächste Schritt war es einen einfachen Klassifizierungs Algorithmus zu schreiben, der sich auf per Hand ausgesuchte Werte verlässt. Diese Werte erhielten wir durch die Analyse des Scatterplots in Figure 1.1.

```
% Classify
if and(prop1(j) > 0.4, prop2(j) > 0.8)
    classified{j} = 'Apple';
elseif and(prop1(j) > 0.3, prop2(j) > 0.45)
    classified{j} = 'Fountain';
elseif and(prop1(j) > 0.4, prop2(j) > 0.25)
    classified{j} = 'Brick';
elseif and(prop1(j) > 0.2, prop2(j) > 0.05)
    classified{j} = 'Watch';
```

```
else
    classified{j} = 'Fork';
end
```

Codeblock 2

Das Ergebnis dieser Klassifizierung ist in Figure 2 mittig zu sehen. Wie wir bei der Evaluierung noch genau sehen werden, arbeitet dieser Algorithmus auf unseren Daten fehlerlos.

2.5 Klassifizierung per K-NN Algorithmus

Unsere letzte Aufgabe war es einen K-NN Algorithmus zu implementieren [Codeblock 3]. Wir haben uns an die Angabe gehalten und Leave-one-out cross-validation (LOOCV) für die Testdaten verwendet. Dies haben wir durch ein simples LOOCV flag erledigt, welches angibt ob die kürzeste Distanz ignoriert (=0) werden soll.

Der K-NN Algorithmus selbst, ist auf Grund diverser Vektorisierungen nicht mehr so einfach lesbar wie in einer ursprünglichen Variante die auf for-Schleifen basierte, jedoch ist er um ein vielfaches schneller geworden.

```
%Pairwise euclidean distance between SAMPLE and TRAIN
%dist(i,j) is the distance between SAMPLE(i) and TRAIN(j)
dist = pdist2(SAMPLE,TRAIN);

%Ascending sort of each row of the dist-Matrix
%only indices important because we need to keep original information
[~, minIndexMatrix] = sort(dist, 2);

%if LOOCV, skip first column which has zero distance
kClasses = TRAINCLASSES(minIndexMatrix(:, 1+LOOCV : K+LOOCV));

%uniqueValues is selfexplanatory, indexMap is basically a numerical
%representation of kClasses.
[uniqueValues, ~, indexMap] = unique(kClasses);
indexMap = reshape(indexMap, testSize, K);
```

```
%Find most frequent class
[Modal, ~, Tie] = mode(indexMap,2);



%Actual classification
for i = 1 : testSize
    if size(Tie{i,1},1) > 1
        %more than one modal value, break tie by choosing the nearest
        %of possible neighbor values
        for j = 1 : K
            if ismember(indexMap(i,j),Tie{i,1})
                Modal(i) = indexMap(i,j);
                break;
            end
        end
    end
    SAMPLECLASSES(i) = uniqueValues(Modal(i));
end
```

Codeblock 3

3. Evaluierung

Um unsere Umsetzung auf ihre Richtigkeit zu überprüfen, vergleichen wir die MATLAB - K-NN Funktion mit unseren Implementierungen unter Verwendung unterschiedlicher Parameter.

```
%Test a couple of different k's
k = [1,2,5,10,15,25,35,50,70,99];
for i = 1 : numel(k);


    %Matlab doesn't have a LOOCV flag, we need to do it in a for loop
    for j = 1:elements
```

```
        ix = [1:j-1,j+1:elements]; %all indices except j

        matC(j) = knnclassify(TRAIN(j,:),TRAIN(ix,:),TRAINCLASSES(ix),k(i));
    end

    ourC = knn(TRAIN,TRAIN,TRAINCLASSES,k(i),true);

    diffMatlab = nnz(~strcmp(matC,ourC)); %count different classifications
end
```

Codeblock 4

In Figure 2 vergleichen wir das Training Set mit dem Ergebnissen des simplen Algorithmus und unserem K-NN Algorithmus bei aktivem LOOCV flag und einem k gleich 5.

Nicht überraschend sind alle drei Ergebnisse identisch – die Effektivität der Algorithmen unter den gewählten Parametern liegt bei 100%.

Daraus lässt sich schließen, dass die einzelnen Klassen-Cluster im Feature-Raum weit genug von einander entfernt liegen um bei diesem k keine Fehler entstehen zu lassen.

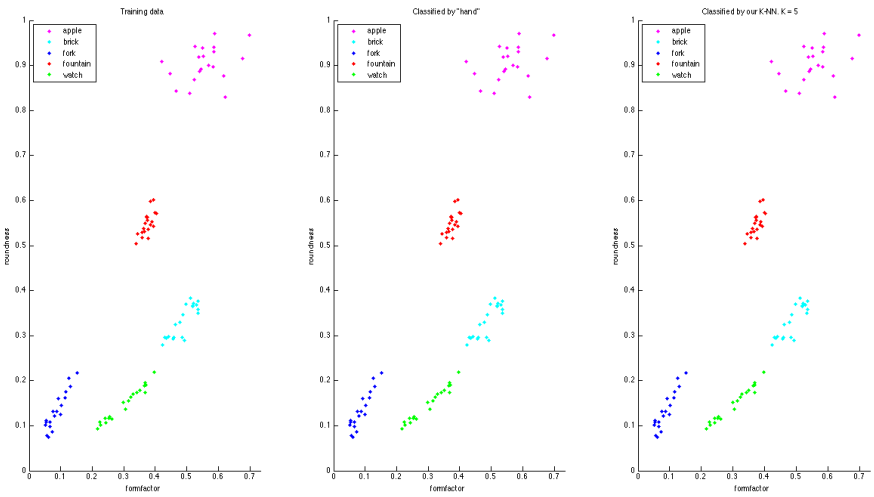

Figure 2

Nachdem wir die Richtigkeit unseres k-NN Algorithmus verifiziert haben, haben wir nun die Effektivität unter verschiedenen Ks getestet [Figure 3]

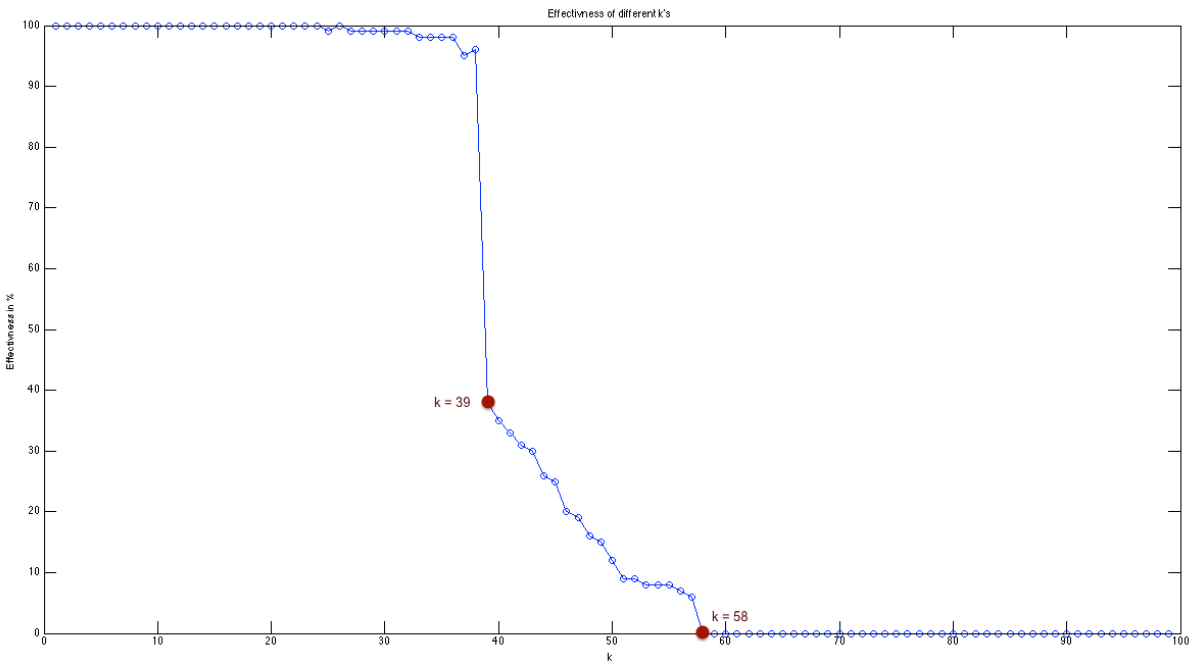

Figure 3

4. Diskussion

Nach Analysieren der Effizienz der verschiedenen Ks im KNN [Figure 3] sind uns ein paar K Werte aufgefallen, die anscheinend bestimmte Eigenschaften haben. Bei 39 fällt die Effizienz stark ab, weniger als 50% der Eingabewerte werden richtig klassifiziert. Dieser massive Sprung in der Effektivität lässt sich folgendermaßen erklären: Da es, wegen dem Leave one out Verfahren, selbst unter optimaler Positionierung nur 19 Trainings-Werte mit der richtigen Klasse geben kann, ist es möglich, dass eine komplette andere Klasse mit 20 Werten teil der 39 nächsten Nachbarn ist. Diese 20 Werte sind dann in der Überzahl und bestimmen die Klassifizierung.

Ein weiterer interessanter K Wert war 58. Ab dieser Anzahl sind die Ergebnisse immer zu 100% falsch. Dies konnten wir nicht so einfach erklären. Da eine Effektivität von 0% erst ab einem k von 5*19+1 (=96) zu 100% garantiert ist schließen wir daraus, dass 58 (= 3*19 +1) ein problemspezifischer Wert sein muss. Intuitiv gesehen ist es aber dennoch verständlich, dass in den 58 nächsten Nachbarn zumindest eine falsche Klasse komplett enthalten ist.

5. Referenzen

[1] EFME 2013 LU Exercise 1, EFME UE LVA WS 2013, TU Wien

[2] Einfuehrung_EFME_II, Yll Haxhimusa, Folie 28, EFME UE LVA WS 2013, TU Wien

6. Paper Summary

Auf den folgenden Seiten.

Zusammenfassung
When Is "Nearest Neighbor" Meaningful? Beyer, Goldstein, Ramakrishnan, Shaft.

Gruppe B9

November 10, 2013

Abstract

Diese Zusammenfassung ist im Zuge der Übung zur Einführung in die Mustererkennung WS 2013 entstanden. Das zusammenzufassende Werk trägt den Titel "When Is 'Nearest Neighbor' Meaningful?" von Beyer, Goldstein, Ramakrishnan, Shaft und wurde an der University of Wisconsin-Madison veröffentlicht.

Gruppe B9
Florian Groh, 1168186
Felix Ledochowski, 1028318
Daniel Witurna, 1125818

1 Zusammenfassung des Papers

Der Artikel behandelt das Problem von hohen Dimensionen bei der Verwendung eines Nearest-Neighbor Algorithmus. Es werden Datensätze vorgestellt, die empirisch zeigen, dass schon ab 10-15 Dimension eine Anwendung von NN-Verfahren kein wertvolles Ergebnis liefern kann. Dies wird insbesondere zum Problem, da immer mehr versucht wird, schwierige und komplexe Daten durch hoch-dimensionale Merkmalsvektoren anzunähern.

Um die Wertigkeit von Datensätzen und damit auch deren Eignung für NN zu bestimmen, führen Beyer et al. die Definition der Instabilität ein. Instabiles Verhalten entsteht, sobald ein Großteil der Nachbarn eines Anfragepunktes kaum weiter entfernt sind als der nächste Nachbar (Nearest Neighbor). Um dies ein wenig formeller auszudrücken führen wir die Begriffe Minimal- und Maximaldistanz ein. Bei der Minimaldistanz (Dmin) handelt es sich um die Entfernung zwischen dem Anfragepunkt und dem nächsten Nachbarn, die Maximaldistanz (Dmax) ist dementsprechend der Nachbar mit der weitesten Entfernung. Ein Datensatz ist genau dann instabil, wenn für einen beliebigen Anfragepunkt die Maximaldistanz um einen kleinen Faktor Epsilon > 0 größer als die Minimaldistanz ist (Dmax = (1+Epsilon) * Dmin).
Von stabilem Verhalten wird dann gesprochen, wenn wenige Punkte in diese erweiterte Umgebung fallen und dadurch die NN-Anfrageumgebung von den anderen Daten sinnvoll getrennt ist. Diese Definition wird verwendet, um zu zeigen, dass in vielen Situationen, in denen die Dimensionalität ansteigt, die Wahrscheinlichkeit der Instabilität einer NN-Anfrage gegen 1 konvergiert. Diese eben erwähnten Situationen treten nur unter Zutreffen der von Bayer et al. aufgestellten Theoreme auf.

Auch auf die Frage welche Datensätze auch bei hoch-dimensionaler Indexierung für den NN-Algorithmus Sinn ergeben gehen die Autoren ein. Beispielsweise macht eine Klassifizierung sehr wohl Sinn, falls Trainings-Werte existieren, die genau der Anfrage entsprechen und dadurch die minimale Distanz 0 wird. Eine andere Möglichkeit bietet sich, wenn Punkte der Anfrage nicht genau übereinstimmen müssen sondern auch in einer kleinen Entfernung eines Datenpunkts erlaubt sind. Je mehr Dimensionen die Datenpunkte allerdings haben, umso wahrscheinlicher wird, dass selbst sorgfältig ausgewählte Trainings-Sets nicht mehr diskriminative Entfernungen haben. Weitere Ausführungen beschäftigen sich mit der Bildung von sogenannten Clustern, in welche die Anfrage fallen muss.

Als Ergebnis dieses Papers werden mehrere Punkte angeführt. Beispielsweise wurde mehrere Szenarien der Anwendung gefunden, in denen die Entfernungen von Punkten und deren nächsten Nachbarn vernachlässigbar klein werden. In diese Anwendungsgebiete fällt auch die übliche und vielgeprüfte Verwendung von NN als Heuristik. Weiters war die Frage in welchen Bereichen die Dimensionalität so hoch ist, dass kein aussagekräftiges Ergebnis mehr gefunden wird.

Die Antwort konnten die Autoren durch Simulationen bestimmen und meinen, dass in den ersten 20 Dimensionen der Abstand der Punkte sehr stark abnimmt und über die 20. Dimension sehr schnell einen Punkt erreicht, ab dem die Entfernung zu gering wird. Der praktische Nutzen dieser Arbeit wird in den Bereichen der Evaluierung eines Nearest Neighbor - Aufgabenbereich sowie einer Nearest Neighbor - Vorgehensweise angegeben.