It is important to notice that the objects are distributed in random places in the image, and that they have random orientations and scale (as shown in the Figure 1). Based on this information, it should already be possible to exclude some features as b eing useless for this application. Which are they?

Useless regionprops:

'Area'
: Number of Pixels in the blob, useless because not scale invariant

'Centroid'
: Center of mass of a blob, useless because not position/rotation invariant

'ConvexImage'
: binary image of the convex hull, not invariant to anything.

'ConvexArea'
: number of pixels in the convex hull, not scale invariant.

'EquivDiameter'
: diameter of a circle that fills the same area as the blob. useless because not scale invariant

'Extrema'
: calculates 8 extrema points, probably useless, does not appear to be invariant to scale or rotation

'FilledArea'
: number of pixels in a FilledImage, useless because not scale invariant

'FilledImage'
: fills in all holes of a binary image blob. useless because…

'Image'
: binary image of the same size as the bounding box, useless because boundingbox is better.

'MajorAxisLength'
: does the same magic as eccentricity but instead of showing the ratio, it shows the length of the major axis, not scale invariant

'MinorAxisLength'
: see above.

'Orientation'
: angle between the major axis and the x axis of the magical eccentric ellipse, probably useless, does not appear to be invariant to rotation

'Perimeter'
: length of an outer boundary, useless because not scale invariant.

'PixelIdxList'
: linear indices for all pixels in the area, useless because not invariant to anything

'PixelList'
: locations of all the pixels in the area, useless because not invariant to anything

'SubarrayIdx'
: likely useless

'MaxIntensity'
    useless for binary images
'MeanIntensity'
    see above
'MinIntensity'
    see above
'PixelValues'
    useless for binary images
'WeightedCentroid'
    see centroid, but it takes intensity values into account. binary image, useless

Useable regionprops:
'BoundingBox'
    Smallest rectangle around the blob, unlikely to be usefull, but useable if the ratio of the height and with is used (always larger side / smaller side to keep it rotation invariant).
    probably worse than convex hull
'EulerNumber'
    number of objects minus the number of holes in the objects, we only use one object with holes (cup)

Usefull regionprops:
'ConvexHull'
    Convex hull around the blob, usefull when checking the number of vertices.
'Extent'
    ratio of number of pixels in the object to the number of pixels in the bounding box, probably usefull
    solidity does the same but with convex hull and is most likely better

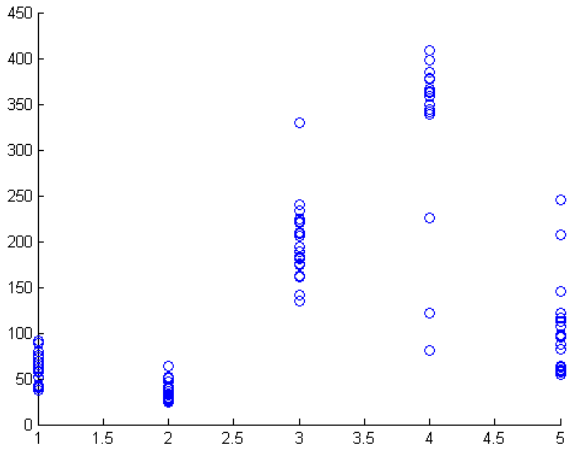
'Eccentricity'
    mathemagical multivariante normaldistribution of pointsin a covariance matrix, this can be interpreted as an ellipse with the same second-moments as the region spanning the blob. The eccentricity is the ratio between the radius of the innermost and outermost point.
    usefull
'Solidity'
    ratio between the pixels in the blob and the pixels in the convex hull, usefull

‘Roundness’

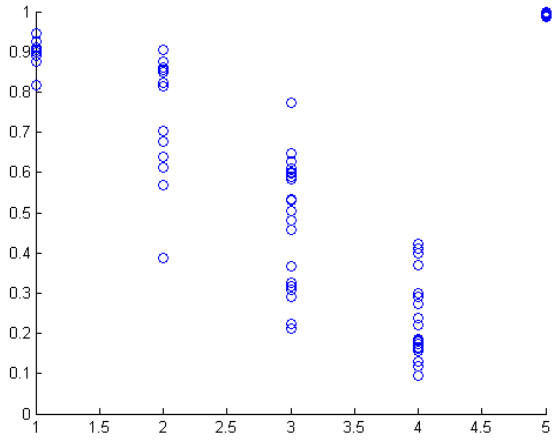
    Roundness of an object, ranges from 0.00 to 1.00. If the Roundness is greater than 0.90, the object is circular in shape. Useful for example when comparing a set of round and angular objects

Chose 5 object classes:
bat, beetle, cup, device 9, pencil

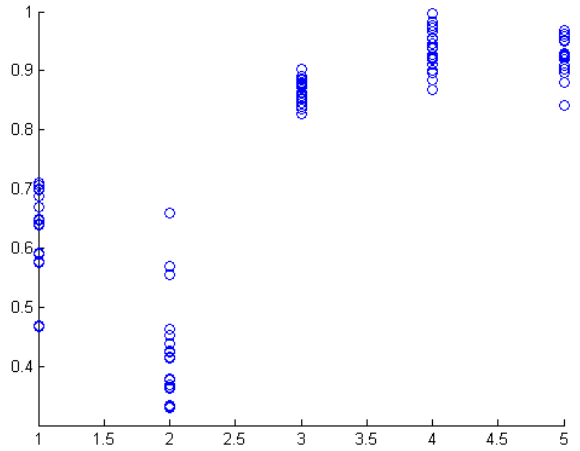
Draw/write up a simple algorithm containing the steps to go through to recognize each type of your chosen objects:
ConvexHull



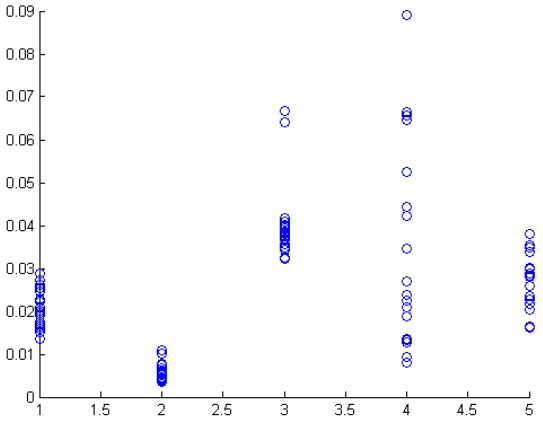
Eccentricity



Solidity



Roundness

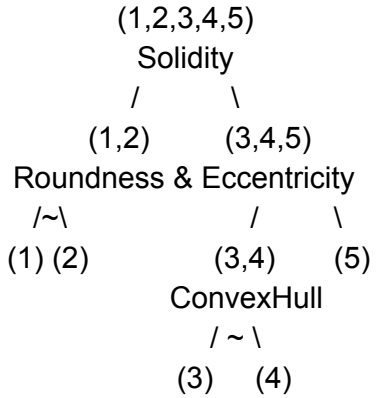


1. bat
little variation between images, always at least slight overlap with beetle,
no overlap in solidity (except beetle in solidity)
roundness + solidity + (maybe eccentricity) = mostly unique

2. beetle
little variation in roundness and convexhull, always overlap with bat
no overlap in solidity (except bat)
roundness + solidity

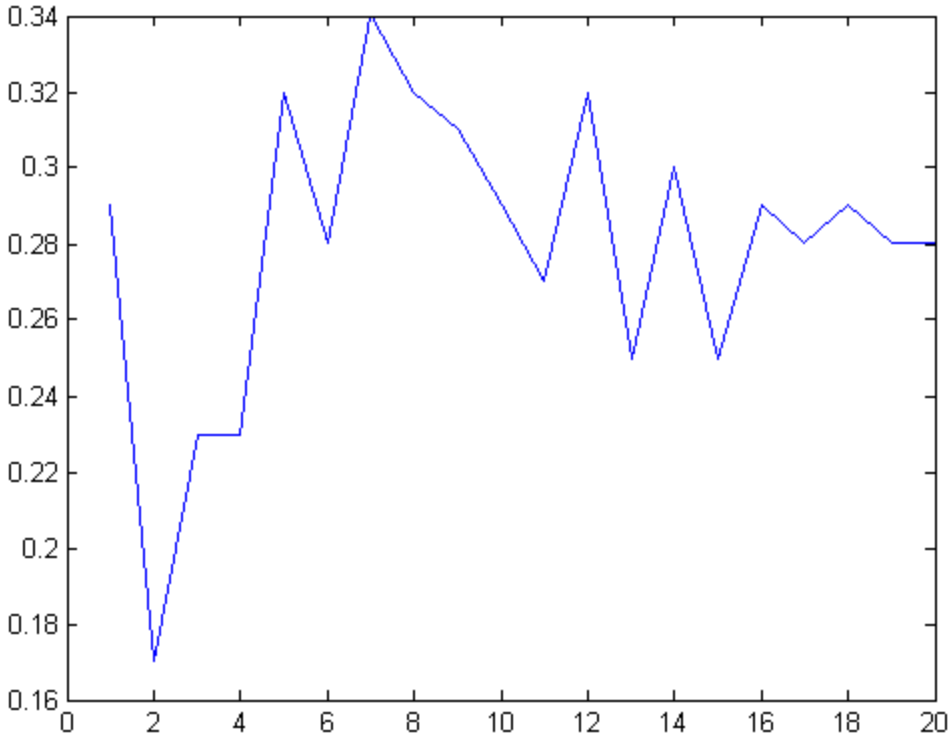
By thinking the features though like this for all classes we arrive at the following decision tree

Decision tree:



Are the features used by your algorithm position, orientation and scale invariant? - Invarcianes of features.
Yes, all features are invariant to scale, position and orientation

How well does your algorithm perform (percentage of objects correctly classied)? - Evaluation of the algorithm.
The finetuned algorithm correctly classifies 96% of objects. One cup is recognized as a device9 and 3 device9 are recognized as cups.



For the given set the algorithm performs best with 2 neighbors (83%), but the accuracy fluctuates around 72% depending on the amount of neighbors. It gets more stable when many (16-20) neighbors are taken into account.
It's not a good idea to chose 2 as the amount of neighbors since that accuracy will be vastly different for different test sets. About 20 neighbors seem to be a good idea.

You should not train and test your classier on the same data. Why? Why would doing this be extremely stupid for the k-NN classifier?
It introduces an automatic bias to the correct result for the test data because the nearest neighbor (with a distance of 0) is the input itself.
It falsifies the accuracy of the algorithm and won't work as well with test data that is separate from the learning set.

The usual approach is to divide the data into a training and test set. However, doing this usually only works well if you have a large amount of training data. Why?
Small trainings sets are prone to random noise which falsifies the results. For example, a new vector could be associated to the wrong class when that class only contains outliner values by chance due to a low amount of data.
A large difference of the amount of data between two classes is also bad (e.g. class A has 3 entries and class B has 20, the chance to get B as a nearest neighbors for a new A is high and

for k = 7 guaranteed)

When Is Nearest Neighbor Meaningful Summary

In general, the article discusses how useful and efficient solutions can be found for the nearest neighbor problem in high dimensional spaces. In general, the nearest neighbor problem tries to find the point in a dataset which is closest to a given point in a specific dataset. Answering this question can get difficult by several factors, such as little difference between the distance to other points, when you only have approximations to the position of other data points which blend into each other, or if too few or too many near datapoints can be found nearby. In the formulation and further analysis of the nearest neighbor problem, the variable *m* is used to describe the dimensionality of a set of data points. In a theorem following this explanation, the authors want to prove that as m increases, the distance between a query point and all data points nearby converges to the same value, making the nearest neighbor concept useless in this situation. To determine the theorem's impact, the following two issues must be adressed: How restrictive the condition necessary to hold the resulting dataset must be, and at what point or dimensionality the determinaton of a nearest neighbor becomes useless.

In the next section of the paper the applicability of said theorem is tested in certain situations by determining if the condition in the second equation is met or not. This is done by either showing that distances converge with independent and identically distributed (IID) dimensions or that the conditions of the analyzed equation are not met if the data and queries fall on one line. Furthermore, scenarios where the “Weak Law of Large Numbers” cannot be applied are illustrated by discussing examples involving correlated attributes and differing variance between dimensions..

Assuming that the data distribution and query distribution are IID in all dimensions, all the appropriate moments are finite and the query point is chosen independently of the data points, the conditions of the first theorem are met, although they are not necessary for the theorem to be applicable. In the next example shown in the article, all dimensions of the query points as well as the data points follow identical distributions, but are completely independent. This way, the underlying query can be converted to a one-dimensional nearest neighbor problem, and the condition of Theorem 1 is not satisfied. In another example which meets the conditions of the theorem,all dimensions are correlated with all other dimensions and the variance of each additional dimension increases. Afterwards, further cases are discussed where the conditions of the first theorem are met although the variance of the distance in each added dimension converges to 0.
A last example shows that the marginal distributions of data and queries change with the dimensionality, where the weak law of large numbers cannot be applied.

Next, meaningful applications of high dimensional indexing are being discussed, meaning that theorem 1 should not be seen as proof that high-dimensional indexing is never meaningful. One way is to use exact match and approximate match queries. For instance, a requirement can be added that the query point must be within a small distance δ of a data point, although it gets increasingly difficult to meet it as the dimension m increases. In this case, the query point simply needs to get closer to a data point. Further generalization can be achieved by creating a data set consisting of randomly choosen points and some additional points which are being distributed in a certain area around the original points. This way, the query is required to fall within one of those data clusters.
Another scenario would be when the underlying dimensionality of the data is much lower than the actual dimensionality.

The following subject is about Experimental Studies of the Nearest Neighbor problem. The theorem that was discussed before simply describes a convergence but not the rate of that convergence. One way to do so is to create a so-called “two degrees of freedom” workload which generates query and data points on a two-dimensional plane. This method quickly fails though as dimensionality increases.

In the next step the performance of a NN procssing technique is being analyzed. Situations where high dimensional nearest neighbor queries are meaningful occur only when using specific datasets, for example a set which consists of small, well-formed clusters, and the query is guaranteed to land in or very near one of these clusters.

When comparing different NN techniques with the linear scan algorithm, it is shown that linear scan is much more efficient if few dimensions are used.

In related work, “The Curse of Dimensionality” is being used as an indication that a high dimensionality might cause problems in certain situations. For the nearest neighbor problem, it means that a query processing technique performs worse as the dimensionality increases. The term is also used to describe a problem where the estimate of the NN query cost can be very poor if so-called “boundary effects” are not taken into account, which means that the query region is mainly outside the hyper-cubic data space.
In computational geometry, the approximate nearest neighbor problem is defined as a weaker problem.
While fractal dimensionality can be a good tool in determining the performance of queries over the data set, there are also examples where this is not the case. For instance, if the data points are sampled uniformly from the vertices of the unit hypercube, its fractal dimensionality is 0.