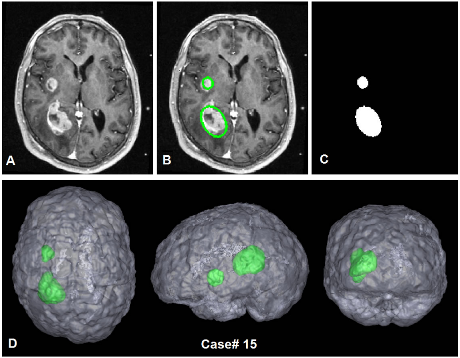
Augmented reality: Lesson 3
Clustering, K-Means, Image segmentation

Francesco Di Giacomo

Augmented reality course
Francesco Di Giacomo
fdigiacom@gmail.com

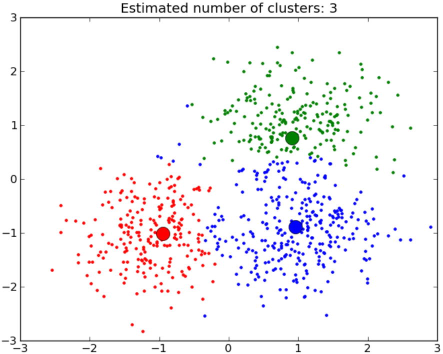
Image segmentation

- Partition an image into superpixels.
- A superpixel is a set of similar pixels.
- Goal: change portions of the image into something which is easier to analyse.
- Applications: medical imaging, facial recognition, etc.



Clustering problem

- We have a set of *objects* (*test set*).
- Each object has a set of *features*.
- All the objects have the same features.
- By analysing the features divide the objects with similar features into k groups (*classification*).



Sample model

- The set of features can be seen as an ordered sequence of elements (or *tuple*).
- A vector is an ordered sequence of space coordinates.
- The features of an object can be represented by space coordinates.

Assigning values to features

- **Continuous features**: Continuous features can be modelled by real numbers (floating point numbers in a computer).
- **Discrete features**: Discrete features can be thought as *labels*. Assign a value to each label.

Example:
Person: (Age, gender, weight, height, hair colour, eye colour)
Object 1: (25, male, 75kg, 1.82m, blonde, green)
Object 2: (23, female, 52kg, 1.68m, black, brown)
The gender, hair and eye colours are discrete features. We can assign a value to each possible value they can take (green = 0, brown = 1, black = 2, ...).

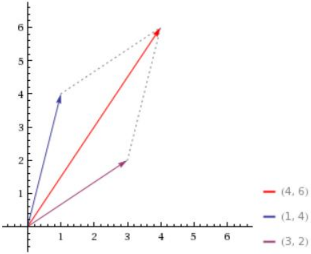
- A vector is an oriented segment defined by its *magnitude* (length) and *orientation* (direction).
- Parallel vectors with the same magnitude are equivalent: they have the same orientation.
- The starting point does not count.
- The vector going from $(0, 0)$ to $(1, 4)$ is equal to the one going from $(1, 3)$ and going to $(2, 7)$.

The sum of two vectors is simply the sum of their components. The sum is commutative.

$$
\begin{aligned}
v_1 &= (x_1, y_1) \\
v_2 &= (x_2, y_2) \\
v_1 + v_2 &= (x_1 + x_2, y_1 + y_2)
\end{aligned}
$$

Geometrical meaning: Point reached by putting $v2$ on the endpoint of $v1$.

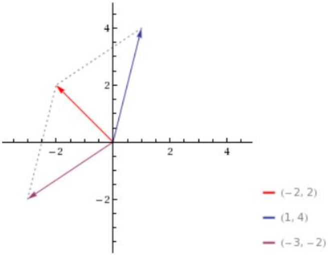
Example:



The difference of two vectors is simply the difference of their components. The difference is not commutative!

$$v_1 = (x_1, y_1)$$
$$v_2 = (x_2, y_2)$$
$$v_1 - v_2 = (x_1 - x_2, y_1 - y_2)$$

Geometrical meaning: Vector going from $v2$ to $v1$.
Example:



A vector can be multiplied by a *scalar* (a real number) to be scaled up or down.
This affects only the vector length (no change of orientation). The scalar product is commutative.

$$v \cdot k = (x \cdot k, y \cdot k)$$

Example: (1,4) * 3 = (3,12)

The length of a vector can be obtained by using Pythagoras theorem. Consider the right triangle whose catheti are the x and y coordinates. The length is the length of the hypothenuse.

Pythagoras theorem: $\|v\| = \sqrt{x^2 + y^2}$

The distance between two vectors is the length of the vector joining their endpoints. It can be computed in two ways:

Pythagoras theorem: $d(v_1, v_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

Length of the difference: $d(v_1, v_2) = \|v1 - v2\|$

All the functions defined above can be generalized for vectors with n components. The syntax $x_1^{(i)}$ denotes the i-th component of $v1$. The same for $v2$.

- **Sum:** $v_1 + v_2 = (x_1^{(1)} + x_2^{(1)}, x_1^{(2)} + x_2^{(2)}, ..., x_1^{(n)} + x_2^{(n)})$
- **Difference:** $v_1 - v_2 = (x_1^{(1)} - x_2^{(1)}, x_1^{(2)} - x_2^{(2)}, ..., x_1^{(n)} - x_2^{(n)})$
- **Length:** $\|v\| = \sqrt{\sum_{i=1}^{n} x^{(i)}}$
- **Distance:** $d(v_1, v_2) = \sqrt{\sum_{i=1}^{n}(x_1^{(i)} - x_2^{(i)})^2}$

- A cluster is defined by his *centroid*.
- A *centroid* is an object whose features are the mean of the features of the points belonging to the cluster.
- An object belongs to a cluster if the distance between it and the centroid is the minimum among all the other centroids.

Choose the number k of clusters. We must initialize their centroids. There are two options of initialization:

- **Randomized coordinates:** Create k points with random coordinates.
- **Randomized object:** Choose k distinct points from the test set and use them as starting centroids.

Assign the object to the clusters: For each object v in the test set do the following steps:

1. Compute the square distance between v and each centroid k of each cluster ($d^2(v,k)$).
2. Assign the object v to the cluster k.

Update the centroids:
For each cluster k compute their average vector.
The average vector is computed analogously to the average of a number: we sum all the vectors in the cluster (with the vector sum) and we divide (scalar product with the inverse) by their amount.

Repeat the two steps above

- In theory the algorithm terminates when no object is reassigned to a different cluster.
- Convergence in practise might require a lot of time.
- Fix a maximum number of iterations i_{max} to perform.

Image segmentation with K-Means

- The RGB (or YUV) colour space is our geometrical space.
- Convert the RGB coordinates into 3D Vectors.
  Do not use directly the colour values. Convert them into floating point numbers.
- Run the K-means algorithm with the vectors.
- Set each object of the cluster to the colour of its centroid.
- Rebuild the output image with the new colour values.

Examples of segmentation

Figure: Forest image segmented with 4 clusters

Figure: Forest image segmented with 6 clusters

Figure: Forest image segmented with 10 clusters

Examples of segmentation

Figure: Forest image segmented with 16 clusters

Figure: Forest image segmented with 32 clusters

Assignment 2

Implement the K-Means algorithm and apply it to image segmentation. Initialize the centroids by picking random objects from the test set. Take the number k of clusters as input. Visualize the input image and the segmented image on the same window.

Hints for the tasks to accomplish:

1. Build a class implementing vectors and vector operations seen in class.
2. Build a function that converts a matrix of `Color` into a matrix of vectors.
3. Build a function that picks k distinct random points from the vector matrix.
4. Run the algorithm on the vectors. Take as input the number of clusters. Keep track of what cluster is assigned to each vector.
5. Build the output image by setting the object colours to be the one of their centroid.