IAML - Week 7

Antonio León Villares

December 2021

Contents

1 K-Means

- Clustering is used to determine underlying patterns in subpopulations of data
- Clustering can be monothetic or polythetic, hard or soft, flat or hierarchical
- K-Means produces K clusters, in which the intra cluster distance is minimised
- K-Means converges to local optima; to pick K, a scree plot can be used
- To evaluate clustering, we can use intrinsic methods (human clustering, reference clustering) or extrinsic methods

1.1 Defining Clustering

Here is a good overview article of most of what will be discussed about clustering: Clustering, and its Methods in Unsupervised Learning

1.1.1 Purpose of Clustering

- **How does clustering differ from classification?**
  - in classification, we learn a model to predict the class of a new data instance
  - in clustering, we attempt to find underlying patters in unlabelled data
- **What questions does clustering answer?**
  - we can use it to ask questions on the composition and distribution of subpopulations in the data
    - how many subpopulations are there?
    - how many members does each subpopulation have?
    - what are common features in subpopulations?
  - this need not mean that if the data had labels, clustering will perfectly split the data based on these labels: it just finds common, impicit patterns in the data

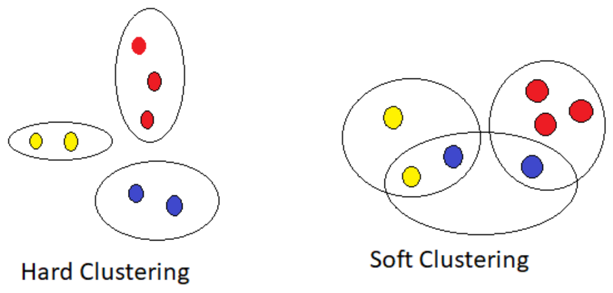
1.1.2 Types of Clustering

- **What are the 3 types of clustering?**
  - Monothetic vs Polythetic

  - Hard vs Soft
  - Flat vs Hierarchical

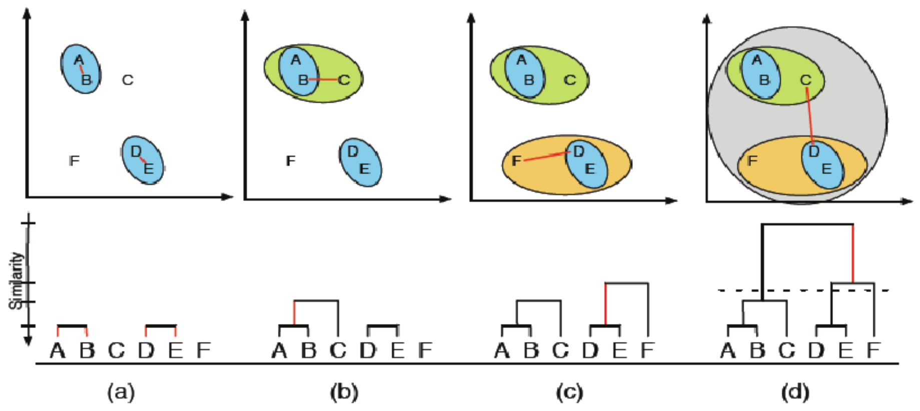
- **How does monothetic clustering differ from polythetic clustering?**
  - in monothetic clustering, we are able to pinpoint the precise property that elements in a cluster have in common (for example, a cluster which represents children who prefer maths over sport - we can quickly label the common property)
  - in polythetic clustering, we can see that elements in the cluster are similar, but we can't pinpoint the exact reason for the similarity (in this case we cluster based on distance - similar points will appear close together, even if they don't all share a common property)

- **How does hard clustering differ from soft clustering?**
  - in hard clustering, each data point belongs to exactly one cluster (that is, clusters partition space exactly)
  - in soft clustering, single data points can belong to different clusters (that is, clusters overlap)
  - in soft clustering, its more interesting to think about the strength of association of a point to a given cluster (for example, a platypus can be part of a mammal and bird cluster, but will be more strongly associated with a mammal)



- **How does flat clustering differ from hierarchical clustering?**
  - in flat clustering, we aim to produce subpopulations which can't be further split (for example, a set of animals subdivided into "dog", "parrot","penguin", "cat", "goldfish")
  - in hierarchical clustering, we can vary the "grain" with which we subdivided the populations (for example, we can choose to be more coarse and split the set of animals into "mammals", "birds" and "fish" - it provides a taxonomical way of splitting the data)

Example: Hierarchical Agglomerative Clustering



1.1.3 Clustering Algorithms

1. **KD-Trees**: splits regions of space based on the median of the subpopulations (each split region can be thought as a cluster
   - *monothetic* (each element in a cluster is defined by the fact that its attributes have values bounded by a set of medians)
   - *hard* (space is partitioned exactly)
   - *hierarchical* (we can reduce or increase the number of splits which we do)
2. **K-Means**: split data into K distinct clusters based on distance
   - *polythetic* (clustering is distance dependent, not based on a single feature)
   - *hard* (each element in a cluster belongs to the cluster only)
   - *flat* (we specify the number of clusters, so subdivisions are more coarse or fine depending on K)
3. **Gaussian Mixture Models and Expectation Maximisation**: fit a mixture of K Gaussians to the data
   - *polythetic* (clustering is dependent on mean and covariance of data, so not one single feature which defines elements of cluster)
   - *soft* (since this based on probability distributions, many Gaussians can be thought as generating a single point)
   - *flat* (we specify the number of Gaussians to fit, so no hierarchy)
4. **Agglomerative Clustering**: produce clusters by joining smaller clusters together

- *polythetic* (clusters are grouped based on how close they are to each other, not because of a distinguishing feature)
- *hard* (clusters won't overlap)
- *hierarchical* (we can specify how many cluster groupings to perform)

1.2 K-Means Clustering

1.2.1 Basis and Uses

- **How does K-Means define a cluster?**
  - data is clustered around K *centroids*
  - a *centroid* is meant to represent the prototypical element of a given cluster
  - we need to determine how many clusters K to use
- **How is K-Means used in practice?**
  - to determine classes in an unsupervised way (for example, using the MNIST dataset, and setting $K = 10$, as to cluster the 10 numbers)
  - provide a smooth representation of elements in data (elements within a clsuter should be similar)
  - dimensionality reduction (can make cluster membership an attribute of each data point, and train a classification algorithm on that; alternatively, set K to be the dimensions that we want to reduce to, and redefine each point based on the distance to each of the cluster centres, training a classifier on this modified data [see this stack exchange and this medium article]

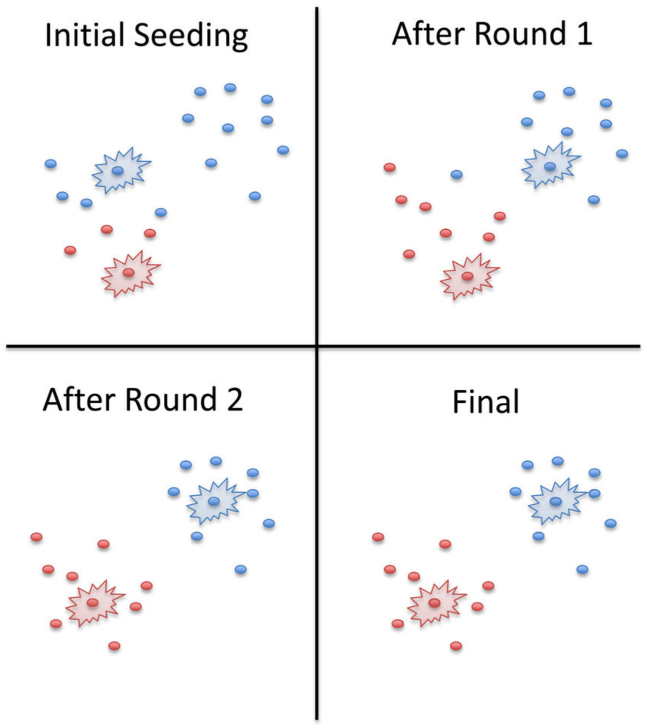
1.2.2 Algorithm

The input to the algorithm is:

- K (number of clusters)
- $\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_n$ (data points)

1. Initialise K random centroids $\underline{c}_1, \underline{c}_2, \ldots, \underline{c}_K$
2. For each $\underline{x}_i$, compute $\arg\min_j\{D(\underline{x}_i, \underline{c}_j)\}$ (the centroid to which $\underline{x}_i$ is closest to), and assign $\underline{x}_i$ to cluster j
3. For each cluster $j = 1, 2, \ldots, K$, recompute its centroid. In the case of D as the Euclidean Distance, this will just be the mean of all points in the cluster:
$$\underline{c}_j = \frac{1}{n_j} \sum_{\underline{x}_i \in j} \underline{x}_i$$

4. Repeat until cluster members don't change (in practice, if there are lot of points, we perform these steps a given number of times, since convergence may be too expensive)



Given N points in D dimensions, and assuming the algorithm runs for I iterations, the runtime is:

$$\mathcal{O}(KIND)$$

1.2.3 Properties

- **Task**: unsupervised/generative method to group data into K clusters
- **Model**: K centroids (D dimensional vectors)
- **Score Function**: average distance from centroid to cluster instance
- **Optimisation**: iteratively assign points to clusters and recompute centroids
- **What is K-Means optimising?**

  - the algorithm tells us that K-Means aims to **minimise** the *aggregate intra-cluster distance*. That is, it wants to reduce the total distance between each centroid and its cluster elements:

$$\min \left[\sum_{j=1}^{K} \sum_{\underline{x}_i \in j} D(\underline{x}_i, \underline{c}_j)^2 \right]$$

    (we square for convenience)
  - if D is the Euclidean Distance, this is equivalent to minimising the total variance within the clusters

- **Is K-Means guaranteed to cluster optimally?**
  - K-Means will attain a local minimum
  - this means that it is sensitive to the initial position of the centroids: changing them means that clustering can change
  - to mitigate this, K-Means can be run repeatedly, as to select a clustering which provides the smallest aggregate intra-cluster distance

- **Does K-Means always cluster like a human would?**
  - since K-Means is sensitive to its starting point, it might cluster in a way that might not make sense to us, but is perfectly valid algorithmically
  - for instance, it doesn't guarantee that if 2 points are close to each other, they will belong to the same cluster

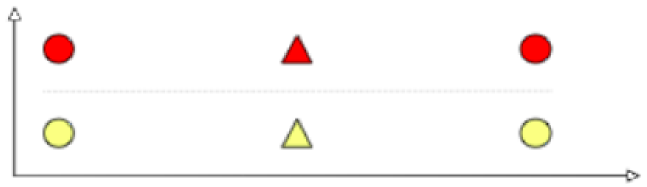


Figure 1: An example of how initial conditions lead to a local minimum. If the cluster centres (triangles) are placed to be equidistant between the red and yellow points, we attain a local minimum. The centroid is already at the average point of its cluster elements, so the clusters won't change.

1.3 Finding Number of Clusters

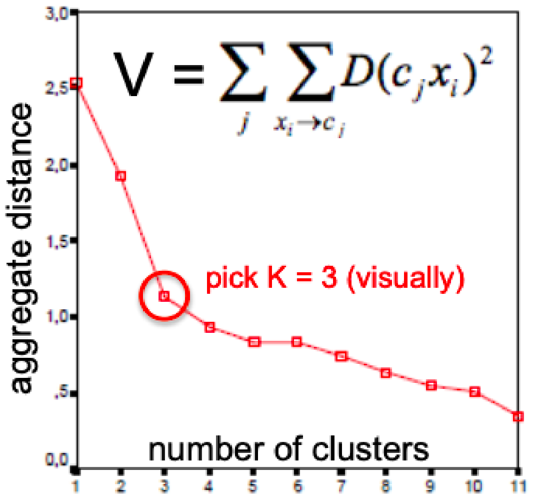
- **What is the optimal value of K?**
  - if we have N data points, $K = N$ can be thought as optimal
  - it means that the aggregate intra-cluster distance will be 0 (because each point is its own cluster centre)

  - however, this is not informative: it gives us no pattern with regards to the data

- **Can we use a validation set to determine K?**
  - we can define a validation set, run K-Means with different K, until we find a minimum intra-cluster distance
  - however, 2 problems:
    - increasing K will tend to reduce intra-cluster distance
    - in some cases, $K = N$ will still be selected

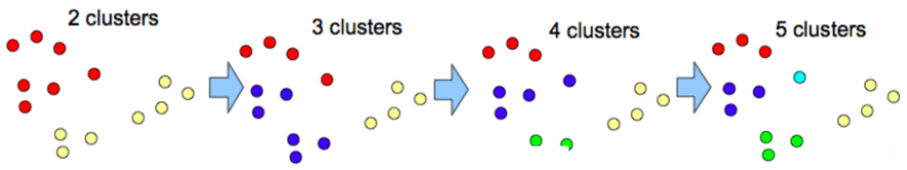
- **Can we use class labels to pick K?**
  - class labels can be used, but they remove the purpose of K-Means as an unsupervised learning algorithm

- **How is K determined in practice?**
  - we can plot a scree plot (aggregate distance vs K)
  - we need to look out for the "knee" or "elbow" of the plot: the point at which aggregate distance stops decreasing due to a better centroid, and starts decreasing by virtue of adding cluster centres



  - this is a consequence of the fact that aggregate distance won't decrease uniformly, and will tend to decrease by less as K increases (as more clusters are added, intra-cluster distance will already be small, so hard to improve)

- the "elbow" represents the point at which adding clusters leads to a big improvement in the metric



- mathematically, we can think of the "elbow" as the point at which the rate at which the metric declines changes the most (so we aim to maximise the second derivative of the metric)
- an alternative more complex method is the **minimum description length**, considering the smallest number of bits required to encode the K centroids and the distances from points to the centroids

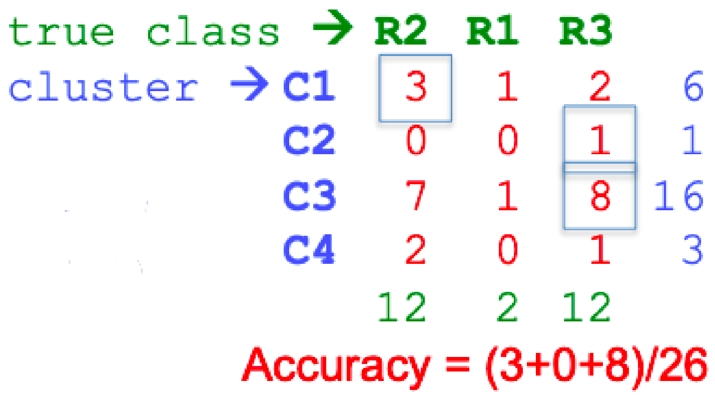
1.4 Evaluating Clustering

There are 2 ways of evaluating a clustering algorithm. This is not as easy with classification, where we have the "expected" labels available.

1. **Intrinsic Evaluation**
   - aims at evaluating how "good" the produced clusters are
   - *qualitative*: does the clustering improve our understanding of the underlying data?
   - *quantitative*: how well do the clusters compare with actual classes/labels?
2. **Extrinsic Evaluation**
   - aims at evaluating how helpful the produced clusters are in solving another problem
   - for example, in using a classifier, does previously clustering the data improve the classifier's performance?
   - improvements cna be brough by:
     - alternative data representation (i.e expressing images in terms of clusters)
     - training a separate classifier on each cluster (and comparing this with training the classifier on all of the data)
     - eliminating outleirs before running classification

1.4.1 Intrinsic Evaluation: Comparison to Reference Clustering

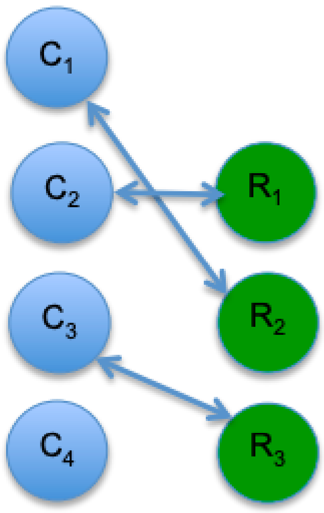
- **How can we use reference clusters to intrinsically evaluate our clustering?**
  - let $C_1, C_2, \ldots, C_K$ are the clusters from K-Means
  - let $R_1, R_2, \ldots, R_N$ be reference clusters (derived from using the class labels from the data)
  - we can produce a surjective correspondence between a C_i and a R_j
  - once this correspondence is built, we can compute our standard evaluation metrics, such as the number of true positives, true negatives, false positives and false negatives
  - notice, the number of system clusters need not be equal to the number of reference clusters
- **How can we create the surjective correspondence between the system and the reference clusters?**
  - we can follow a greedy algorithm
  - firstly, construct a grid by using the system clusters as rows, and the reference clusters as columns
  - each cell (i, j) in the grid should tell us the number of elements which are in both C_i and R_j
  - at the start, we can then assign a system cluster to the reference cluster with which it has maximum overlap
  - however, this can result in many system clusters being assigned to the same reference cluster
  - we follow a greedy approach: assign C_i to the reference cluster R_j, such that C_i is the system cluster with the largest intersection with R_j
  - for all other system cluster which have intersections with R_j, assign them to the next reference cluster with which they have the highest intersection
  - continue in this way, until all the reference clusters are mapped to
  - this can result in certain system clusters having no assignment, but the greedy approach maximises the data points that can be compared between clusters



true class →	R2	R1	R3	
cluster → C1	$\boxed{3}$	1	2	6
C2	0	0	$\boxed{1}$	1
C3	7	1	$\boxed{8}$	16
C4	2	0	1	3
	12	2	12	

Accuracy = (3+0+8)/26

Figure 2: At the start, the algorithm assigns $C_1 \rightarrow R_2, C_2 \rightarrow R_3, C_3 \rightarrow R_3, C_4 \rightarrow R_2$. To resolve R_2, we notice that C_1 has 3 overlaps, whilst C_4 only has 2 overlaps. Hence, we assign $C_1 \rightarrow R_2$, and $C_4 \rightarrow R_3$. Clearly, C_3 has the highest intersection, so $C_3 \rightarrow R_3$, and $C_4 \rightarrow R_1, C_2 \rightarrow R_1$. Once again we have to resolve this tie. We can choose to drop C_4, and we get the final assignment given below:



- once we have the table, computing accuracy is extremely easy: its the sum of the number of intersections, divided by the number of elements in the reference clusters

- **Why can't multiple system clusters be assigned to one reference cluster?**
  - consider a system clustering, in which each data point is a cluster

  - if we allowed multiple system clusters to be assigned to one reference cluster, each singleton cluster (data point) will just be assigned to its correct reference cluster (since it will have 1 intersection with it, and 0 intersections with the rest)
  - this then means that we would get 100% accuracy, even if the singleton clusters don't represent the reference clusters well

- **Why can't a single system cluster be assigned to multiple reference cluster?**
  - consider a system clustering, in which all data points are in one cluster
  - if we allowed a cluster to be assiegned to multiple reference cluster, then we could assign this giant cluster to all reference clusters
  - again, we would get 100% accuracy, even though the large cluster is not representative of the class distribution, and it isn't useful

- **Why can't we allow system clusters to overlap?**
  - consider a clustering technique which produces soft clusters
  - assume that we don't allow multiple assignments
  - we could still cheat the system
  - for example, a clustering system which clusters using the power set of the data
  - then, since the reference clusters are just subsets of the data, this power set clustering will always have a cluster which is in one-to-one correspondence with another reference cluster
  - again, we would get 100% accuracy, but the power set is literally giving all possible ways of clustering the points, which is not informative

1.4.2 Intrinsic Evaluation: Comparison to Human Clustering

- **How can humans help to intrinsically evaluate a clustering?**
  - we can't ask humans to cluster the whole dataset
  - instead, we can ask whether 2 instances x_i, x_j should be clustered together
  - once the algorithm clusters, we can count errors in clustering, compared to what humans say, and derive measures such as accuracy, TP, TN, FP and FN
  - *TP*: x_i, x_j clustered together & human says they should be clustered together
  - *TN*: x_i, x_j not clustered together & human says they should not be clustered together

- *FP*: x_i, x_j clustered together & human says they should not be clustered together
- *TN*: x_i, x_j not clustered together & human says they should be clustered together

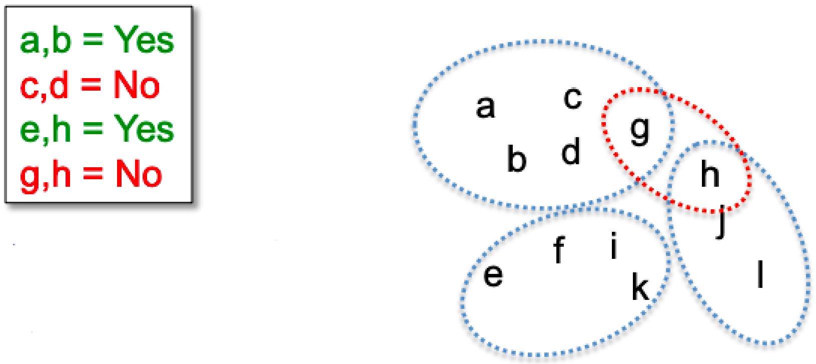


Figure 3: Given the sample of 4 pairs, we can see that (a,b) is a TP, (c,d) is a FP, (e,h) is a FN, and (g,h) is a TN.

- **How does this strategy compare with the alignment strategy above?**
  - simple: don't need to compute alignments with reference clusters
  - it can handle soft clustering (a given pair can be computed as 2 different metrics, like TN and FP)
- **Can this process be automatised?**
  - if we have class labels, we can automaticlaly generate the pairins

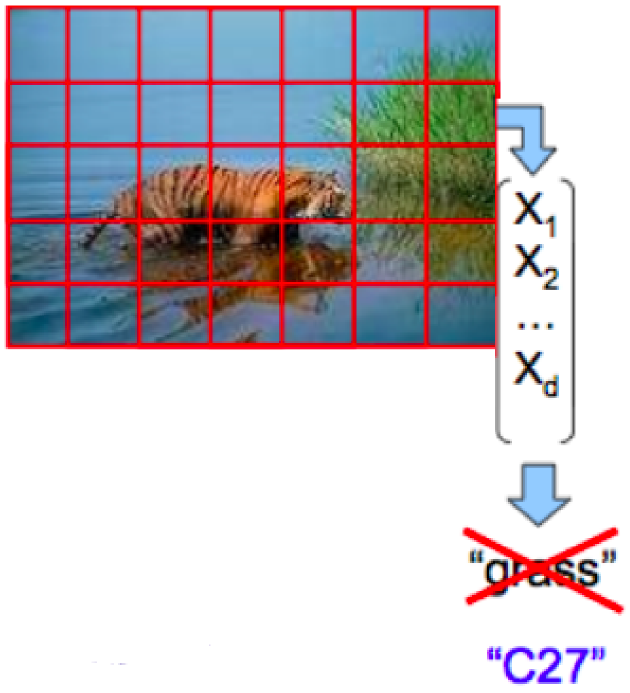
1.4.3 Extrinsic Evaluation: Applying Clustering to Image Representation

We consider the goal of detecting an object in an image.

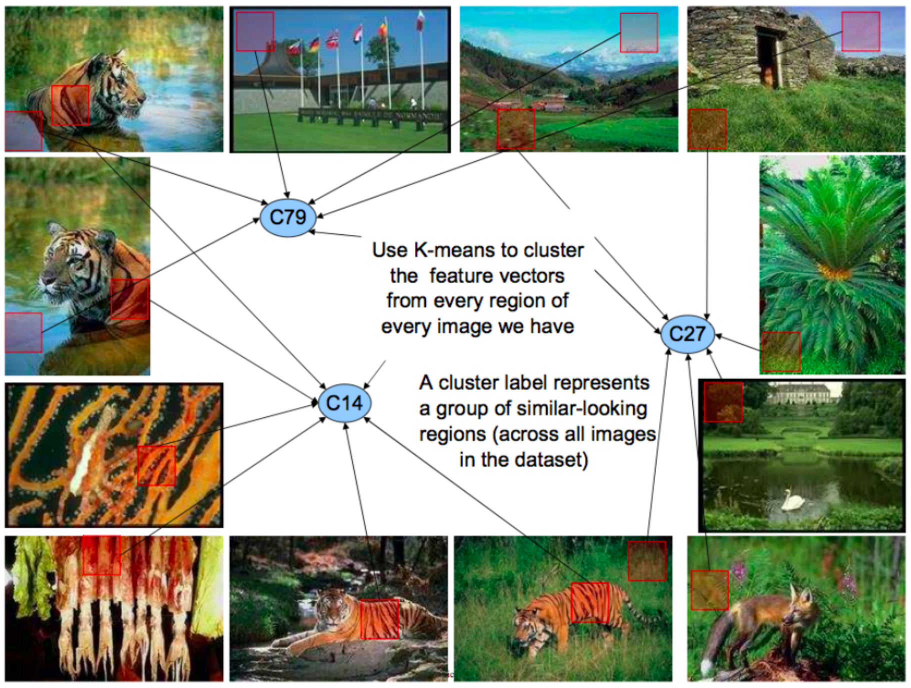
- **Can we represent an image as a vector of pixels?**
  - not too practical: decent resolution RGB images can have millions of pixels
  - not too useful for learning
- **Can we represent an image using a bag of words representation?**
  - naively, not too practical again: we would require humans to determine the elements of the image

  - we might as well use the human to determine if the object is present or not

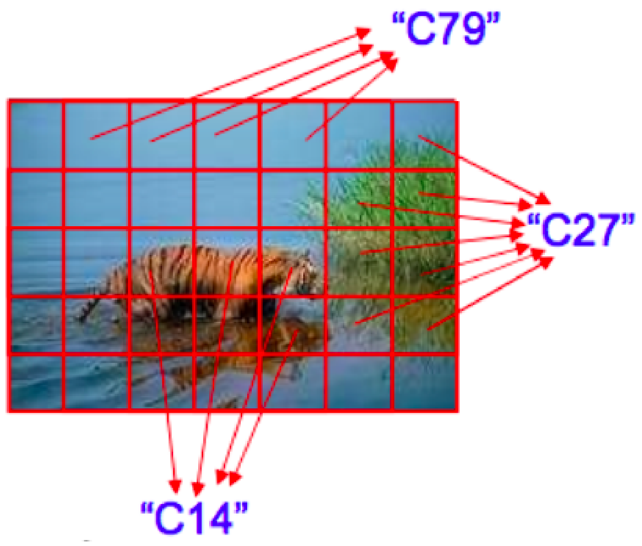
- **How does segmenting an image help detect an object?**
  - the idea is to "cut" the image up into small segments
  - for each segment, we can compute a feature vector, containing meta-data on that image (colour distribution histogram, orientation, edges, etc ...)
  - if we can meaningfully label such feature vectors (for example, as "grass", "water", "tiger"), we could potentially create a representation



- **How does K-Means help when segmenting the image?**
  - the idea is that the feature vectors of segments which are similar should be close together in space
  - if we apply K-Means, we could cluster similar feature vectors within a cluster (say C27)
  - in essence, we are then able to ID an image segment, based on to which cluster it belongs, with the idea that similar looking segments should have the same ID


C79
C27
C14
Use K-means to cluster the feature vectors from every region of every image we have
A cluster label represents a group of similar-looking regions (across all images in the dataset)

– we can then produce a K dimensional representation of an image, based on how many times a given cluster ID appears in the image (for example, $\langle 0, 0, 0, 4, 12, 0, 0, 13 \rangle$ could indicate that C4 appears 4 times, C5 appears 12 times, and C9 appears 9 times)



2 Gaussian Mixture Models

- GMMs are probabilistic clustering models which perform soft clustering
- GMMs aim to maximise the likelihood, but can't do this directly, so they use expectation maximisation
- Selecting the number of Gaussians is a complex problem; Occam's Razor can be used as a heuristic
- In many cases, GMMs are similar to K-Means

2.1 Motivating Mixture Models

- **What are the 2 types of clustering?**
  - **hard clustering**: points belong to a single cluster (2 clusters never overlap)
  - **soft clustering**: points can belong to more than one cluster (clusters may overlap, with a point having different "strength of association" for different clusters)
- **What are mixture models?**
  - a probabilistic form of soft clustering
  - clusters are defined by **generative models**
  - for continuous cases, the model can be a **Gaussian**
  - for discrete cases, the model can be a **Multinomial**

2.2 The Chicken and Egg Problem: Gaussian Mixture Models

- **What are Gaussian Mixture Models?**
  - the use of Gaussian distributions to form a mixture model
  - we can think of the data as being sampled from K Gaussian distributions
- **Are GMMs unsupervised methods?**
  - yes, GMMs are constructed in order to determine the parameters of the Gaussian Distributions
  - these parameters include:
    * the mean μ of the distribution (vector or real number)

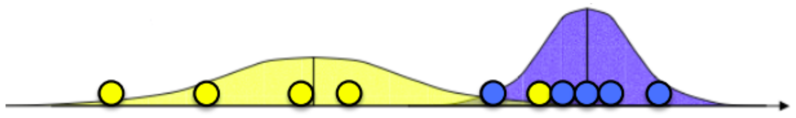
    * the variance σ or covariance matrix Σ
    * the prior probabilities of the classes that need clustering

- **How could the Gaussians be estimated if we knew the class labels?**
  - lets assume we want to use 2 Gaussians to estimate the distribution of data, and we now the class labels
  - even if we don't know the parameters of the Gaussians, we can estimate them, since we know the labels of each point. For example, for some class b:

$$\mu_b = \frac{1}{n_b} \sum_{x_i \in b} x_i$$

$$\sigma_b = \frac{1}{n_b} \sum_{x_i \in b} (x_i - \mu_b)^2$$

Figure 4: We consider points whose source (class) we know. From these, we can estimate the parameters of the Gaussians which generate said points.

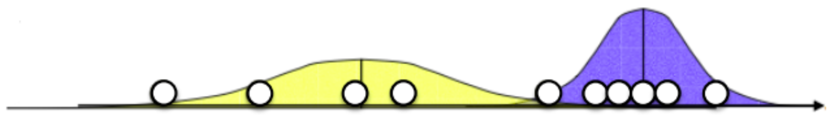


- **How can the class labels be estimated if we know the Gaussian parameters?**
  - lets assume the data is generated by 2 Gaussians of known parameters, but we have unlabelled data points
  - even if we don't know the labels of the points, we can use the Gaussian parameters to determine the probability of a given point being generated by the Gaussians. We can then assign a class label based on the highest probability.
  - to compute the probability we just have to use Bayes' rule, alongside the probability density formulae. For example, for the Gaussian modelling class b with mean μ_b and variance σ_b, alongside the prior of a point belonging to class b, $P(b)$:

$$P(x_i|b) = \frac{1}{\sqrt{2\pi\sigma_b^2}} \times \exp\left(-0.5 \times \left[\frac{x_i - \mu_b}{\sigma_b}\right]^2\right)$$

$$P(b|x_i) = \frac{P(x_i|b) \times P(b)}{P(x_i|b) \times P(b) + P(x_i|a) \times P(a)}$$

Figure 5: We consider unlabelled points generated by known Gaussians. From this, we can compute the probability of a given point being generated by a certain Gaussian.



- **What is the chicken and egg problem?**
  - we have seen that if we have the labels, we can estimate the Gaussian paramters
  - similarly, if we have the parameters, we can estimate the labels
  - in the case of GMMs, we have neither, so we can't figure out the parameters from the labels, or viceversa

2.3 The Expectation Maximisation Algorithm

- Generla overview of GMMs
- Thorough explanation of EM, explanation of which max log likelihood can't be used and even example of convergence of EM
- **Can we use likelihood to figure out the model parameters?**
  - before, when fitting a Gaussian to data, we used maximum log liklihood to derive our estimates for μ, σ^2, so it could make sense to use it for GMMs
  - the issue is that we are trying to fit many Gaussians, so the log likelihood becomes of fitting K mixture models to N data points results in:

$$L = \sum_{i=1}^{N} \log \left(\sum_{k=1}^{K} P(k) \times \frac{1}{\sqrt{2\pi\sigma_b^2}} \times \exp\left(-0.5 \times \left[\frac{x_i - \mu_k}{\sigma_k} \right]^2 \right) \right)$$

  - the issue is that differentiating this and attempting to solve anlytically for μ_k, σ_k^2 is **impossible**
- **What is Expectation Maximisation?**

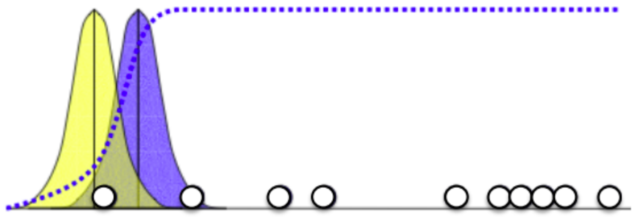
  - expectation maximisation (EM) is the technique used in order to, in a completely unsupervised manner, numerically derive the Gaussian Parameters for the K GMMs
  - EM is guaranteed to reach a local optimum (in terms of log likelihood)

- **What is the EM algorithm?**
  1. Initialise K random Gaussians
  2. For each point x_i in the data, compute $P(k|x_i)$ (**expectation** step), and assign x_i to the label with max probability (does x_i look as if its generated by k?)
  3. Readjust μ_k, σ_k^2 based on the points assigned to class k (**maximisation** step)
  4. Iterate until convergence (for example, when the likelihood changes by very little)

2.4 EM in 1 Dimension

We consider an example in which 2 Gaussians are fit to n datapoints.

1. Initialise 2 random Gaussians, calling them a and b, and defined the parameters $\mu_a, \sigma_a^2, P(a), \mu_b, \sigma_b^2, P(b)$:



   (technically, we only need $P(b)$ as a parameter, since $P(a) = 1 - P(b)$. In general, if we have K classes, we only need $K - 1$ priors).

2. Compute the probability of a point belonging to one of the Gaussians.

   (a) First, determine the probability of Gaussian b generating x_i. This is just $P(x_i|b)$, which can be computed using the Gaussian (technically this is a density):

$$P(x_i|b) = \frac{1}{\sqrt{2\pi\sigma_b^2}} \times \exp\left(-0.5 \times \left[\frac{x_i - \mu_b}{\sigma_b}\right]^2\right)$$

(b) Then, we can determine the probability of Gaussian x_i being of class b:

$$P(b|x_i) = \frac{P(x_i|b) \times P(b)}{P(x_i|b) \times P(b) + P(x_i|a) \times P(a)}$$

(c) We can use the same formulae with μ_a, σ_a^2 to compute $P(x_i|a)$ and $P(a|x_i)$

3. Once we have the probabilities for each data point, we can recompute the parameters of the Gaussians. For example, for class b:

$$\mu_b = \frac{1}{\sum_{i=1}^{n} P(b|x_i)} \times \sum_{i=1}^{n} P(b|x_i) \times x_i$$

$$\sigma_b^2 = \frac{1}{\sum_{i=1}^{n} P(b|x_i)} \times \sum_{i=1}^{n} P(b|x_i) \times (x_i - \mu_b)^2$$

We recompute the parameters by giving each point a weighting, dependeing on how probable x_i is of being generated by b. The same formulae apply by using $P(a|x_i)$ instead. Notice, we are not saying whether a point is generated by exactly one of the Gaussians, but rather that both Gaussians can potentially generate the point. As EM progresses, we expect the probabilities to be better defined.

4. After these steps, we can either keep the priors $P(a), P(b)$ the same as at the start, or update them, via:

$$P(b) = \frac{1}{n} \times \sum_{i=1}^{n} P(b|x_i)$$

$P(a)$ can be updated in a similar way (in this case, we can just use $P(a) = 1 - P(b)$ since there are only 2 classes)

5. We then repeat these steps, until the parameters don't change too much. For the example shown above:

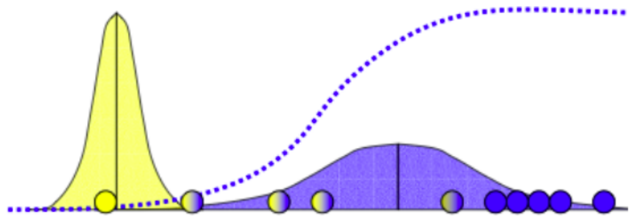


Figure 6: We can see that the blue Gaussian has high variance, since it is trying to be generative for most data points. On the other hand, the yellow Gaussian is only close to 2 points, so it has much lower variance.

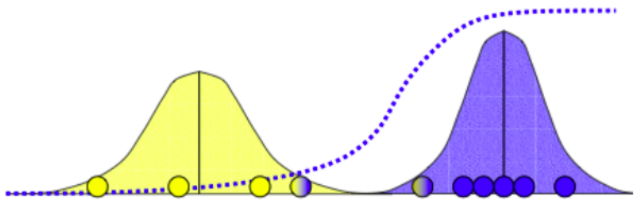


Figure 7: After more steps, the Gaussians are more differentiated.

2.5 EM for Multidimensional Data

For multidimensional data, we follow very similar steps. The main difference is that we work with vectors and matrices. Consider n points of D dimensions. We consider fitting K Gaussians $c_1, c_2, \ldots, c_K$.

1. Enumerate the K Gaussians randomly, such that c_j is defined by $\underline{\mu}_{c_j}, \Sigma_{c_j}, P(c_j)$.

2. For each point $\underline{x}_i$, determine the probability of each c_j generating x_i:

$$P(\underline{x}_i|c_j) = \frac{1}{\sqrt{(2\pi)^D \times |\Sigma_{c_j}|}} \times \exp\left(-0.5 \times (\underline{x}_i - \underline{\mu}_{c_j})^T \Sigma_{c_j} (\underline{x}_i - \underline{\mu}_{c_j})\right)$$

$$P(c_j|\underline{x}_i) = \frac{P(\underline{x}_i|c_j)P(c_j)}{\sum_{m=1}^{K} P(\underline{x}_i|c_m)P(c_m)}$$

3. We now compute a **weight** indicating the importance of x_i for the Gaussian c_j. We can think of this in teh following way: out of all the points generated by c_j, how important is x_i. This is just for algebraic convenience:

$$w_{i,j} = \frac{P(c_j|\underline{x}_i)}{\sum_{m=1}^{n} P(c_j|\underline{x}_m)}$$

4. We can now recompute the parameters of c_j. The mean is given by:

$$\underline{\mu}_{c_j} = \sum_{i=1}^{n} w_{i,j}\underline{x}_i$$

The mean of attribute a will be given by:

$$\mu_{c_j,a} = \sum_{i=1}^{n} w_{i,j}x_{i,a}$$

The covariance of 2 attributes a and b is given by:

$$\Sigma_{c_j,a,b} = \sum_{i=1}^{n} w_{i,j}(x_{i,a} - \mu_{c_j,a})(x_{i,b} - \mu_{c_j,b})$$

so we can construct:

$$\Sigma_{c_j} = \begin{pmatrix} \Sigma_{c_j,0,0} & \Sigma_{c_j,0,1} & \dots & \Sigma_{c_j,0,D} \\ \Sigma_{c_j,1,0} & \Sigma_{c_j,1,1} & \dots & \Sigma_{c_j,1,D} \\ & \vdots & \ddots & \vdots \\ \Sigma_{c_j,D,0} & \Sigma_{c_j,D,1} & \dots & \Sigma_{c_j,D,D} \end{pmatrix}$$

5. If we want to recompute the prior:

$$P(c_j) = \frac{1}{n} \times \sum_{i=1}^{n} P(c_j|\underline{x}_i)$$

6. We repeat this until convergence

2.6 Determining the Number of Mixture Models

- **Can we use likelihood to determine K?**
  - no, since to maximise likelihood, we can just pick $K = n$
  - this means that each data point has its own Gaussian, with extremely small variance
  - this clearly won't generalise well
  - $$L = \sum_{i=1}^{N} \log \left(\sum_{k=1}^{K} P(k) \times P(x_i|k \right) = \sum_{i=1}^{N} \log \left(\sum_{k=1}^{K} P(k) \times \frac{1}{\sqrt{2\pi\sigma_b^2}} \times \exp\left(-0.5 \times \left[\frac{x_i - \mu_k}{\sigma_k} \right]^2 \right) \right)$$
- **Can we use validation to determine K?**
  - create a validation set V, and for each K, compute the likelihood of the model on V
  - in some cases this can still give $K = n$
- **What other technique can we use?**
  - Occam's Razor: pick the simplest model
  - if p are all the parameters of the GMM, we can measure simplicity via:
    * Bayes Information Criterion: $\max_p \left\{ L - \frac{1}{2} p \log n \right\}$
    * Akaike Information Criterion: $\min_p \{2p - L\}$
  - recall the parameters are given by the means, variances and priors. For K GMMs, we expect $3K - 1$ parameters.
  - ultimately, we can decide on a good value of K based on how it helps us solve another task that requires clustering (extrinsic evaluation)

2.7 Comparing K-Means and GMMs

- **How are K-Means and GMMs similar?**
  - performance can depend on initial conditions (centroids, Gaussian parameters)
  - converge to local maximum
  - convergence when likelihood is small
  - likelihood grows with K, so can't magically find it
- **How are K-Means and GMMs dissimilar?**
  - GMM performs **soft** clustering, whilst K-Means performs **hard** clustering
  - K-Means clusters based on distance; GMM clusters based on covariance (K-Means and GMM will be similar if data is spherical)