

- The target features are not given in the training examples
- The aim is to construct a natural classification that can be used to predict features of the data.
- The examples are partitioned in into **clusters** or **classes**. Each class predicts feature values for the examples in the class.
 - ▶ In **hard clustering** each example is placed definitively in a class.
 - ▶ In **soft clustering** each example has a probability distribution over its class.
- Each clustering has a prediction error on the examples. The best clustering is the one that minimizes the error.

k -means algorithm

The k -means algorithm is used for hard clustering.

Inputs:

- training examples
- the number of classes, k

Outputs:

- a prediction of a value for each feature for each class
- an assignment of examples to classes

k -means algorithm formalized

- E is the set of all examples
- the input features are X_1, \dots, X_n
 $X_j(e)$ is the value of feature X_j for example e .
- there is a class for each integer $i \in \{1, \dots, k\}$.

The k -means algorithm outputs

- function $class : E \rightarrow \{1, \dots, k\}$.
 $class(e) = i$ means e is in class i .
- prediction $\hat{X}_j(i)$ for each feature X_j and class i .

The sum-of-squares error for $class$ and $\hat{X}_j(i)$ is

$$\sum_{e \in E} \sum_{j=1}^n \left(\hat{X}_j(class(e)) - X_j(e) \right)^2.$$

Aim: find $class$ and prediction function that minimize sum-of-squares error.

The sum-of-squares error for *class* and $\hat{X}_j(i)$ is

$$\sum_{e \in E} \sum_{j=1}^n \left(\hat{X}_j(\text{class}(e)) - X_j(e) \right)^2.$$

- Given *class*, the \hat{X}_j that minimizes the sum-of-squares error is the mean value of X_j for that class.
- Given \hat{X}_j for each j , each example can be assigned to the class that minimizes the error for that example.

k -means algorithm

Initially, randomly assign the examples to the classes.

Repeat the following two steps:

- For each class i and feature X_j , let

$$\hat{X}_j(i) \leftarrow \frac{\sum_{e: \text{class}(e)=i} X_j(e)}{|\{e : \text{class}(e) = i\}|}$$

(the prediction of class i on feature X_j)

- For each example e , assign e to the class i that minimizes

$$\sum_{j=1}^n \left(\hat{X}_j(i) - X_j(e) \right)^2.$$

until the second step does not change the assignment of any example.

Sufficient statistics:

- $cc[c]$ is the number of examples in class c ,
- $fs[j, c]$ is the sum of the values for $X_j(e)$ for examples in class c .

then define $pn(j, c)$, current estimate of $\hat{X}_j(c)$

$$pn(j, c) = fs[j, c]/cc[c]$$

$$class(e) = \arg \min_c \sum_{j=1}^n (pn(j, c) - X_j(e))^2$$

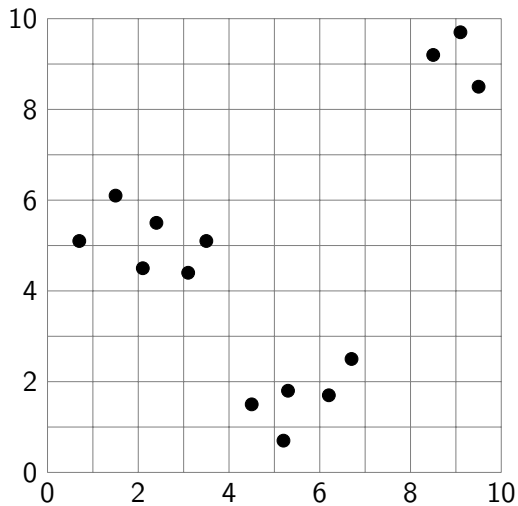
These can be updated in one pass through the training data.

```

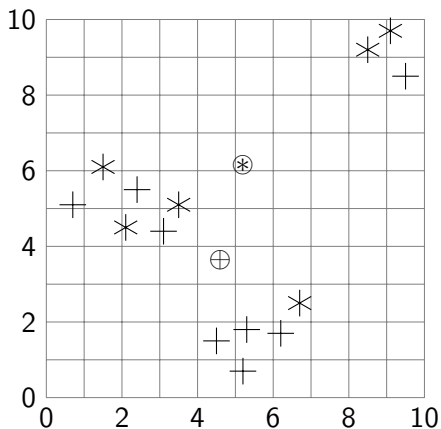
1: procedure k-means( $Xs, Es, k$ )
2:   Initialize fs and cc randomly (based on data)
3:   def pn( $j, c$ ) =  $fs[j, c] / cc[c]$ 
4:   def class( $e$ ) =  $\arg \min_c \sum_{j=1}^n (pn(j, c) - X_j(e))^2$ 
5:   repeat
6:     fsn and ccn initialized to be all zero
7:     for each example  $e \in Es$  do
8:        $c := class(e)$ 
9:        $ccn[c] + = 1$ 
10:      for each feature  $X_j \in Xs$  do
11:         $fsn[j, c] + = X_j(e)$ 
12:       $stable := (fsn=fs)$  and  $(ccn=cc)$ 
13:       $fs := fsn$ 
14:       $cc := ccn$ 
15:   until stable
16:   return class, pn

```

Example Data

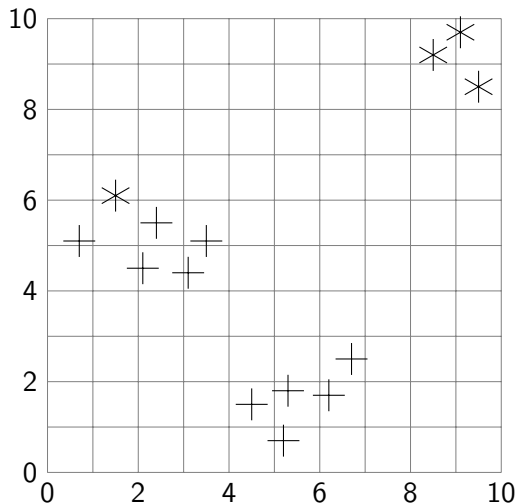


Random Assignment to Classes

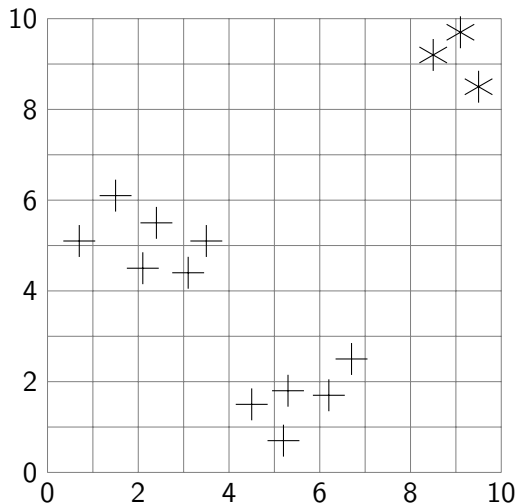


⊕ is mean of + and ⊗ is mean of *

Assign Each Example to Closest Mean



Reassign Each Example to Closest Mean

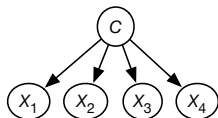


Properties of k -means

- An assignment of examples to classes is **stable** if running both the M step and the E step does not change the assignment.
- This algorithm will eventually converge to a stable local minimum.
- Any permutation of the labels of a stable assignment is also a stable assignment.
- It is not guaranteed to converge to a global minimum.
- It is sensitive to the relative scale of the dimensions.
- Increasing k *can* always decrease error (but does not always) until k is the number of different examples.
How? Given an assignment with k classes, for $k + 1$ classes start with the same assignment, but with the point most distant from its class center in its own new cluster.

- Used for soft clustering — examples are probabilistically in classes.
- k -valued random variable C

Model



Data

X_1	X_2	X_3	X_4
t	f	t	t
f	t	t	f
f	f	t	t
	...		

⇔ Probabilities

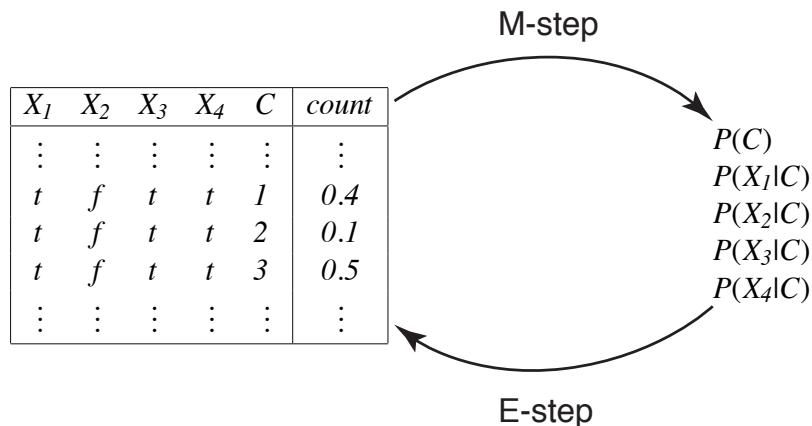
$$P(C)$$

$$P(X_1|C)$$

$$P(X_2|C)$$

$$P(X_3|C)$$

$$P(X_4|C)$$



- Repeat the following two steps:
 - ▶ **E-step** give the expected number of data points for the unobserved variables based on the given probability distribution.
 - ▶ **M-step** infer the (maximum likelihood or maximum a posteriori probability) probabilities from the data.
- Start either with made-up data or made-up probabilities.
- EM will converge to a local maxima.

Augmented Data — E step

Suppose $k = 3$, and $dom(C) = \{1, 2, 3\}$.

$$P(C = 1 | X_1 = t, X_2 = f, X_3 = t, X_4 = t) = 0.407$$

$$P(C = 2 | X_1 = t, X_2 = f, X_3 = t, X_4 = t) = 0.121$$

$$P(C = 3 | X_1 = t, X_2 = f, X_3 = t, X_4 = t) = 0.472:$$

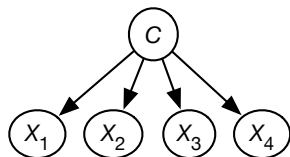
X_1	X_2	X_3	X_4	Count
\vdots	\vdots	\vdots	\vdots	\vdots
t	f	t	t	100
\vdots	\vdots	\vdots	\vdots	\vdots



$A[X_1, \dots, X_4, C]$

X_1	X_2	X_3	X_4	C	Count
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
t	f	t	t	1	40.7
t	f	t	t	2	12.1
t	f	t	t	3	47.2
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots

X_1	X_2	X_3	X_4	C	Count
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
t	f	t	t	1	40.7
t	f	t	t	2	12.1
t	f	t	t	3	47.2
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots



$$P(C=c)$$

$$P(X_i = v | C=c)$$

- cc , a k -valued array, $cc[c]$ is the sum of the counts for $class=c$.
- fc , a 3-dimensional array such that $fc[i, v, c]$, is the sum of the counts of the augmented examples t with $X_i(t) = val$ and $class(t) = c$.
- The probabilities can be computed by:

$$P(C=c) = \frac{cc[c]}{|Es|}$$

$$P(X_i = v | C=c) = \frac{fc[i, v, c]}{cc[c]}$$

```

1: procedure  $EM(Xs, Es, k)$ 
2:    $cc[c] := 0; fc[i, v, c] := 0$ 
3:   repeat
4:      $cc\_new[c] := 0; fc\_new[i, v, c] := 0$ 
5:     for each example  $\langle v_1, \dots, v_n \rangle \in Es$  do
6:       for each  $c \in [1, k]$  do
7:          $dc := P(C = c \mid X_1 = v_1, \dots, X_n = v_n)$ 
8:          $cc\_new[c] := cc\_new[c] + dc$ 
9:         for each  $i \in [1, n]$  do
10:           $fc\_new[i, v_i, c] := fc\_new[i, v_i, c] + dc$ 
11:         $stable := (cc \approx cc\_new) \text{ and } (fc \approx fc\_new)$ 
12:         $cc := cc\_new$ 
13:         $fc := fc\_new$ 
14:   until  $stable$ 
15:   return  $cc, fc$ 

```