# Introduction to Clustering

Mark Johnson

Department of Computing
Macquarie University

# Outline

# Data for supervised and unsupervised learning

- In both supervised and unsupervised learning, goal is to *map novel items to labels*
  - example 1: map names to their gender
  - example 2: map documents to their topic
  - example 3: maps words to their parts of speech
- The difference is the *kind of training data used*
  - in *supervised learning* the training data *contains the labels to be learnt*
    - example 1: input to supervised learner is
      `[('Adam','male'), ('Eve','female'), ...]`
  - in *unsupervised learning* the training data *does not contain the labels to be learnt*
    - example 1: input to unsupervised learner is
      `['Adam', 'Eve', ...]`

# Why is unsupervised learning important?

- Supervised learning requires *labelled training data*
- Manually labelling the training examples is often *expensive* and *slow*
  - $\Rightarrow$ often labelled training data sets are *very small*
- Unsupervised learning uses *unlabelled training data*, which is often *cheap* and *plentiful*
- There are *semi-supervised learning techniques* which can take as input a *labelled data set* and an *unlabelled data set*
  - usually the labelled data set is much smaller than the unlabelled data set
  - these typically build on unsupervised learning techniques

# The role of labels in unsupervised classification

- Training data for unsupervised classification *does not contain labels*
  - Name-gender example: *input* is `['Adam', 'Eve', 'Ida', ...]`

⇒ No way to learn the *names of the labels* (e.g., `'male','female'`)
  - but in e.g., *document clustering*, we can identify *key words* in documents in each cluster

⇒ Use *arbitrary identifiers* as labels (e.g., integers)
  - In name-gender example: *output* is `[0, 1, 1, ...]`

⇒ Since the unsupervised labels are arbitrary, *all that matters is whether two data items have the same label or different labels*

# Unsupervised classification as clustering

- In *unsupervised learning*, the learner associates unlabelled items with labels from an *arbitrary label set*
  - In name-gender example:
    *input* is `['Adam', 'Eve', 'Ida', 'Bill', ...]`
    and *output* is `[0, 1, 1, 0, ...]`
- Since the labels are *arbitrary*, all they do is *cluster items into groups*
- To convert a labelling into a clustering, *put all items with the same label into the same cluster*
  - items with label 0 form cluster `'Adam'`, `'Bill'`, ...
  - items with label 1 form cluster `'Eve'`, `'Ida'`, ...

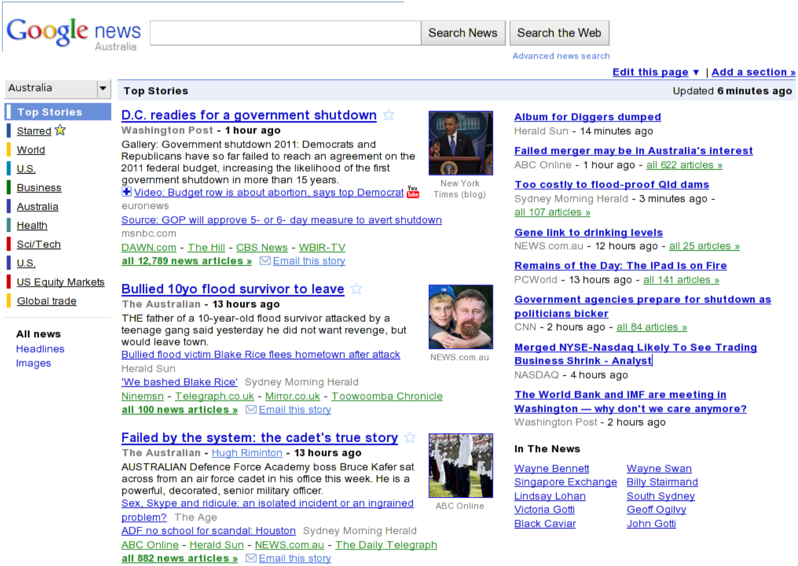
⇒ *Unsupervised classification is equivalent to clustering*

# Truth in advertising about machine learning

- In *supervised learning* the labels in the training data tell the learner what to look for
- In *unsupervised learning* the learner tries to group items that look similiar
  - ⇒ the *features* and *distance function* are *more important* than in supervised learning
- Unsupervised learning often returns *surprising results*
  - you might cluster names in the hope of automatically learning gender
  - but the clusterer might group them by ethnicity instead!
- Supervised learning works faily reliably if you have good data
  - usually most modern supervised classification algorithms have similiar performance
  - too many features doesn't hurt (so long as there are some informative features)
- Unsupervised learning is much more uncertain
  - different algorithms can produce very different results
  - choice of features is extremely important

# Outline

# Clustering news stories by topic

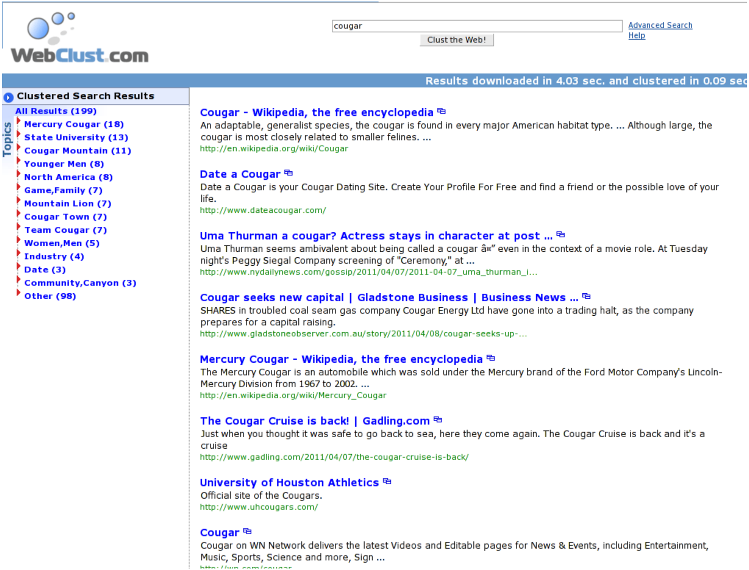



MACQUARIE UNIVERSITY

# Document clustering and keyword identification

- Document clustering identifies thematically-similiar documents in a document collection
  - news stories about the same topic in a collection of news stories
  - tweets on related topics from a twitter feed
  - scientific articles on related topics
- We can use *key-word identification methods* to identify the *most characteristic words in each cluster*
  - treat each cluster as a giant "meta-document"
    (i.e., append all of the documents in a document cluster together)
  - run Tf.Idf or similiar term-weighting program on the meta-documents to weight the words (and/or phrases) in the "meta-documents"
  - identify the words and/or phrases with the highest scores in each "meta-document"
  - use these high-scoring words and/or phrases as a label for the corresponding cluster

# Clustering in search




MACQUARIE UNIVERSITY

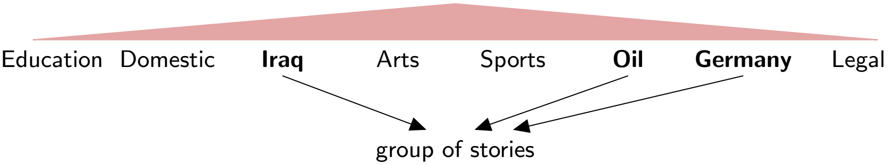
# Scatter-gather search

Education Domestic **Iraq** Arts Sports **Oil** **Germany** Legal

**Scatter**

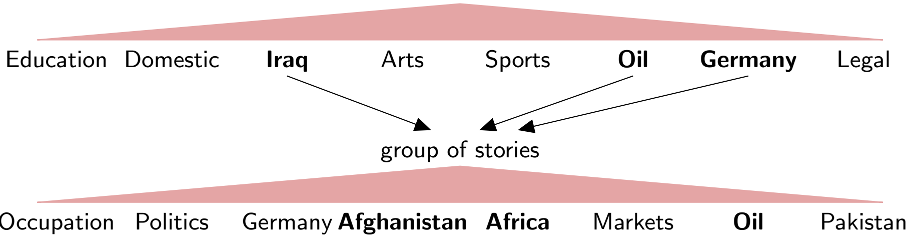
MACQUARIE UNIVERSITY

# Scatter-gather search




**Gather**

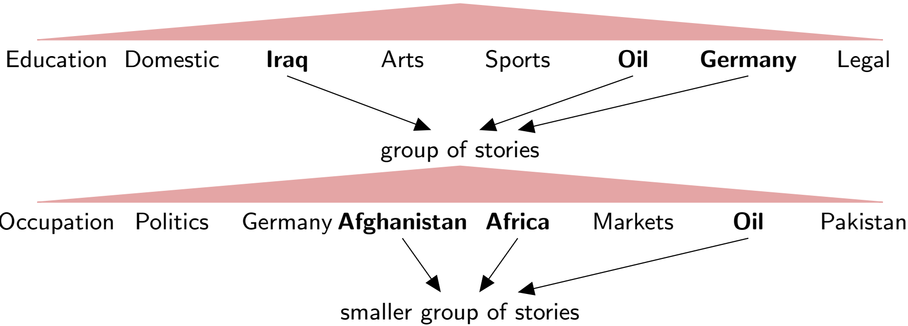
# Scatter-gather search




**Scatter**

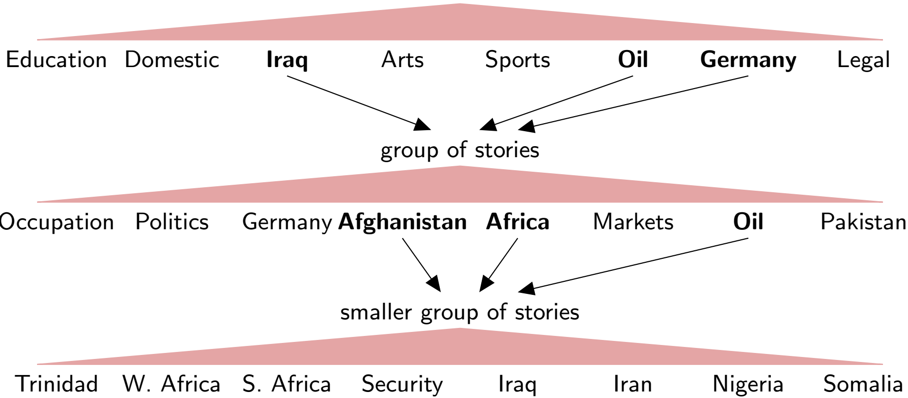
MACQUARIE UNIVERSITY

# Scatter-gather search




**Gather**

# Scatter-gather search




**Scatter**

# Outline

# Internal and external measures of clustering accuracy

- **Internal measures:** A clustering procedure should return a clustering where:
  - all data items in the same cluster are very similiar (i.e., "close" to each other)
  - if two data items come from different clusters, then the data items are different (i.e., "distant" from each other)
- **External measures:** Sometimes for evaluation we can obtain *labels* for (a subset of) the training data
  - labels are not available to clustering program (i.e., clustering program is *unsupervised*)
  - it's usually not reasonable to expect the clustering program to recover the labels
  - but the labels define a *clustering of the data items*
    - data items with the same label are assigned to the same cluster
  - so all we can really do is *compare the way that clustering program groups data items with the way the labels cluster data items*

# Confusion matrices

- *Confusion matrices* depict the relationship between two clusterings.
- Each cell shows the *number of items* in the *cross product* of the clusterings.
- They can sometimes help us understand just what a clustering has found.

| | c1 | c2 | c3 | c4 |
|---|---|---|---|---|
| science | 0 | 4 | 10 | 4 |
| romance | 10 | 0 | 1 | 0 |
| politics | 0 | 10 | 5 | 12 |
| news | 1 | 12 | 5 | 10 |

# Purity

- The *purity* of a clustering is the *fraction of data items assigned to the majority label of each cluster*.
- If $n$ is the number of data items and $\boldsymbol{C} = (C_1, \ldots, C_m)$ and $\boldsymbol{C}' = (C'_1, \ldots, C'_{m'})$ are two clusterings (partitions of the data items), then:

$$\text{purity}(\boldsymbol{C}, \boldsymbol{C}') = \frac{1}{n} \sum_{k=1}^{m} \max_{j=1:m'} |C_k \cap C'_j|$$

- In this example, purity $= 44/84 \approx 0.52$

| | c1 | c2 | c3 | c4 |
|---|---|---|---|---|
| science | 0 | 4 | 10 | 4 |
| romance | 10 | 0 | 1 | 0 |
| politics | 0 | 10 | 5 | 12 |
| news | 1 | 12 | 5 | 10 |

# The problem with purity

- *Purity* is the fraction of data items assigned to the majority label of each cluster
- If the clusters only contain a single data item, whatever label it happens to have will be the majority label of that cluster

$\Rightarrow$ In one-item clusters, the data item in that cluster will have the majority label

$\Rightarrow$ To produce a clustering algorithm that has *perfect purity* just *assign each data item to its own cluster*

  - the number of clusters is the number of data items

- Purity is a useful measure *if the number of clusters is fixed* and *much smaller than the number of data items*

# The Rand Index

- Developed by William Rand in 1971 to avoid problems with purity
- Central idea: given two data items $x, x' \in D$, a clustering $\boldsymbol{C}$ either *places them into the same cluster* or *places them into different clusters*
- Given two clusterings $\boldsymbol{C}$ and $\boldsymbol{C}'$, the Rand Index $R$ is *the number of pairs $x, x' \in D$ that are classified the same way by $\boldsymbol{C}$ and $\boldsymbol{C}'$ divided by the total number of pairs $x, x' \in D$*
  - if $a$ is the number of pairs $x, x' \in D$ that are in the *same cluster* in $\boldsymbol{C}$ and in the *same cluster* in $\boldsymbol{C}'$, and
  - if $b$ is the number of pairs $x, x' \in D$ that are in *different clusters* in $\boldsymbol{C}$ and in *different clusters* in $\boldsymbol{C}'$, and
  - $n' = n(n-1)/2$ is the *number of pairs* $x, x' \in D$, then:

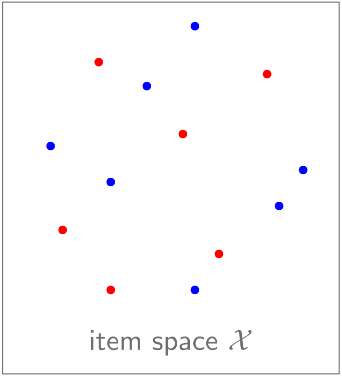
$$R = \frac{a+b}{n'}$$

# Outline

# Review: the k-nearest neighbour algorithm

The *k-nearest neighbour algorithm* for *supervised* classification:

To classify a data item $x$:

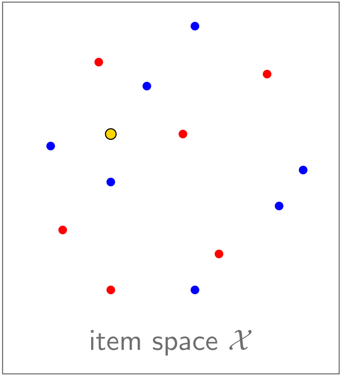



colour indicates label $\mathcal{Y}$

# Review: the k-nearest neighbour algorithm

The *k-nearest neighbour algorithm* for *supervised* classification:

To classify a data item $x$:



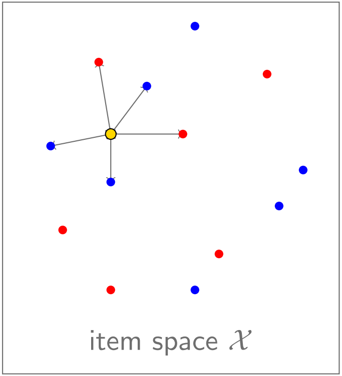

colour indicates label $\mathcal{Y}$

# Review: the k-nearest neighbour algorithm

The *k-nearest neighbour algorithm* for *supervised* classification:

To classify a data item $x$:

- set $N$ to the *$k$-nearest neighbours* of $x$ in $D$
  - the $k$-nearest neighbours of $x$ are the $k$ training items in $D$ with the *smallest $d(x, x')$ values*



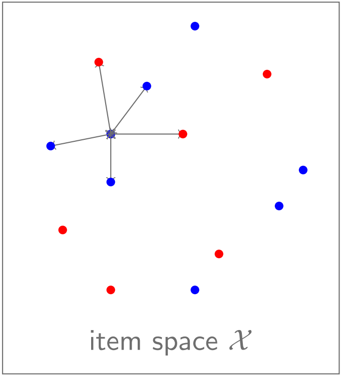

colour indicates label $\mathcal{Y}$

# Review: the k-nearest neighbour algorithm

The *k-nearest neighbour algorithm* for *supervised* classification:

To classify a data item $x$:

- set $N$ to the *$k$-nearest neighbours* of $x$ in $D$
  - the $k$-nearest neighbours of $x$ are the $k$ training items in $D$ with the *smallest $d(x, x')$ values*
- count how often each label $y'$ appears in $N$



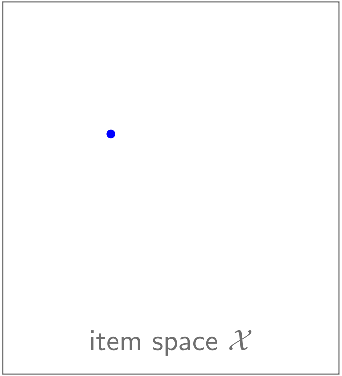

colour indicates label $\mathcal{Y}$

# Review: the k-nearest neighbour algorithm

The *k-nearest neighbour algorithm* for *supervised* classification:

To classify a data item $x$:

- set $N$ to the *k-nearest neighbours* of $x$ in $D$
  - the $k$-nearest neighbours of $x$ are the $k$ training items in $D$ with the *smallest* $d(x, x')$ *values*
- count how often each label $y'$ appears in $N$
- return *the most frequent label* $y$ in the $k$-nearest neighbours $N$ of $x$ as the predicted label for $x$




colour indicates label $\mathcal{Y}$

MACQUARIE UNIVERSITY

# The k-nearest neighbour algorithm

- Let $D$ be a *labelled training data set*:

$$D \quad = \quad ((x_1, y_1), \ldots, (x_n, y_n))$$

  where each item $x_i \in \mathcal{X}$ has label $y_i \in \mathcal{Y}$

- and let $d$ be *distance function*, where $d(x, x')$ is the distance between two items $x, x' \in \mathcal{X}$
- Given a novel item $x'$ to label, the *1-nearest neighbour label* $\widehat{y}(x')$ is:

$$\widehat{y}(x) \quad = \quad \underset{y \in \mathcal{Y}}{\operatorname{argmin}} \; \min_{x \in D_y} d(x, x')$$

  where for each $y \in \mathcal{Y}$, $D_y$ is the multiset of the training data items labelled $y$, i.e.,

$$D_y \quad = \quad \{x : (x, y) \in D\}$$

MACQUARIE UNIVERSITY

# The difference between min and argmin

- The min of a set of values is *the smallest value in a set*
  - If $S = \{\text{'Alpha'}, \text{'Beta'}, \text{'Gamma'}\}$
  - and $\texttt{len}(s)$ returns *the length of string s*
  - then

$$\min_{s \in S} \texttt{len}(s) = 4$$

- The argmin of a function is *the value that minimises that function*

$$\operatorname{argmin}_{s \in S} \texttt{len}(s) = \text{'Beta'}$$

- argmin can also be be used to find *the location of the smallest value in a sequence*
  - If $T = (\text{'Alpha'}, \text{'Beta'}, \text{'Gamma'})$ then:

$$\operatorname{argmin}_{i \in 1:3} \texttt{len}(T_i) = 2$$

- There are corresponding max and argmax functions as well

MACQUARIE UNIVERSITY

# Python code for min and argmin

- Python has `min` and `max` functions:

```
>>> xs = ('Alpha','Beta','Gamma')
>>> min(xs)
'Alpha'
```

  - with no arguments, `min` returns the smallest element in a sequence *with respect to Python's default ordering*

- Use *comprehensions* to compute more complex expressions

```
>>> xs = ('Alpha','Beta','Gamma')
>>> min(len(x) for x in xs)
4
```

  computes $\min_{s \in S} \texttt{len}(s)$ where $S = \{\text{'Alpha'}, \text{'Beta'}, \text{'Gamma'}\}$

- Use the `key` argument to specify a function to compute argmin

```
>>> xs = ('Alpha','Beta','Gamma')
>>> min(xs, key=len)
'Beta'
```

  computes $\operatorname{argmin}_{s \in S} \texttt{len}(s)$ where $S = \{\text{'Alpha'}, \text{'Beta'}, \text{'Gamma'}\}$

# More fun with min and max

- To find the *location* or *index* of the smallest item in a sequence, use `enumerate`

```
>>> xs = ('Gamma','Beta','Alpha')
>>> min(enumerate(xs), key=lambda ix: len(ix[1]))[0]
1
```

computes $\text{argmin}_{i \in 0:n-1} \, \texttt{len}(X_i)$
where $X = (\text{'Gamma'}, \text{'Beta'}, \text{'Alpha'})$

# Understanding argmin with enumerate

- `enumerate` generates pairs consisting of an index and an object

```
>>> xs = ('Alpha','Beta','Gamma')
>>> enumerate(xs)
<enumerate object at 0x7f536dd9d5f0>
>>> list(enumerate(xs))
[(0, 'Alpha'), (1, 'Beta'), (2, 'Gamma')]
```

- `lambda ix: len(ix[1])` maps a pair to *the length of its second element*

```
>>> (lambda ix: len(ix[1]))( (1,'Beta') )
4
```

- `min(enumerate(xs), key=lambda ix: ix[1])` returns the pair *whose second element minimises the* `len` *function*

```
>>> min(enumerate(xs), key=lambda ix: len(ix[1]))
(1, 'Beta')
```

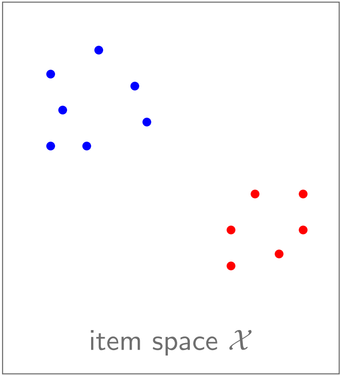
MACQUARIE UNIVERSITY

# Informal description of the k-means classification algorithm

- Note: *this is not a serious classification algorithm.*
  It is just a stepping stone to the k-means clustering algorithm.
- The **k-means classification algorithm**:
  - At training time (i.e., when you have the training data $D$, but before you see any test data)
    - Given training data $D$, let $D_y$ be subset of training data items with label $y$
    - For each label $y$, let $c_y$ be the *mean* or *centre* of $D_y$
  - To classify a new test item $x'$:
    - for each cluster mean $c_y$, compute $d_y =$ distance from $c_y$ to $x'$
    - return the $y$ that minimises $d_y$
      (i.e., the $y$ such that $x'$ is closest to $c_y$)
- This classifier might not be too bad *if the $D_y$ are well clustered*

# Graphical depiction of k-means classification algorithm

At training time:

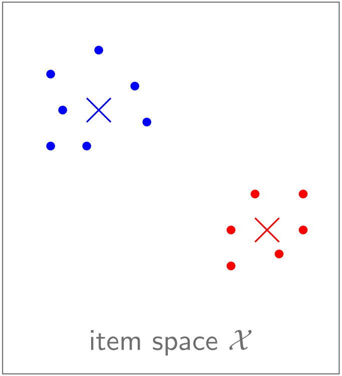



colour indicates label $\mathcal{Y}$

# Graphical depiction of k-means classification algorithm

At training time:

- compute cluster means $c_y$




colour indicates label $\mathcal{Y}$

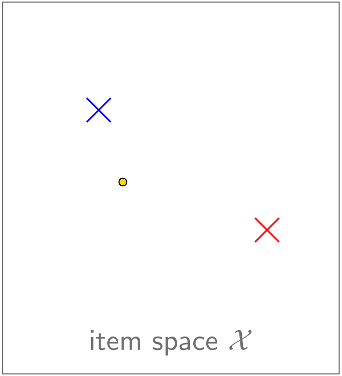
# Graphical depiction of k-means classification algorithm

At training time:

- compute cluster means $c_y$

At test time:

To classify a new data item $x'$:




colour indicates label $\mathcal{Y}$

# Graphical depiction of k-means classification algorithm

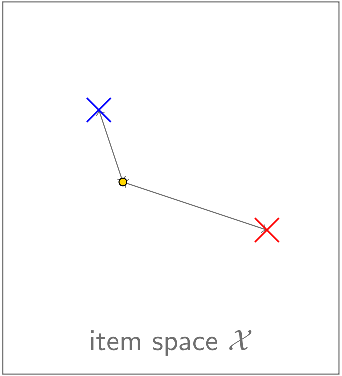
At training time:

- compute cluster means $c_y$

At test time:

To classify a new data item $x'$:

- compute the *distances* $d(c_y, x')$ from each cluster mean $c_y$ to $x'$




colour indicates label $\mathcal{Y}$

# Graphical depiction of k-means classification algorithm

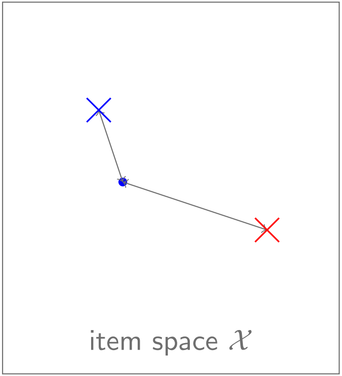
At training time:

- compute cluster means $c_y$

At test time:

To classify a new data item $x'$:

- compute the *distances* $d(c_y, x')$ from each cluster mean $c_y$ to $x'$
- return the label $y$ of the closest cluster mean $c_y$ to $x'$




colour indicates label $\mathcal{Y}$

# The k-means classification algorithm

- Let $D$ be a *labelled training data set*:

$$D = ((x_1, y_1), \ldots, (x_n, y_n))$$

  where each item $x_i \in \mathcal{X}$ has label $y_i \in \mathcal{Y}$ *and $\mathcal{X}$ is a real-valued vector space* (i.e., the features have numeric values), and let $d$ be a *distance function* as before
- For each $y \in \mathcal{Y}$, let $D_y = \{x : (x, y) \in D\}$
- For each $y \in \mathcal{Y}$, let $c_y$ be the *mean* or *centre* of $D_y$

$$c_y = \frac{1}{|D_y|} \sum_{x \in D_y} x$$

- Given a novel item $x' \in \mathcal{X}$ to label, the k-means classification algorithm returns:

$$\widehat{y}(x') = \operatorname*{argmin}_{y \in \mathcal{Y}} d(c_y, x')$$

MACQUARIE UNIVERSITY

# Review of set size and summation notation

- $|S|$ is *number of elements in the set $S$*
  - If $S = \{1, 3, 5, 7\}$ then $|S| = 4$
- If $S = \{v_1, \ldots, v_n\}$ is a set (or a sequence) of elements $v_i$ that can be added then:

$$\sum_{v \in S} v \quad = \quad v_1 + \ldots + v_n$$

  - If $S = \{1, 3, 5, 7\}$ then $\sum_{v \in S} v = 16$
  - If $S = \{(1, 2), (5, 4), (2, 2)\}$ then $\sum_{v \in S} v = (8, 8)$

MACQUARIE UNIVERSITY

# Set size and summation in Python

- `len` returns the size of a set
- `sum` returns the sum of the values of its arguments

```
>>> S = set([1,5,10,20])
>>> S
set([1, 10, 20, 5])
>>> len(S)
4
>>> sum(S)
36
```

MACQUARIE UNIVERSITY

# Summing vectors in Python

- Python doesn't directly support vector arithmetic
  - but specialised libraries like `numpy` do
- But it's easy to sum sequences of vectors

```
>>> vectors = [(1,5), (2,7), (4,9)]
>>> [sum(vector) for vector in zip(*vectors)]
[7, 21]
>>> zip(*vectors)
[(1, 2, 4), (5, 7, 9)]
```

# Using Python's Counter class to count

```
>>> import collections
>>> cntr = collections.Counter(['a','b','r','a'])
>>> cntr
Counter({'a': 2, 'r': 1, 'b': 1})

>>> cntr['b']
1

>>> cntr['O'] += 1
>>> cntr
Counter({'a': 2, 'O': 1, 'r': 1, 'b': 1})

>>> cntr.update(['E','E','N','I','E'])
>>> cntr
Counter({'E': 3, 'a': 2, 'b': 1, 'I': 1, 'N': 1, 'O': 1, 'r': 1})

>>> cntr.most_common(2)
[('E', 3), ('a', 2)]
```

# Outline

# Informal description of k-means clustering

The *k-means clustering algorithm* is an *iterative algorithm* that *reassigns data items to clusters at each iteration*

# Informal description of k-means clustering

The *k-means clustering algorithm* is an *iterative algorithm* that *reassigns data items to clusters at each iteration*

initialise the $k$ cluster centres $c_1, \ldots, c_k$ somehow

repeat until done:

  clear the clusters $C_1, \ldots, C_k$
  for each training data item $x$:

    find the closest cluster center $c_j$ to $x$
    add $x$ to cluster $C_j$

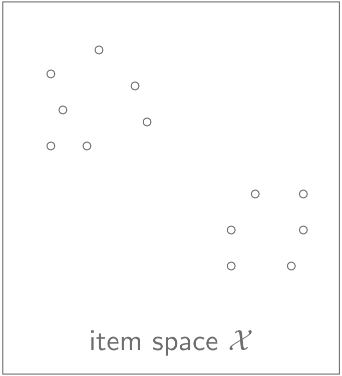
  for each cluster $C_j$:

    set $c_j$ to the *mean* or *centre* of cluster $C_j$

MACQUARIE UNIVERSITY

# Graphical depiction of k-means clustering algorithm

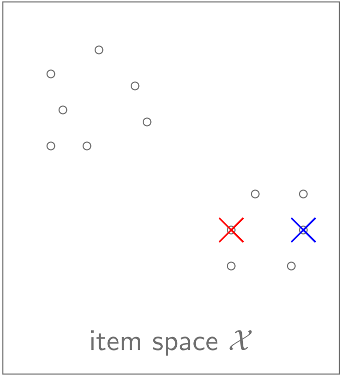
- Unlabelled training data




colours show clusters

# Graphical depiction of k-means clustering algorithm

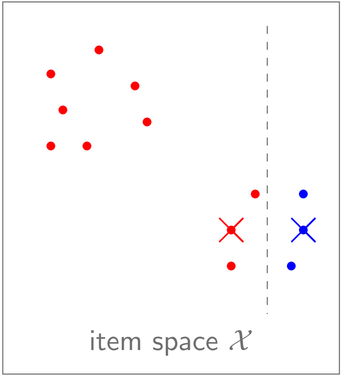
- Unlabelled training data
- Initialise cluster centers somehow




colours show clusters

# Graphical depiction of k-means clustering algorithm

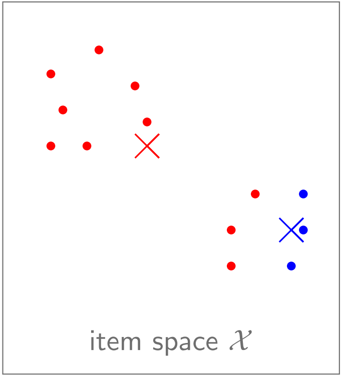
- Unlabelled training data
- Initialise cluster centers somehow
- Move each data item into closest cluster




colours show clusters

# Graphical depiction of k-means clustering algorithm

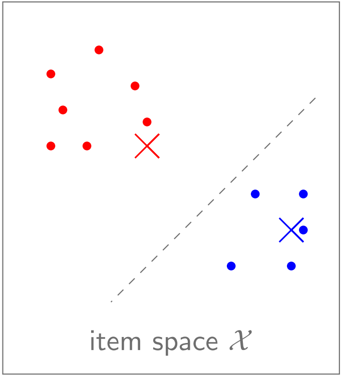
- Unlabelled training data
- Initialise cluster centers somehow
- Move each data item into closest cluster
- Move cluster centres to mean of data items in cluster




colours show clusters

# Graphical depiction of k-means clustering algorithm

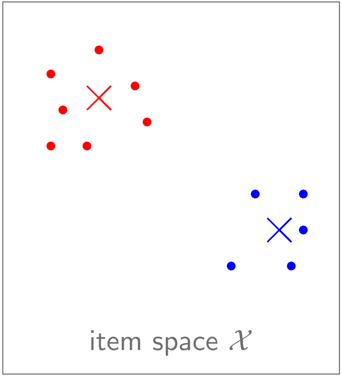
- Unlabelled training data
- Initialise cluster centers somehow
- Move each data item into closest cluster
- Move cluster centres to mean of data items in cluster
- Move each data item into closest cluster




colours show clusters

# Graphical depiction of k-means clustering algorithm

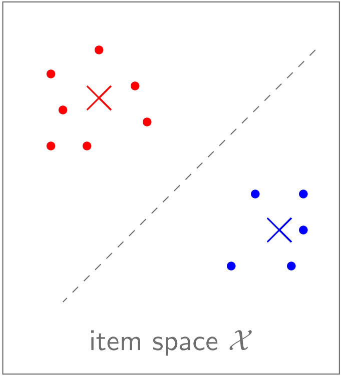
- Unlabelled training data
- Initialise cluster centers somehow
- Move each data item into closest cluster
- Move cluster centres to mean of data items in cluster
- Move each data item into closest cluster
- Move cluster centres to mean of data items in cluster




colours show clusters

# Graphical depiction of k-means clustering algorithm

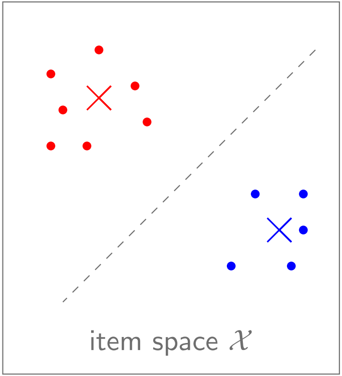
- Unlabelled training data
- Initialise cluster centers somehow
- Move each data item into closest cluster
- Move cluster centres to mean of data items in cluster
- Move each data item into closest cluster
- Move cluster centres to mean of data items in cluster
- Move each data item into closest cluster




colours show clusters

# Graphical depiction of k-means clustering algorithm

- Unlabelled training data
- Initialise cluster centers somehow
- Move each data item into closest cluster
- Move cluster centres to mean of data items in cluster
- Move each data item into closest cluster
- Move cluster centres to mean of data items in cluster
- Move each data item into closest cluster
- *No data items moved clusters, so we're done*




colours show clusters

# The k-means clustering algorithm

- Input to k-means clustering algorithm:
  - Unlabelled training data $D = (x_1, \ldots, x_n)$, where each $x_i \in \mathcal{X}$
  - a *distance function* $d$, where $d(x, x')$ is the distance between $x, x' \in \mathcal{X}$
  - the *number of clusters* $k$
- K-means clustering algorithm:

  Initialise cluster centres $c_j$, for $j = 1, \ldots, k$
  while not converged:
      $C_j = \emptyset$, for $j = 1, \ldots, k$
      for $i = 1, \ldots, n$:
          $j' = \operatorname{argmin}_{j \in 1, \ldots, k} d(c_j, x_i)$
          add $x_i$ to $C_{j'}$
      for $j = 1, \ldots, k$:
          set $c_j = \operatorname{mean}(C_j)$

# Initialising the k-means algorithm

- How the initial cluster centres are chosen makes a *big difference* to the clusters produced by the k-means algorithm
- There are many different initialisation strategies
- A simple and commonly-used strategy:
  - pick $k$ different items from the training data at random
  - initialise cluster centre $c_j$ to the $j$ randomly-chosen item
- Random initialisation $\Rightarrow$ *each run produces different clusters*
- Simple initialisation strategies (like this) can result in isolated *1-item clusters*
- Unfortunately even complicated initialisation strategies have draw-backs

# Determining convergence of the k-means algorithm

- Tracing the *the number of items moved from one cluster to another*, and the *intra-cluster distance S*

$$S = \sum_{i=1,\ldots,k} \sum_{x \in C_i} d(c_i, x)$$

  is a good way to monitor convergence.
  - these usually *drops quickly with the first few iterations*
  - and *change very slowly after that*
- Often after "enough" iterations *no data items are reassigned from one cluster to another*
  - $\Rightarrow$ further iterations will not change cluster assignments
  - $\Rightarrow$ *the algorithm has converged*
- Unfortunately the $k$-means algorithm only converges to a *local optimum*, which in general is not the *global optimum*

MACQUARIE UNIVERSITY

# Clustering as an optimisation problem

- The *intra-cluster distance* $S$ (distance from data items to their cluster centres) measures how well the cluster centres describe the data

$$S = \sum_{i=1,\ldots,k} \sum_{x \in C_i} d(c_i, x)$$

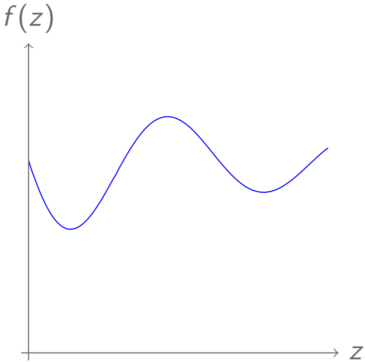
- The clusters $C_i$ are determined by the *cluster centers* $c_i$ and the data $D = (x_1, \ldots, x_n)$
- Goal of clustering: find cluster centres $\boldsymbol{c} = (c_1, \ldots, c_k)$ that minimise the intra-cluster distance

$$\begin{aligned} \widehat{\boldsymbol{c}} &= \operatorname*{argmin}_{\boldsymbol{c}} S \\ &= \operatorname*{argmin}_{\boldsymbol{c}} \sum_{i=1,\ldots,k} \sum_{x \in C_i} d(c_i, x) \end{aligned}$$

- The $k$-means algorithm is a way of finding cluster centres $\boldsymbol{c}$ that approximately minimise the the intra-cluster distance $S$

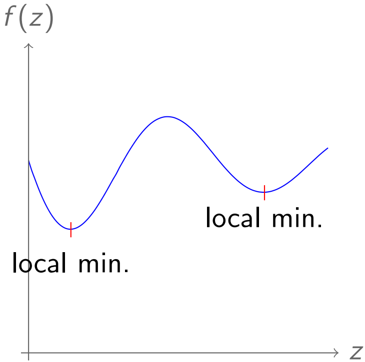
# Global and local optima in optimisation problems

- Machine learning algorithms like $k$-means involve solving an optimisation problem
  - these are usually *multi-dimensional*
  - the graph here only shows 1 dimension
- There can be several *local minima*
- But only one *global minimum*
- Iterative optimisation algorithms often are *attracted into the closest basin of attraction*



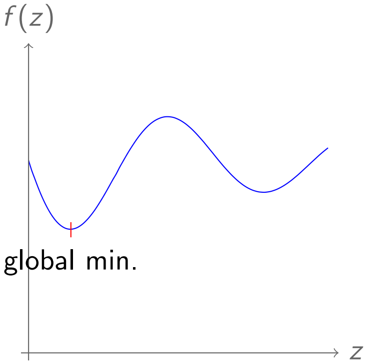
# Global and local optima in optimisation problems

- Machine learning algorithms like $k$-means involve solving an optimisation problem
  - these are usually *multi-dimensional*
  - the graph here only shows 1 dimension
- There can be several *local minima*
- But only one *global minimum*
- Iterative optimisation algorithms often are *attracted into the closest basin of attraction*



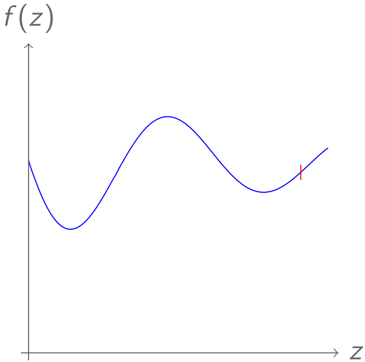
# Global and local optima in optimisation problems

- Machine learning algorithms like $k$-means involve solving an optimisation problem
  - these are usually *multi-dimensional*
  - the graph here only shows 1 dimension
- There can be several *local minima*
- But only one *global minimum*
- Iterative optimisation algorithms often are *attracted into the closest basin of attraction*



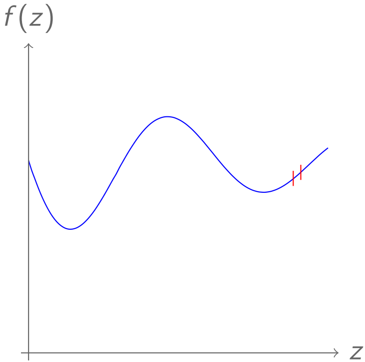
# Global and local optima in optimisation problems

- Machine learning algorithms like $k$-means involve solving an optimisation problem
  - these are usually *multi-dimensional*
  - the graph here only shows 1 dimension
- There can be several *local minima*
- But only one *global minimum*
- Iterative optimisation algorithms often are *attracted into the closest basin of attraction*



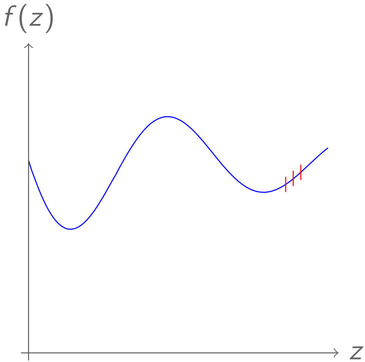
# Global and local optima in optimisation problems

- Machine learning algorithms like $k$-means involve solving an optimisation problem
  - these are usually *multi-dimensional*
  - the graph here only shows 1 dimension
- There can be several *local minima*
- But only one *global minimum*
- Iterative optimisation algorithms often are *attracted into the closest basin of attraction*



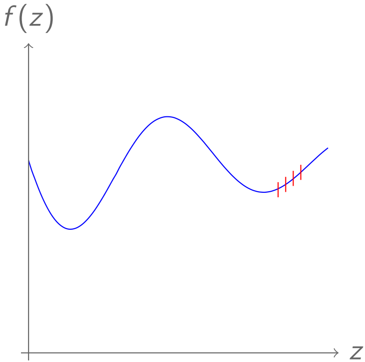
# Global and local optima in optimisation problems

- Machine learning algorithms like $k$-means involve solving an optimisation problem
  - these are usually *multi-dimensional*
  - the graph here only shows 1 dimension
- There can be several *local minima*
- But only one *global minimum*
- Iterative optimisation algorithms often are *attracted into the closest basin of attraction*



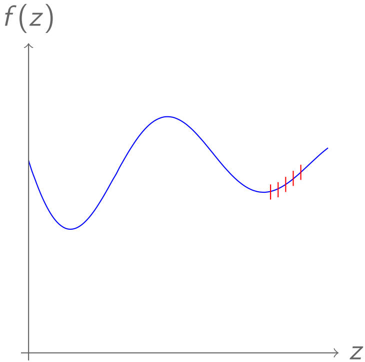
# Global and local optima in optimisation problems

- Machine learning algorithms like $k$-means involve solving an optimisation problem
  - these are usually *multi-dimensional*
  - the graph here only shows 1 dimension
- There can be several *local minima*
- But only one *global minimum*
- Iterative optimisation algorithms often are *attracted into the closest basin of attraction*



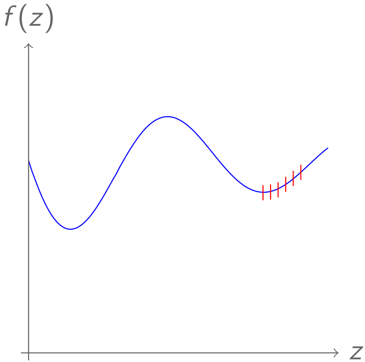
# Global and local optima in optimisation problems

- Machine learning algorithms like $k$-means involve solving an optimisation problem
  - these are usually *multi-dimensional*
  - the graph here only shows 1 dimension
- There can be several *local minima*
- But only one *global minimum*
- Iterative optimisation algorithms often are *attracted into the closest basin of attraction*



# Global and local optima in optimisation problems

- Machine learning algorithms like $k$-means involve solving an optimisation problem
  - these are usually *multi-dimensional*
  - the graph here only shows 1 dimension
- There can be several *local minima*
- But only one *global minimum*
- Iterative optimisation algorithms often are *attracted into the closest basin of attraction*



# K-means clustering in Python

```
class kmeans_clusters:

    def __init__(self, data, k, max_iterations, meanf, distf):
        self.data = data
        self.meanf = meanf
        self.distf = distf
        self.k = k
        self.initial_assignment_of_data_to_clusters()
        for iteration in xrange(max_iterations):
            self.compute_cluster_centres()
            nchanged = self.assign_data_to_closest_clusters()
            if nchanged == 0:
                break
```

MACQUARIE UNIVERSITY

# Initialising k-means clustering in Python

In class `kmeans_clusters`:

```
def initial_assignment_of_data_to_clusters(self):
    self.cluster_centres = random.sample(self.data, self.k)
    self.cids = [self.closest_cluster(item)
                 for item in self.data]

def closest_cluster(self, item):
    distances = [self.distf(centre, item)
                 for centre in self.cluster_centres]
    closest = min(xrange(self.k), key=lambda j:distances[j])
    return closest
```

- `self.cluster_centres` is a list of the $k$ cluster centers
- `self.cids` is a list of the "cluster ids" (integers that index into `self.cluser_centres`
- You'll also need an `import random` statement at the start of your code

MACQUARIE UNIVERSITY

# Computing cluster centres in Python

In class `kmeans_clusters`:

```
    def compute_cluster_centres(self):
        self.cluster_centres =
         [self.meanf([item
                      for item,cid in zip(self.data,self.cids)
                      if cid == id])
          for id in xrange(self.k)]
```

- This uses the user-supplied function `meanf` to compute the *mean* or the centre of a set of data items

MACQUARIE UNIVERSITY

# Updating the clusters in Python

In class `kmeans_clusters`:

```
def assign_data_to_closest_clusters(self):
    old_cids = self.cids
    self.cids = []
    nchanged = 0
    for i in xrange(len(self.data)):
        cid = self.closest_cluster(self.data[i])
        self.cids.append(cid)
        if cid != old_cids[i]:
            nchanged += 1
    return nchanged
```

# Outline



MACQUARIE UNIVERSITY

# Document clustering

- Input: a collection of documents
  - we'll use all *500 documents* in `nltk.corpus.brown`
  - these are classified into news, popular fiction, etc.
  - the *k*-means clusterer won't see these classes
  - ...but we'll use them to *evaluate* its output
- We need to provide:
  - a *distance function* and
  - a *mean function* that computes the centre of a document cluster
- We'll use a *bag of words representation* for each document
  - each document is represented by a dictionary mapping *words to their frequency counts*
  - for computational efficiency we'll only use a *subset of the vocabulary*

MACQUARIE UNIVERSITY

# Finding the keywords

```
import random, re
import nltk, nltk.corpus

word_rex = re.compile(r"^[A-Za-z]+$")

def compute_keywords(corpus, nwords):
    cntr = collections.Counter(w.lower()
                               for w in corpus.words()
                               if word_rex.match(w))
    return set(word for word,count in cntr.most_common(nwords))
```

- The clusters that the algorithm finds depend on which features it uses
- `compute_keywords(corpus, 1000)` returns a set containing the the 1,000th most frequent words
- We'll use these words as features

# Mapping fileids to word-frequencies

```
def fileid_featurevalues(corpus, fileid, keywords):
    cntr = collections.Counter(w.lower()
                               for w in corpus.words(fileid)
                               if word_rex.match(w) and
                               w.lower() in keywords)
    return cntr
```

- This returns a dictionary *mapping words to their frequencies* in document specified by `fileid`
- This is a *sparse representation* of word frequencies
  - words with zero frequency are *not present in the dictionary*

# Distance between two word-frequency dictionaries

```
def sparse_sum_squared_differences(key_val1, key_val2):
    keys = set(key_val1.iterkeys()) | set(key_val2.iterkeys())
    return sum(pow(key_val1.get(key,0) - key_val2.get(key,0), 2)
                   for key in keys)
```

- `key_val1` and `key_val2` are two dictionaries mapping words to their frequencies
    - they represent different documents
- `keys` is a set containing the union of their keys
- The result is the *sum of the square of the differences* in the frequencies

MACQUARIE UNIVERSITY

# Calculating the cluster means

```
def sparse_means(key_vals):
    keys = set()
    keys.update(*(key_val.iterkeys() for key_val in key_vals))
    n = len(key_vals)+1e-100
    key_meanval = {}
    for key in keys:
        key_meanval[key] = sum(key_val.get(key,0)
                               for key_val in key_vals)/n
    return key_meanval
```

- `key_vals` is a list of word-frequency dictionaries
- `keys` is a set containing *the union of the keys* in `key_vals`
- `n` is the number of word-frequency dictionaries in `key_vals`
- `key_meanval` is a dictionary mapping each key to its mean value

MACQUARIE UNIVERSITY

# Outline

# Numerical features in machine learning

- The $k$-means algorithms (and many other machine learning algorithms) require features to have *numerical values*
- In many applications, features naturally take *categorical values*
  - in name-gender application, the `'suffix1'` feature takes 1-letter values and the `'suffix2'` feature takes 2-letter values
    `gender_features('Christiana') = {'suffix1':'a', 'suffix2':'na'}`
- We'll *re-express these category-valued features as vectors of Boolean-valued features*
  - Boolean-valued features are numeric if we treat $\texttt{False} = 0$ and $\texttt{True} = 1$
- A feature $f$ can be viewed as a *function* from items $\mathcal{X}$ to feature values $\mathcal{V}$ (for Boolean features, $\mathcal{V} = \{0, 1\}$)
  - $\texttt{suffix1}(\texttt{'Cynthia'}) = \texttt{'a'}$

MACQUARIE UNIVERSITY

# Re-expressing a categorical feature using Boolean features

- Suppose a categorical feature $f$ ranges over values $\mathcal{V} = \{v_1, \ldots, v_m\}$
  - Example: `'suffix1'` ranges over $\{$`'a'`$, \ldots,$ `'z'`$\}$
- We re-express a categorical feature pair $f$ with a *vector $\boldsymbol{b} = (b_1, \ldots, b_m)$ of $m$ Boolean-valued features*
- If $x \in \mathcal{X}$ is a data item then:

$$f(x) = v_j \quad \Leftrightarrow \quad b_j(x) = 1$$

  - Example: If `'suffix1':'e'` is a feature-value pair for an item then:

$$\boldsymbol{b} \quad = \quad \begin{array}{ccccccc} ( \; 0, & \ldots, & 0, & 1, & 0, & \ldots, & 0 \; ) \\ \texttt{'a'} & & \texttt{'d'} & \texttt{'e'} & \texttt{'f'} & & \texttt{'z'} \end{array}$$

  - This is called a *one-hot encoding* of the feature
- Question: how are the `suffix2` features expressed as Boolean features?

MACQUARIE UNIVERSITY

# Multiple categorical features as Boolean features

- Each categorical feature can be represented as a vector of Boolean features
- To represent several categorical features, *concatenate the vectors of Boolean features that represent them*
- Example:

$$\boldsymbol{b} \quad = \quad (\underbrace{0, \ldots, 0, 1, 0, \ldots, 0}_{\texttt{'suffix1'}}, \underbrace{0, \ldots, 0, 1, 0, \ldots, 0}_{\texttt{'suffix2'}})$$

MACQUARIE UNIVERSITY

# Representations for sparse numerical features

- A set of features is *sparse* if *most features have value 0*
  - when categorical features are converted to binary features, all but one binary feature has value 0
  
  ⇒ the resulting binary feature vectors are *sparse*
- Representing very sparse feature vectors as as standard arrays *wastes space and time*
- Idea: only store features whose value is non-zero
- Represent sparse feature vectors as *a set of feature:value pairs for each feature that has a non-zero value*
- if a feature is not represented, its value is zero
- Example: `gender_features('Christiana') = {'suffix1=a':1, 'suffix2=na':1}`

# Outline

Supervised versus unsupervised learning

Applications of clustering in text processing

Evaluating clustering algorithms

Background for the *k*-means algorithm

The *k*-means clustering algorithm

Document clustering with *k*-means clustering

Numerical features in machine learning

Summary

# Summary

- Supervised versus unsupervised learning
  - unsupervised learning is generally far more challenging
- Unsupervised learning as clustering
- The $k$-means clustering algorithm
- Confusion matrices as ways of comparing two clusterings
- Evaluating clustering is difficult
  - cluster purity and its problems
  - the Rand index
- The difference between local and global optima, and the problems this causes for unsupervised learning