# CS4618: Artificial Intelligence I

# Clustering: Continued


### Derek Bridge
### School of Computer Science and Information Technology
### University College Cork


# Initialization

In [ ]:
```
%reload_ext autoreload
%autoreload 2
%matplotlib inline
```

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:
```
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.base import BaseEstimator, TransformerMixin

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler

from sklearn.cluster import AgglomerativeClustering

from sklearn.metrics import silhouette_score
from sklearn.metrics import adjusted_rand_score

from scipy.cluster.hierarchy import dendrogram

# Class, for use in pipelines, to select certain columns from a DataFram
e and convert to a numpy array
# From A. Geron: Hands-On Machine Learning with Scikit-Learn & TensorFlo
w, O'Reilly, 2017
# Modified by Derek Bridge to allow for casting in the same ways as pand
as.DataFrame.astype
class DataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names, dtype=None):
        self.attribute_names = attribute_names
        self.dtype = dtype
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        X_selected = X[self.attribute_names]
        if self.dtype:
            return X_selected.astype(self.dtype).values
        return X_selected.values

# Function to convert the output of scikit-learn's AgglomerativeClusteri
ng into the linkage matrix required by
# scipy's dendrogram function
# It takes in the model fit by AgglomerativeClustering, plus all the usu
al arguments of the dendrogram
# function: https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/
scipy.cluster.hierarchy.dendrogram.html
# Original by Mathew Kallada (BSD 3 licence), https://github.com/scikit-
learn/scikit-learn/pull/3464/files
# Original computes numbers of children incorrectly
# Fixed by Derek Bridge 2017
def plot_dendrogram(model, **kwargs):
    tree_as_list = model.children_
    sizes = {}
    linkage_array = []
    start_idx = len(tree_as_list) + 1
    idx = start_idx
    for children in tree_as_list:
        linkage = []
        size = 0
        for child in children:
            linkage += [child]
            if child < start_idx:
                size += 1
            else:
                size += sizes.get(child)
        linkage += [idx - start_idx + 1, size]
        sizes[idx] = size
        idx += 1
        linkage_array += [linkage]
    dendrogram(np.array(linkage_array).astype(float), **kwargs)
```

# Hierarchical Clustering

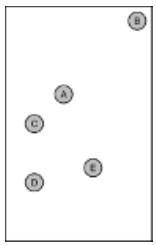
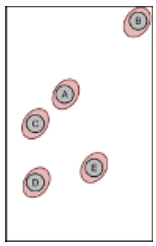
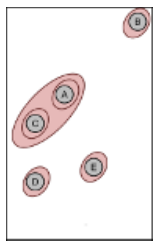
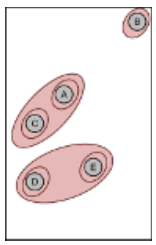
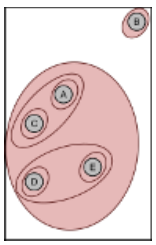
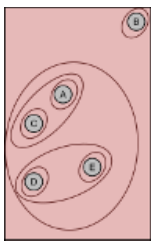
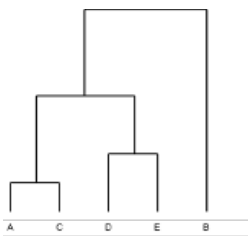
- We'll look at a bottom-up, agglomerative algorithm, simply called **Agglomerative Clustering**

# Agglomerative Clustering

AgglomerativeClustering($\boldsymbol{X}$)

- Create a cluster for each $\boldsymbol{x} \in \boldsymbol{X}$;
- while there is more than one cluster
  - choose two clusters;
  - merge the two clusters into one;
- return the tree of clusters;

# Toy Example

| Dataset $\boldsymbol{X}$ | Initial clusters | Merge | Merge | Merge | Merge | Tree |
|---|---|---|---|---|---|---|
|  |  |  |  |  |  |  |

# Agglomerative Clustering

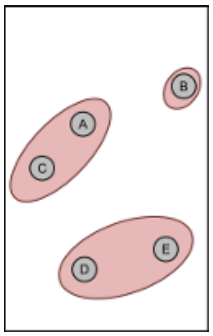
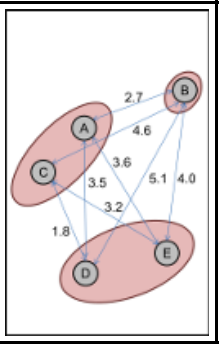
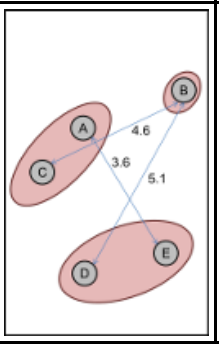
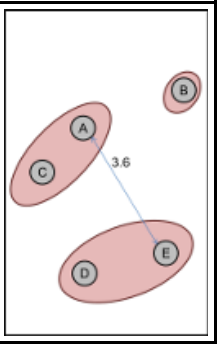
- How do we choose which to merge?
  - The pair of clusters for which the **linkage criterion** is *smallest*
- Linkage criteria (for two clusters $\boldsymbol{A}$ and $\boldsymbol{B}$)

| Name | Definition |
|---|---|
| **Complete-linkage clustering** | $\max \{ \operatorname{dist}(a, b) : a \in A,\ b \in B \}$ |
| **Average-linkage clustering** | $\frac{1}{\lvert A \rvert \lvert B \rvert} \sum_{a \in A} \sum_{b \in B} \operatorname{dist}(a, b)$ |

  (scikit-learn offers one more, which is called Ward's method, based on minimum variance, but it only works for Euclidean distance. We'll ignore it, and all the others that you can find in the literature!)
- We'll do an example using complete-linkage clustering and Euclidean distance

# Example of complete-linkage clustering

- In effect, complete-linkage chooses to merge the two clusters that have the smallest maximum distances

| Which to merge? | Distances | Max | Min |
|---|---|---|---|
|  |  |  |  |



# Comparison

- They often produce the same clustering, especially when clusters are compact and quite separate
- Complete-linkage (min of the maximum distances):
  - Sensitive to outliers
  - Some examples may be closer to other clusters than they are to the rest of their own cluster
- Average-linkage (min of the averages)
  - Since it calculates with the distances (instead of just comparing them), it is more sensitive to the quality of your distance measure and your use of scaling

# Agglomerative clustering in scikit-learn

- We'll cluster the zoo dataset: http://archive.ics.uci.edu/ml/datasets/zoo (http://archive.ics.uci.edu/ml/datasets/zoo)
- We explore the dataset using pandas and you'll see:
  - 99 examples (I discarded 2)
  - 18 columns but we only use 16
    - the first column is the animal name (we don't use it)
    - the last column is the result of manual clustering into 7 clusters (we don't use it for clustering!)
    - 15 of the rest are Boolean (already binarized), whether the animal has feathers, is a predator, etc.
    - the remaining column is the number of legs (min 0, max 8 here): we min-max scale it

In [ ]:
```
# Use pandas to read the CSV file into a DataFrame
df = pd.read_csv("datasets/dataset_zoo.csv")
```

In [ ]:
```
df.shape
```

In [ ]:
```
df.columns
```

In [ ]:
```
df.dtypes
```

In [ ]:
```
df.head(3)
```

In [ ]:
```
# The features we want to select
numeric_features = ["legs"]
nominal_features = ["hair", "feathers", "eggs", "milk", "airborne", "aqu
atic", "predator", "toothed",
                    "backbone", "breathes", "venomous", "fins", "tail",
"domestic", "catsize"]

# Create the pipelines
numeric_pipeline = Pipeline([
        ("selector", DataFrameSelector(numeric_features, "float64")),
        ("scaler", MinMaxScaler())
    ])

nominal_pipeline = Pipeline([
        ("selector", DataFrameSelector(nominal_features))
    ])

pipeline = Pipeline([("union",
                FeatureUnion([("numeric_pipeline", numeric_pipeline),
                              ("nominal_pipelines", nominal_pipeline)]))
])
```

In [ ]:
```
# Run the pipeline
pipeline.fit(df)
X = pipeline.transform(df)
```

In [ ]:
```
# Create the clustering object
agg_complete = AgglomerativeClustering(linkage="complete") # The alterna
tive is linkage="average"
```

In [ ]:
```
# Run it and store the result in a variable
clustering_complete = agg_complete.fit(X)
```

In [ ]:
```
# Draw the tree: not very informative!
fig = plt.figure()
plot_dendrogram(clustering_complete)
plt.show()
```

In [ ]:
```
# Draw the tree but using some extra parameters
# See https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.
cluster.hierarchy.dendrogram.html

names = df["name"] # the animal name from the original dataset

def llf(id):
    return names[id]

fig = plt.figure(figsize=(10,20))
plot_dendrogram(clustering_complete, orientation="left", leaf_label_func
=llf, leaf_font_size=10)
plt.show()
```

# Agglomerative Clustering: Discussion

- You don't have to run the algorithm to completion. You could exit early:
  - when you have a certain number of clusters, $k < m$
  - or when the next merge would result in a 'bad' cluster, using some measure such as max distance within a cluster
- This algorithm is only suitable for relatively small datasets
  - You would probably calculate the distance between every pair of objects in advance
  - But, in every iteration, it compares every cluster with every other
  - In the first iteration, that's $m^2$, then $(m-1)^2$, then $(m-2)^2$, … until we have just one cluster
  - That's $O(m^3)$
  - A smarter implementation only compares the new cluster with existing ones at each step, keeping track of the linkage distances between pairs in a priority-ordered queue, and taking $O(m^2 \log m)$
- Advanced: You can define 'connectivity constraints' which forbid the merging of certain examples with certain others
  - This allows you to incorporate domain-specific knowledge or preferences
  - E.g. only cluster together web pages that link to each other
  - It then often runs faster too

# Evaluating clustering, part 2

- We've seen one way to evaluate the quality of clustering
  - Using a score, such as Silhouette Score
- There is another way — if you have a 'ground truth':
  - I.e. if someone has already clustered the dataset manually

  Then we can compare the algorithm's clustering with the manual one
- In many cases, this would be absurd: if we have clustered manually, then we don't need an algorithm!
  - But a researcher who develops a new algorithm might test it against manual clustering
  - And, even in industry, sometimes we will have manually clustered a subset of the dataset, which we can use for evaluating the algorithm's clustering
- E.g. the **Adjusted Rand Score** compares two clusterings
  - For each pair $\boldsymbol{x}, \boldsymbol{x}'$ in $\boldsymbol{X}$, it counts up whether the two clusterings agree (they put $\boldsymbol{x}$ and $\boldsymbol{x}'$ into the same cluster) or not
  - It divides by the number of pairs
  - There is then an adjustment to 'correct for chance'
  - The adjusted rand score lies in $[-1, 1]$

In [ ]:
```
# Evaluate it
silhouette_score(X, agg_complete.labels_, metric='euclidean')
```

In [ ]:
```
# Interestingly, average-linkage is better
agg_average = AgglomerativeClustering(linkage="average")
clustering_average = agg_average.fit(X)
silhouette_score(X, agg_average.labels_, metric='euclidean')
```

In [ ]:
```
# Manual labels
ground_truth = df["label"]
```

- The manual clustering has 7 clusters (1-7)
- So, to be fair, we should get `AgglomerativeClustering` to do the same

In [ ]:
```
agg_complete = AgglomerativeClustering(linkage="complete", n_clusters=7)
clustering_complete = agg_complete.fit(X)

adjusted_rand_score(ground_truth, agg_complete.labels_)
```

In [ ]:
```
agg_average = AgglomerativeClustering(linkage="average", n_clusters=7)
clustering_average = agg_average.fit(X)

adjusted_rand_score(ground_truth, agg_average.labels_)
```

In [ ]:
```
# And, for curiosity, how much do they agree with each other
adjusted_rand_score(agg_complete.labels_, agg_average.labels_)
```

# Final Remarks on Clustering

- There are numerous other clustering algorithms
- There are numerous other ways of scoring clusterings to evaluate them
- But it's time for us to move on…