04_clustering_comparison

April 17, 2025

Load necessary libraries.

[1]:
```
### Basics
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import seaborn as sns
import os

### ML packages
from sklearn.cluster import KMeans, SpectralClustering, AgglomerativeClustering␣
↪#Hierarchial Clustering
from sklearn.mixture import GaussianMixture
import scipy.cluster.hierarchy as sch
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from collections import defaultdict
import umap

### Msc
import warnings

### OOP
from ml_utils import ClusterEvaluator
```

Load dataset.

[2]:
```
df = pd.read_csv('00_authors.csv').rename(columns = {'Unnamed: 0': 'Author'}).
↪drop(columns = 'BookID')
X = df.copy().drop(['Author'], axis=1)
X = X.to_numpy() # change pd.DataFrame to np.ndarray
authors = df['Author'].values  # n_samples-length array
```

We can also assign a hyperparameter $K = 4$. This represents the number of clusters (which we already know; there are 4 authors). Later on we will be hyperparameter tuning for some K based on stability!

[3]:
```
K_ = 4
```

Let us also define a dictionary to store our results.

[4]:
```
accuracy_dict = {}
```

1 Clustering Methods

1.1 Kmeans++

[5]:
```
kmeans = KMeans(n_clusters=K_, init='k-means++', n_init=10, max_iter=300) #␣
↪K-means++ initialization
kmeans.fit(X)
y_kmeans = kmeans.predict(X)
centers_pp = kmeans.cluster_centers_

accuracy, mapping = ClusterEvaluator(y_kmeans,df['Author'].values).accuracy
print(f'The mapping based on mode = {mapping}')
print(f'The accuracy of Kmeans++ = {accuracy}')
accuracy_dict['kmeans++'] = accuracy
```

```
The mapping based on mode = {0: 'Austen', 1: 'Shakespeare', 2: 'London', 3:
'London'}
The accuracy of Kmeans++ = 0.9096313912009513
```

1.2 Gaussian Mixture Models

[6]:
```
gmm = GaussianMixture(n_components=K_)
gmm.fit(X)
y_gmm = gmm.predict(X)

accuracy, mapping = ClusterEvaluator(y_gmm,df['Author'].values).accuracy
print(f'The mapping based on mode = {mapping}')
print(f'The accuracy of Kmeans++ = {accuracy}')
accuracy_dict['gmm'] = accuracy
```

```
The mapping based on mode = {0: 'Austen', 1: 'Shakespeare', 2: 'London', 3:
'Milton'}
The accuracy of Kmeans++ = 0.9512485136741974
```

1.3 Spectral Clustering

Spectral clustering already had a dimensional reduction by design, i.e., spectral clustering is essentially spectral embedding followed by kmeans! Therefore, we expect this 'raw' method to perform best compared to the other methods without any form of dimensionality reduction!

[7]:
```
spectral = SpectralClustering(n_clusters=K_, affinity='nearest_neighbors')
y_spectral = spectral.fit_predict(X)
```

```
accuracy, mapping = ClusterEvaluator(y_spectral,df['Author'].values).accuracy
print(f'The mapping based on mode = {mapping}')
print(f'The accuracy of Kmeans++ = {accuracy}')
accuracy_dict['spectral clustering'] = accuracy
```

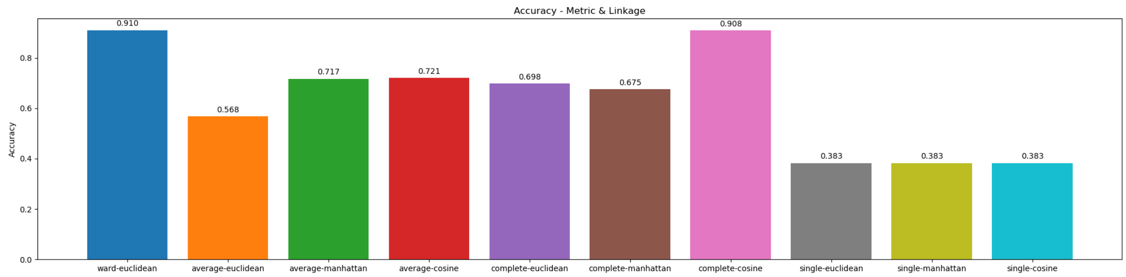
```
The mapping based on mode = {0: 'Shakespeare', 1: 'Austen', 2: 'London', 3:
'Milton'}
The accuracy of Kmeans++ = 0.9881093935790726
```

1.4 Hierarchial Clustering

[8]:
```
# Establish all possible combinations for linkage and metric!
methods = []
linkage_methods = ['ward', 'average', 'complete', 'single']
metrics = ['euclidean', 'manhattan', 'cosine']

for linkage in linkage_methods:
    for metric in metrics:
        if linkage == 'ward' and metric != 'euclidean':
            continue  # ward only supports euclidean
        methods.append((linkage, metric))

# Loop through all combinations and determine best accuracy!
hierarchical_accuracy_dict = {}
vals = df['Author'].values
for linkage, metric in methods:
    hierarchical = AgglomerativeClustering(n_clusters=K_, linkage=linkage,␣
 ↪metric=metric)
    y_hierarchical = hierarchical.fit_predict(X)  # Fit and predict in one step
    accuracy, mapping = ClusterEvaluator(y_hierarchical,vals).accuracy # Use␣
 ↪OOP code
    # print(f'The mapping based on mode = {mapping}')
    # print(f'The accuracy hierarchial clustering with {linkage} and {metric} =␣
 ↪{accuracy}')
    hierarchical_accuracy_dict[f'{linkage}-{metric}'] = accuracy
```

[9]:
```
keys = list(hierarchical_accuracy_dict.keys())
vals = list(hierarchical_accuracy_dict.values())
colors = plt.colormaps.get_cmap('tab10').colors

fig, ax = plt.subplots(figsize=(20, 5))
bars = ax.bar(keys, vals, color=colors)
ax.bar_label(bars, fmt='%.3f', padding=3)

ax.set_ylabel('Accuracy')
ax.set_title('Accuracy - Metric & Linkage')
plt.tight_layout()
```

```python
plt.savefig('Media/viz/04/04_hier_accuracy_across_metric_linkage')
plt.show()
```

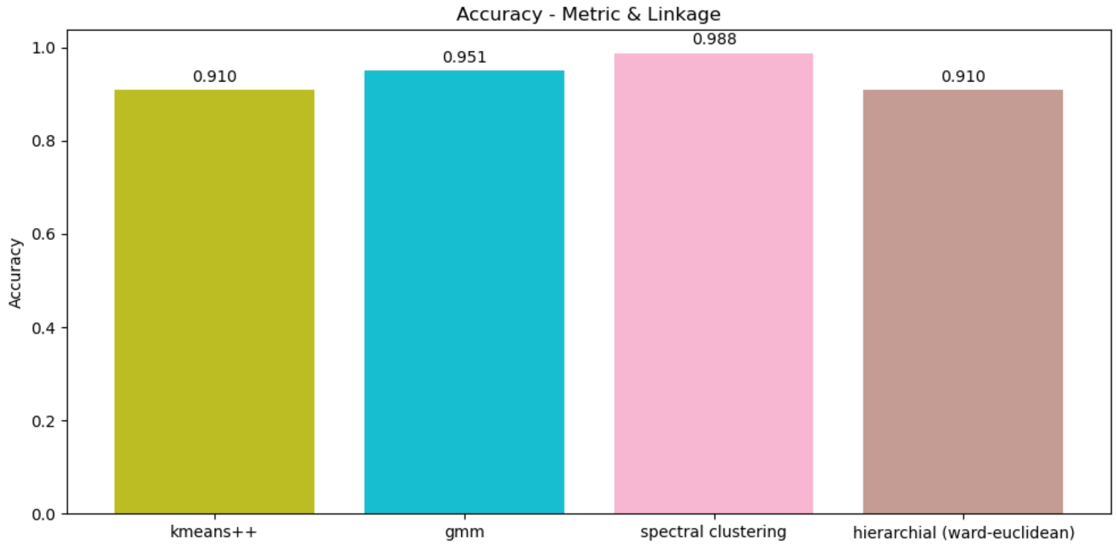


Lets store the highest accuracy params to compare across other methods - ward + euclidean.

[10]:
```python
accuracy_dict['hierarchial (ward-euclidean)'] = hierarchical_accuracy_dict['ward-euclidean']
```

1.5 Comparison

[11]:
```python
keys = list(accuracy_dict.keys())
vals = list(accuracy_dict.values())

colors = ['#bcbd22', '#17becf', '#f7b6d2', '#c49c94']

fig, ax = plt.subplots(figsize=(10, 5))
bars = ax.bar(keys, vals, color=colors)
ax.bar_label(bars, fmt='%.3f', padding=3)

ax.set_ylabel('Accuracy')
ax.set_title('Accuracy - Metric & Linkage')
plt.tight_layout()
plt.savefig('Media/viz/04/04_accuracy_across_methods')
plt.show()
```



Clearly, spectral clustering performs the best because it has a build in dimensional reduction! Therefore, now let us apply UMAP to all these methods and recompare.

2 Dimensional Reduction + Clustering Methods

Use UMAP as dimensionality reduction for all methods (except spectral clustering as this already uses spectral embedding so we will not repeat this).

[12]:
```
warnings.filterwarnings("ignore", category=UserWarning, module="umap") #␣
↪Suppress the specific UMAP warning on parralelism
umap_model = umap.UMAP(n_neighbors=10, min_dist=0.3)
X_umap = umap_model.fit_transform(X)
```

Store results in the following dictionary to compare across methods.

[13]:
```
umap_accuracy_dict = {}
umap_accuracy_dict['spectral clustering'] = accuracy_dict['spectral clustering']
```

2.1 Kmeans++

[14]:
```
# Run K-means++ on UMAP-reduced data
kmeans = KMeans(n_clusters=K_, init='k-means++', n_init=10, max_iter=300)
kmeans.fit(X_umap)
y_kmeans = kmeans.predict(X_umap)
centers_pp = kmeans.cluster_centers_

accuracy, mapping = ClusterEvaluator(y_kmeans,df['Author'].values).accuracy
print(f'The mapping based on mode = {mapping}')
print(f'The accuracy of Kmeans++ = {accuracy}')
```

```
umap_accuracy_dict['kmeans++'] = accuracy
```

```
The mapping based on mode = {0: 'Austen', 1: 'London', 2: 'Shakespeare', 3:
'Milton'}
The accuracy of Kmeans++ = 0.9916765755053508
```

2.2 Gaussian Mixture Models

[15]:
```
gmm = GaussianMixture(n_components=K_)
gmm.fit(X_umap)
y_gmm = gmm.predict(X_umap)

accuracy, mapping = ClusterEvaluator(y_gmm,df['Author'].values).accuracy
print(f'The mapping based on mode = {mapping}')
print(f'The accuracy of Kmeans++ = {accuracy}')
umap_accuracy_dict['gmm'] = accuracy
```

```
The mapping based on mode = {0: 'Shakespeare', 1: 'London', 2: 'Austen', 3:
'Milton'}
The accuracy of Kmeans++ = 0.9916765755053508
```

2.3 Hierarchial Clustering

[16]:
```
linkage = 'ward'
metric = 'euclidean'
hierarchical = AgglomerativeClustering(n_clusters=K_, linkage=linkage,␣
 ↪metric=metric)
y_hierarchical = hierarchical.fit_predict(X_umap)  # Fit and predict in one step
accuracy, mapping = ClusterEvaluator(y_hierarchical,df['Author'].values).
 ↪accuracy # Use OOP code
print(f'The mapping based on mode = {mapping}')
print(f'The accuracy hierarchial clustering with {linkage} and {metric} =␣
 ↪{accuracy}')
umap_accuracy_dict[f'hierarchial ({linkage}-{metric})'] = accuracy
```

```
The mapping based on mode = {0: 'Austen', 1: 'London', 2: 'Shakespeare', 3:
'Milton'}
The accuracy hierarchial clustering with ward and euclidean = 0.9916765755053508
```

2.4 Comparison

As we can see, all these methods (with the exception of spectral clustering as this did not use UMAP) converge to the same accuracy. This is because after applying UMAP we get very clearly defined clusters so there is not much/any room for these methods to diverge in their clustering assignments! Therefore they will yield the same accuracy; this makes sense!

[17]:
```
keys = list(umap_accuracy_dict.keys())
vals = list(umap_accuracy_dict.values())
```

```
colors = ['#9467bd', '#8c564b', '#e377c2', '#7f7f7f']

fig, ax = plt.subplots(figsize=(10, 5))
bars = ax.bar(keys, vals, color=colors)
ax.bar_label(bars, fmt='%.3f', padding=3)

ax.set_ylabel('Accuracy')
ax.set_title('Accuracy w/ UMAP - Metric & Linkage')
plt.tight_layout()
plt.savefig('Media/viz/04/04_accuracy_across_methods_with_umap')
plt.show()
```