Unsupervised Learning

UNSUPERVISED LEARNING IN PYTHON

Benjamin Wilson
Director of Research at lateral.io

Unsupervised learning

- Unsupervised learning finds patterns in data
- E.g., *clustering* customers by their purchases
- Compressing the data using purchase patterns (*dimension reduction*)

Supervised vs unsupervised learning

- *Supervised* learning finds patterns for a prediction task
- E.g., classify tumors as benign or cancerous (*labels*)
- Unsupervised learning finds patterns in data
- ... but *without* a specific prediction task in mind

Iris dataset

- Measurements of many iris plants
- Three species of iris:
  - *setosa*
  - *versicolor*
  - *virginica*
- Petal length, petal width, sepal length, sepal width (the *features* of the dataset)

[1] https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html

Arrays, features & samples

- 2D NumPy array
- Columns are measurements (the *features*)
- Rows represent iris plants (the *samples*)

Iris data is 4-dimensional

- Iris samples are points in 4 dimensional space
- Dimension = number of features
- Dimension too high to visualize!
- ... but unsupervised learning gives insight

k-means clustering

- Finds clusters of samples
- Number of clusters must be specified
- Implemented in `sklearn` ("scikit-learn")

```
print(samples)
```

```
[[ 5.   3.3  1.4  0.2]
 [ 5.   3.5  1.3  0.3]
 ...
 [ 7.2  3.2  6.   1.8]]
```

```
from sklearn.cluster import KMeans
model = KMeans(n_clusters=3)
model.fit(samples)
```

```
KMeans(n_clusters=3)
```

```
labels = model.predict(samples)
print(labels)
```

```
[0 0 1 1 0 1 2 1 0 1 ...]
```

Cluster labels for new samples

- New samples can be assigned to existing clusters
- k-means remembers the mean of each cluster (the "centroids")
- Finds the nearest centroid to each new sample

Cluster labels for new samples

```
print(new_samples)
```

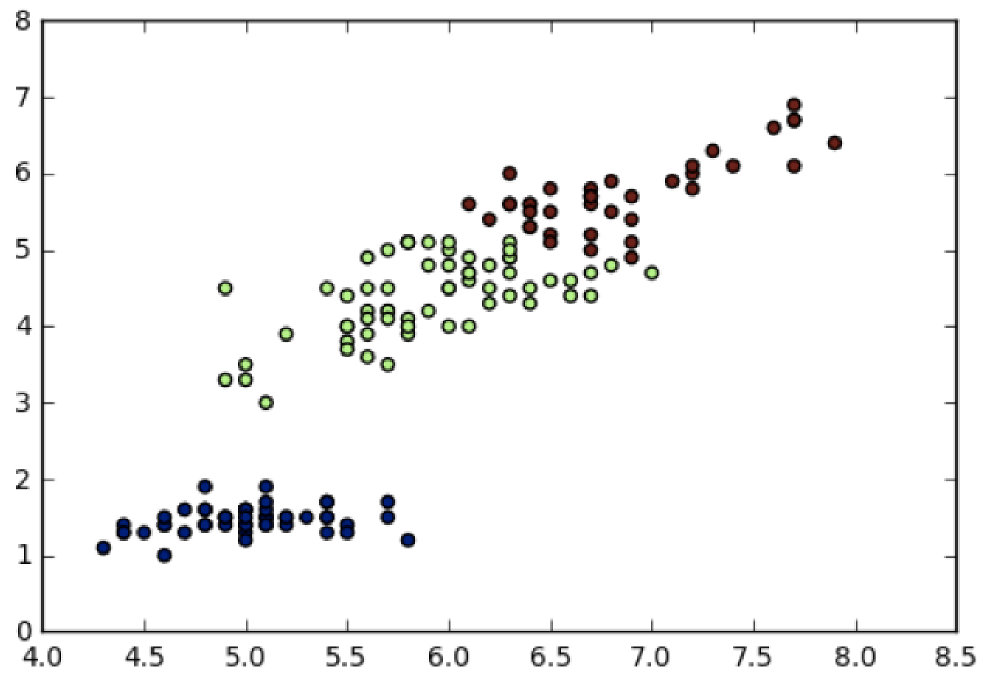
```
[[ 5.7  4.4  1.5  0.4]
 [ 6.5  3.   5.5  1.8]
 [ 5.8  2.7  5.1  1.9]]
```

```
new_labels = model.predict(new_samples)
print(new_labels)
```

```
[0 2 1]
```

Scatter plots

- Scatter plot of sepal length vs. petal length
- Each point represents an iris sample
- Color points by cluster labels
- PyPlot (`matplotlib.pyplot`)



Scatter plots

```python
import matplotlib.pyplot as plt
xs = samples[:,0]
ys = samples[:,2]
plt.scatter(xs, ys, c=labels)
plt.show()
```

Let's practice!

UNSUPERVISED LEARNING IN PYTHON

Evaluating a clustering

UNSUPERVISED LEARNING IN PYTHON

Benjamin Wilson
Director of Research at lateral.io

Evaluating a clustering

- Can check correspondence with e.g. iris species
- ... but what if there are no species to check against?
- Measure quality of a clustering
- Informs choice of how many clusters to look for

Iris: clusters vs species

- k-means found 3 clusters amongst the iris samples
- Do the clusters correspond to the species?

```
species  setosa  versicolor  virginica
labels
0             0           2         36
1            50           0          0
2             0          48         14
```

Cross tabulation with pandas

- Clusters vs species is a "cross-tabulation"
- Use the `pandas` library
- Given the species of each sample as a list `species`

```
print(species)
```

```
['setosa', 'setosa', 'versicolor', 'virginica', ... ]
```

Aligning labels and species

```python
import pandas as pd
df = pd.DataFrame({'labels': labels, 'species': species})
print(df)
```

```
     labels     species
0         1      setosa
1         1      setosa
2         2  versicolor
3         2   virginica
4         1      setosa
...
```

Crosstab of labels and species

```
ct = pd.crosstab(df['labels'], df['species'])
print(ct)
```

```
species   setosa  versicolor  virginica
labels
0              0           2         36
1             50           0          0
2              0          48         14
```

How to evaluate a clustering, if there were no species information?

Measuring clustering quality

- Using only samples and their cluster labels
- A good clustering has tight clusters
- Samples in each cluster bunched together

Inertia measures clustering quality

- Measures how spread out the clusters are (*lower* is better)
- Distance from each sample to centroid of its cluster
- After `fit()`, available as attribute `inertia_`
- k-means attempts to minimize the inertia when choosing clusters

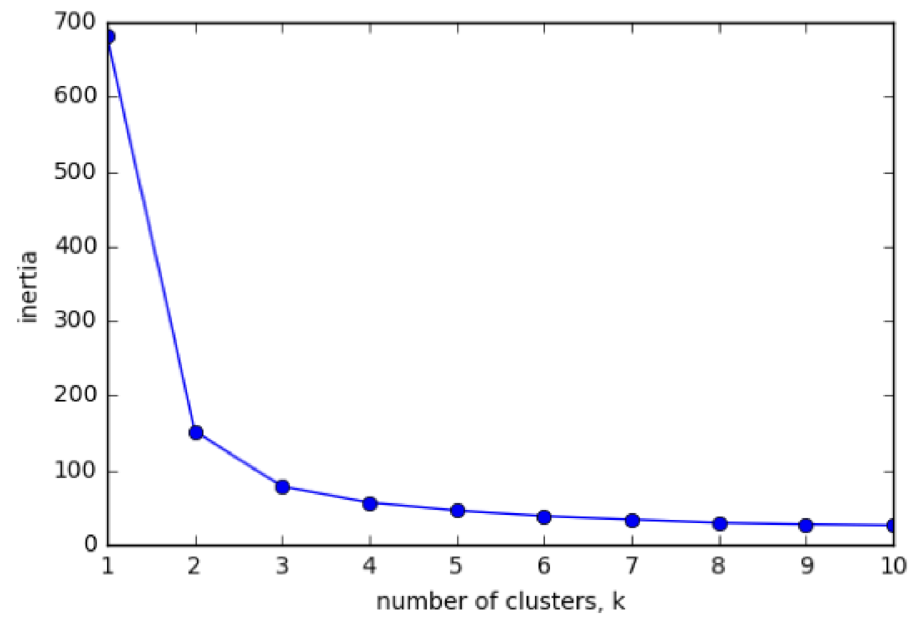
```python
from sklearn.cluster import KMeans

model = KMeans(n_clusters=3)
model.fit(samples)
print(model.inertia_)
```

```
78.9408414261
```

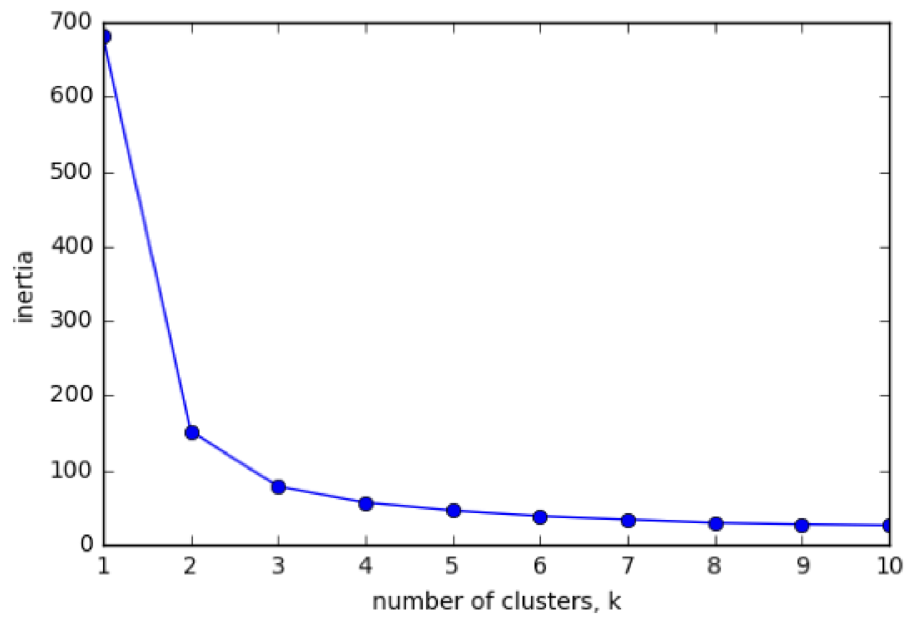
The number of clusters

- Clusterings of the iris dataset with different numbers of clusters
- More clusters means lower inertia
- What is the best number of clusters?



How many clusters to choose?

- A good clustering has tight clusters (so low inertia)
- ... but not too many clusters!
- Choose an "elbow" in the inertia plot
- Where inertia begins to decrease more slowly
- E.g., for iris dataset, 3 is a good choice



Let's practice!

Transforming features for better clusterings

UNSUPERVISED LEARNING IN PYTHON

Benjamin Wilson
Director of Research at lateral.io

Piedmont wines dataset

- 178 samples from 3 distinct varieties of red wine: Barolo, Grignolino and Barbera
- Features measure chemical composition e.g. alcohol content
- Visual properties like "color intensity"

[1] Source: https://archive.ics.uci.edu/ml/datasets/Wine

Clustering the wines

```python
from sklearn.cluster import KMeans
model = KMeans(n_clusters=3)
labels = model.fit_predict(samples)
```

Clusters vs. varieties

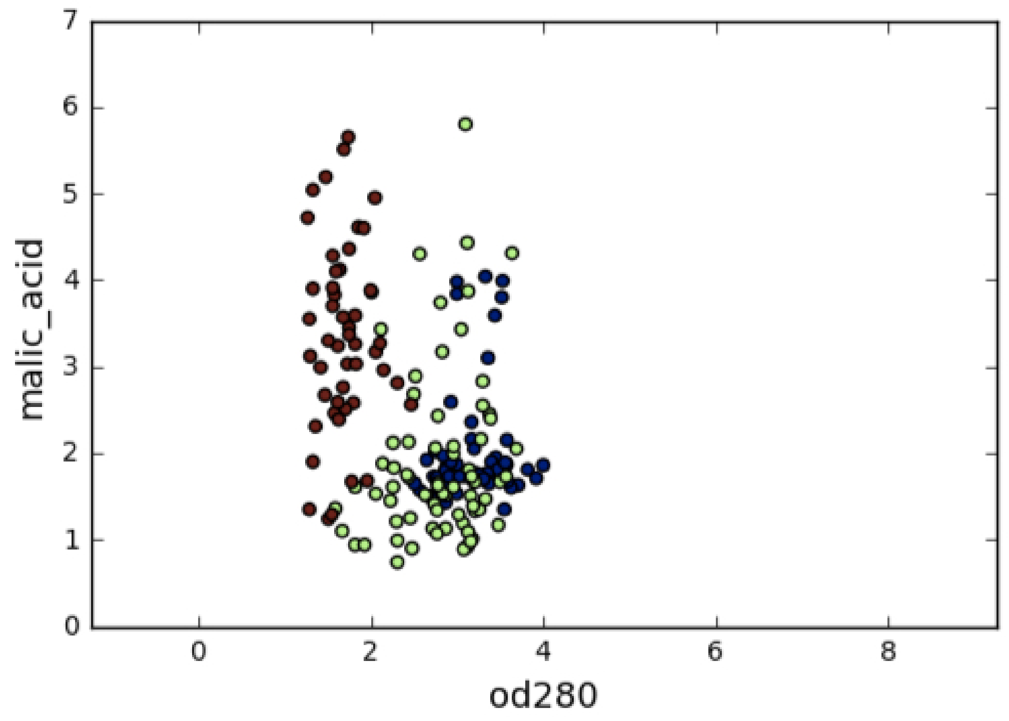
```
df = pd.DataFrame({'labels': labels,
                       'varieties': varieties})
ct = pd.crosstab(df['labels'], df['varieties'])
print(ct)
```

```
varieties  Barbera  Barolo  Grignolino
labels
0               29      13          20
1                0      46           1
2               19       0          50
```

Feature variances

- The wine features have very different variances!
- Variance of a feature measures spread of its values

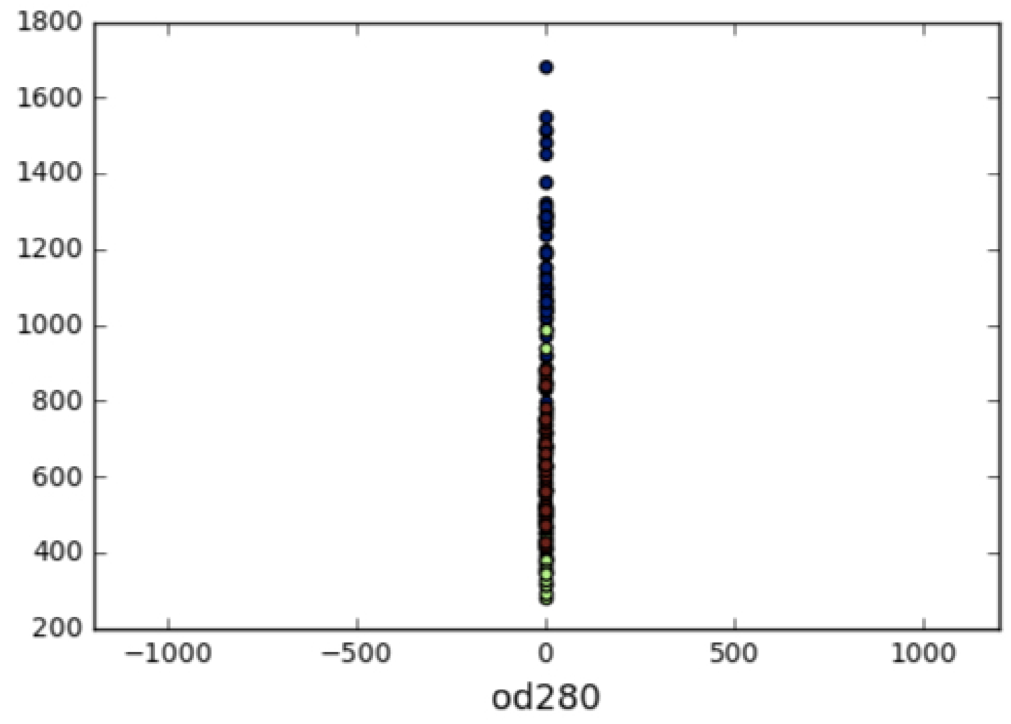
```
feature      variance
alcohol          0.65
malic_acid       1.24
...
od280            0.50
proline      99166.71
```



Feature variances

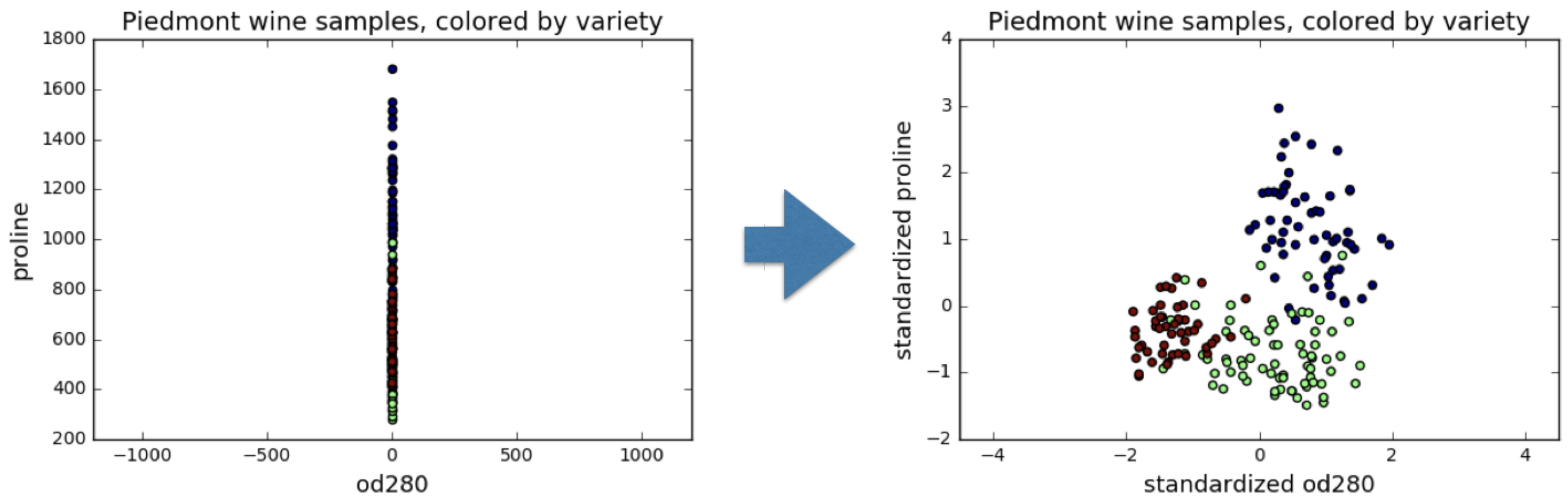
- The wine features have very different variances!
- Variance of a feature measures spread of its values

```
feature        variance
alcohol            0.65
malic_acid         1.24
...
od280              0.50
proline        99166.71
```



StandardScaler

- In kmeans: feature variance = feature influence
- `StandardScaler` transforms each feature to have mean 0 and variance 1
- Features are said to be "standardized"



sklearn StandardScaler

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(samples)
StandardScaler(copy=True, with_mean=True, with_std=True)
samples_scaled = scaler.transform(samples)
```

Similar methods

- `StandardScaler` and `KMeans` have similar methods
- Use `fit()` / `transform()` with `StandardScaler`
- Use `fit()` / `predict()` with `KMeans`

StandardScaler, then KMeans

- Need to perform two steps: `StandardScaler` , then `KMeans`
- Use `sklearn` pipeline to combine multiple steps
- Data flows from one step into the next

Pipelines combine multiple steps

```python
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
scaler = StandardScaler()
kmeans = KMeans(n_clusters=3)
from sklearn.pipeline import make_pipeline
pipeline = make_pipeline(scaler, kmeans)
pipeline.fit(samples)
```

```
Pipeline(steps=...)
```

```python
labels = pipeline.predict(samples)
```

Feature standardization improves clustering

With feature standardization:

```
varieties  Barbera  Barolo  Grignolino
labels
0                0      59           3
1               48       0           3
2                0       0          65
```

Without feature standardization was very bad:

```
varieties  Barbera  Barolo  Grignolino
labels
0               29      13          20
1                0      46           1
2               19       0          50
```

sklearn preprocessing steps

- `StandardScaler` is a "preprocessing" step
- `MaxAbsScaler` and `Normalizer` are other examples

Let's practice!

UNSUPERVISED LEARNING IN PYTHON