Unsupervised Learning: Clustering
Introduction and Simple K-means

Some material adapted from slides by Andrew Moore, CMU
Yann LeCun on Unsupervised Learning

“Most of human and animal learning is *unsupervised learning*. If intelligence was a cake, unsupervised learning would be the cake, *supervised learning* would be the icing on the cake, and *reinforcement learning* would be the cherry on the cake. ... We know how to make the icing and the cherry, but we don't know how to make the cake. We need to solve the unsupervised learning problem before we can even think of getting to true AI.”*
Unsupervised Learning

• Supervised learning used labeled data pairs \((x, y)\) to learn a function \(f : X \rightarrow y\)
• What if we don’t have labels?
• No labels = unsupervised learning
• Only some points are labeled = semi-supervised learning
  – Getting labels is expensive, so we only get a few
• Clustering is the unsupervised grouping of data points based on similarity
• It can be used for knowledge discovery
Clustering algorithms

• Many clustering algorithms
• Clustering typically done using a distance measure defined between instances or points
• Distance defined by instance feature space, so it works with numeric features
  – Requires encoding of categorial values; may benefit from normalization
• We’ll look at three popular approaches
  1. Centroid-based clustering (e.g., Kmeans)
  2. Hierarchical clustering
  3. DBSCAN
Clustering Data

Given a collection of points \((x, y)\), group them into one or more clusters based on their distance from one another.

How many clusters are there?

How can we find them?
(1) K-Means Clustering

- Randomly choose k cluster center locations, aka centroids
- Loop until convergence
  - assign a point to cluster of closest centroid
  - re-position cluster centroids based on its data assigned
- Convergence: no point is re-assigned to a different cluster

\[ k = 5 \]
k-MEANS CLUSTERING

1. k centerpoints are randomly initialized.
2. Observations are assigned to the closest centerpoint.
3. Centerpoints are moved to the center of their members.
4. Repeat steps 2 and 3 until no observation changes membership in step 2.

Chris Albon
distance, centroids

• Distance between points \((X_0,Y_0,Z_0)\) and \((X_1,Y_1,Z_1)\) is just
  \(\sqrt{(X_0 - X_1)^2 + (Y_0 - Y_1)^2 + (Z_0 - Z_1)^2}\)

• In numpy

  >>> import numpy as np
  >>> p1 = np.array([0,-2,0,1]) ; p2 = np.array([0,1,2,1])
  >>> np.linalg.norm(p1 - p2)
  3.605551275463989

• Computing centroid of set of points easy

  >>> points = np.array([[1,2,3], [2,1,1], [3,1,0]])  # 3D points
  >>> centroid = np.mean(points, axis=0)  # mean across columns
  >>> centroid
  array([2. , 1.33, 1.33])
(1) K-Means Clustering

• Randomly choose k cluster center locations, aka centroids
• Loop until convergence
  - assign a point to cluster of the closest centroid
  - re-estimate cluster centroids based on its data assigned
• Convergence: no point is assigned to a different cluster
K-Means Clustering

K-Means \((k, \text{data})\)

- Randomly choose \(k\) cluster center locations (centroids)
- **Loop until convergence**
  - Assign each point to the cluster of closest centroid
  - Re-estimate cluster centroids based on data assigned to each
- **Convergence:** no point is assigned to a different cluster

**veroni diagram:** add lines for regions of points closest to each centroid
K-Means Clustering

K-Means \((k, \text{ data})\)

- Randomly choose \(k\) cluster center locations (centroids)
- **Loop until convergence**
  - Assign each point to the cluster of closest centroid
  - **Re-estimate cluster centroids based on data assigned to each**
- **Convergence**: no point is assigned to a different cluster
K-Means Clustering

K-Means \((k, \text{data})\)

- Randomly choose \(k\) cluster center locations (centroids)
- Loop until convergence
  - Assign each point to the cluster of closest centroid
  - Re-estimate cluster centroids based on data assigned to each
- **Convergence:** no point is assigned to a different cluster
Visualizing k-means:
Clustering the Iris Data

• Let’s try using unsupervised clustering on the Iris Data
Cluster mode

- Use training set
- Supplied test set: Set...
- Percentage split: % 66
- Classes to clusters evaluation
- Store clusters for visualization

Clusterer output

Within cluster Sum of Squared errors: 7401.745438325893

Initial starting points (random):
Cluster 0: 6.1, 2, 9, 4.7, 1.4, Iris-versicolor
Cluster 1: 6.2, 2, 9, 4.3, 1.3, Iris-versicolor
Cluster 2: 6.9, 3.1, 5.1, 2.3, Iris-virginica

Missing values globally replaced with mean/mode

Final cluster centroids:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Full Data (150.0)</th>
<th>Cluster 0 (50.0)</th>
<th>Cluster 1 (50.0)</th>
<th>Cluster 2 (50.0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sepalwidth</td>
<td>5.8433</td>
<td>5.936</td>
<td>5.006</td>
<td>6.588</td>
</tr>
<tr>
<td>sepalwidth</td>
<td>3.054</td>
<td>2.77</td>
<td>3.418</td>
<td>2.974</td>
</tr>
<tr>
<td>petalwidth</td>
<td>3.7587</td>
<td>4.26</td>
<td>1.464</td>
<td>5.552</td>
</tr>
<tr>
<td>petalwidth</td>
<td>1.1987</td>
<td>1.326</td>
<td>0.244</td>
<td>2.026</td>
</tr>
<tr>
<td>class</td>
<td>Iris-setosa</td>
<td>Iris-versicolor</td>
<td>Iris-setosa</td>
<td>Iris-virginica</td>
</tr>
</tbody>
</table>

Time taken to build model (full training data): 0 seconds

=== Model and evaluation on training set ===

Clustered Instances

<table>
<thead>
<tr>
<th>Clusters</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
</tr>
</tbody>
</table>
Perfect results, but we forgot to remove ground truth nominal attribute!
Select “Classes to cluster evaluation” to identify that class.

Getting results that are too good is usually a red flag
Cluster mode:
- Use training set
- Supplied test set: Set...
- Percentage split: 66%
- Classes to clusters evaluation: (Nom) class
- Store clusters for visualization

Clusterer output:

```
sepal length  5.8433  5.8885  5.006  6.8462
sepal width   3.054    2.7377  3.418  3.0021
petal length  3.7587   4.3967  1.464  5.7026
petal width   1.1987   1.418   0.244  2.0795
```

Time taken to build model (full training data): 0 seconds

--- Model and evaluation on training set ---

Clustered Instances:

<table>
<thead>
<tr>
<th></th>
<th>61 (41%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50 (33%)</td>
</tr>
<tr>
<td>2</td>
<td>39 (26%)</td>
</tr>
</tbody>
</table>

Class attribute: class
Classes to Clusters:

```
0 1 2  \--- assigned to cluster
0 50 0  | Iris-setosa
47 0 3  | Iris-versicolor
14 0 36 | Iris-virginica

Cluster 0 \--- Iris-versicolor
Cluster 1 \--- Iris-setosa
Cluster 2 \--- Iris-virginica

Incorrectly clustered instances : 17.0 11.3333 %
```
2.3. Clustering

Clustering of unlabeled data can be performed with the module `sklearn.cluster`.

Each clustering algorithm comes in two variants: a class, that implements the `fit` method to learn the clusters on train data, and a function, that, given train data, returns an array of integer labels corresponding to the different clusters. For the class, the labels over the training data can be found in the `labels_` attribute.

Input data

One important thing to note is that the algorithms implemented in this module can take different kinds of matrix as input. All the methods accept standard data matrices of shape `[n_samples, n_features]`. These can be obtained from the classes in the `sklearn.feature_extraction` module. For `AffinityPropagation`, `SpectralClustering` and `DBSCAN` one can also input similarity matrices of shape `[n_samples, n_samples]`. These can be obtained from the functions in the `sklearn.metrics.pairwise` module.

2.3.1. Overview of clustering methods
sklearn K-means clustering on Fisher's Iris dataset

[14] %matplotlib inline
from sklearn import cluster, datasets, metrics
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

[15] # load training data
iris = sklearn.datasets.load_iris()

The Iris dataset has 150 instances:
- X floats for Sepal Length, Sepal Width, Petal Length and Petal Width *y: integer (0,1,2) representing species (Setosa, Versicolour, Virginica)

[20] X = iris.data
y = iris.target

[25] # show first three rows of training instance data and the target classes
print(X[:3])
print(y[:3])

[27] # show all values for ground truth class (0,1,2)
print(y[...])
Problems with K-Means

• Only works for numeric data (typically reals)

• *Very* sensitive to the initial points
  – *fix*: Do many runs, each with different initial centroids
  – *fix*: Seed centroids with non-random method, e.g., *farthest-first* sampling

• Sensitive to outliers
  – E.g.: find three
  – *fix*: identify and remove outliers

• Must manually choose k
  – Learn optimal k using some performance measure
Problems with K-Means

• How do you tell it which clustering you want?

• Constrained clustering technique provides hints

---

Same-cluster constraint (must-link)  Different-cluster constraint (cannot-link)
K-means Clustering Summary

• Clustering useful & effective for many tasks
• K-means clustering one of simplest & fastest techniques, but
  – Requires knowing how many clusters is right
  – Doesn’t handle outliers well
• There are many other clustering options