Unsupervised Learning: Clustering

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.”

* Yann LeCun (Head of Facebook AI, NYU CS Prof.) on AlphaGo’s success and AI, 2016
Unsupervised Learning

• Supervised learning used labeled data pairs \((x, y)\) to learn a function \(f : X \rightarrow y\)
• But, 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 categorical values; may benefit from normalization
• We’ll look at three popular approaches
  1. Centroid-based clustering
  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-estimate cluster centroids based on its data assigned
- Convergence: no point is re-assigned to a different cluster
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) # get mean across columns
>>> centroid
array([2.0, 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

$k = 5$
K-Means Clustering

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

K-Means (k, data)
- Randomly choose k cluster center locations (centroids)
- Loop until convergence
  - Assign each point to the cluster of the closest centroid
  - Re-estimate the cluster centroids based on the 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
Clusterer

SimpleKMeans -init 0 -max-candidates 100 -periodic-pruning 10000 -min-density 2.0 -t1 -1.25 -t2 -1.0 -N 3 -A "weka.core.EuclideanDistance" -R first

Cluster mode

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

Clusterer output

Within cluster Sum of Squared Errors: 77.677438592303374

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</th>
<th>Cluster# 0</th>
<th>Cluster# 1</th>
<th>Cluster# 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>sepal length</td>
<td>5.8433</td>
<td>5.936</td>
<td>5.006</td>
<td>6.588</td>
</tr>
<tr>
<td>sepal width</td>
<td>3.054</td>
<td>2.77</td>
<td>3.418</td>
<td>2.974</td>
</tr>
<tr>
<td>petal length</td>
<td>3.7587</td>
<td>4.26</td>
<td>1.464</td>
<td>5.552</td>
</tr>
<tr>
<td>petal width</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 Iris-versicolor Iris-setosa Iris-virginica</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Clipped Model and evaluation on training set:

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Count ( % )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>50 (33%)</td>
</tr>
<tr>
<td>1</td>
<td>50 (33%)</td>
</tr>
<tr>
<td>2</td>
<td>50 (33%)</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
- Percentage split
- Classes to clusters evaluation
- Store clusters for visualization

Clusterer output

```
sepal length  5.8433  5.8885  5.006  6.8462
sepal width   3.054   2.7377  3.418  3.0821
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

0    61 (41%)
1    50 (33%)
2    39 (26%)

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

- MiniBatchKMeans
- AffinityPropagation
- MeanShift
- SpectralClustering
- Ward
- AgglomerativeClustering
- DBSCAN
- Birch
- GaussianMixture
sklearn K-means clustering on Fisher's Iris dataset

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

[15] 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] print(X[:3])
    print(y[:3])

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]]
[0 0 0]
```

[27] 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

---

Same-cluster constraint (must-link)  Different-cluster constraint (cannot-link)
(2) Hierarchical clustering

• **Agglomerative**
  – **Bottom-up** approach: elements start as individual clusters & clusters are merged as one moves up the hierarchy

• **Divisive**
  – **Top-down** approach: elements start as a single cluster & clusters are split as one moves down the hierarchy
Hierarchical Clustering

Recursive partitioning/merging of a data set
Dendogram

- Tree structure representing all data partitionings
- Constructed as clustering proceeds
Dendogram

- Tree structure representing all data partitionings
- Constructed as clustering proceeds
- Get a K-clustering by looking at connected components at any given level
- Often binary dendograms, but n-ary ones easy to obtain with minor algorithm changes

Four clusters
Hierarchical clustering advantages

• Need not specify number of clusters
• Good for data visualization
  – See how data points interact at many levels
  – Can view data at multiple granularity levels
  – Understand how all points interact
• Specifies all of the K clusterings/partitions
Divisive hierarchical clustering

• Top-down technique to find best partitioning of data, generally exponential in time

• Common approach:
  – Let $C$ be a set of clusters
  – Initialize $C$ to be a one-clustering of data
  – While there exists a cluster $c$ in $C$
    • remove $c$ from $C$
    • partition $c$ into 2 clusters ($c_1$ and $c_2$) using a flat clustering algorithm (e.g., k-means)
      • Add to $c_1$ and $c_2$ $C$
    • Bisecting k-means
Divisive clustering
Divisive clustering

start with one cluster
Divisive clustering

use flat clustering to split into two clusters (e.g., using K-means with $k=2$)
Divisive clustering
Divisive clustering

split using flat clustering, e.g., Kmeans
Divisive clustering

split using flat clustering, e.g., Kmeans
Divisive clustering

Stop when clusters reach some constraint
Agglomerative Clustering

All observations start as their own cluster. Clusters meeting some criteria are merged. This process is repeated, growing clusters until some end point is reached.

Chris Albon
Hierarchical Agglomerative Clustering

• Let $\mathbf{C}$ be a set of clusters
• Initialize $\mathbf{C}$ to all points/docs as separate clusters
• While $\mathbf{C}$ contains more than one cluster
  – find $c_1$ and $c_2$ in $\mathbf{C}$ that are closest together
  – remove $c_1$ and $c_2$ from $\mathbf{C}$
  – merge $c_1$ and $c_2$ and add resulting cluster to $\mathbf{C}$
• Merging history forms a binary tree or hierarchy
• Q: How to measure distance between clusters?
Distance between clusters

Single-link: Similarity of the most similar (single-link)

$$\max_{l \in L, r \in R} \text{sim}(l, r)$$
Distance between clusters

**Complete-link:** Similarity of the “furthest” points, the *least* similar

\[
\min_{l \in L, r \in R} \sim(l, r)
\]

Weka: **linkType=COMPLETE**
Distance between clusters

Centroid: Clusters whose centroids (centers of gravity) are the most similar

\[ \| \mu(L) - \mu(R) \|^2 \]

Weka: linkType=CENTROID
Distance between clusters

**Average-link:** Average similarity between all pairs of elements

\[
\frac{1}{|L| \cdot |R|} \sum_{x \in L, y \in R} \|x - y\|^2
\]

Weka: linkType=AVERAGE
Default SINGLE cluster distance gives poor results here.
Using **AVERAGE** cluster distance measure improves results
Knowing when to stop

• General issue is knowing when to stop merging/splitting a cluster
• We may have a problem specific desired range of clusters (e.g., 3-6)
• There are some general metrics for assessing quality of a cluster
• There are also domain specific heuristics for cluster quality
(3) DBSCAN Algorithm

• Density-Based Spatial Clustering of Applications with Noise

• Clusters close points based on a distance and a minimum number of points
  – Key parameters: $\text{eps}=$ maximum distance between two points; $\text{minPoints}=$ minimal cluster size

• Marks as outliers points in low-density regions

• Needn’t specify number of clusters expected

• Fast
DBSCAN looks for densely packed observations and makes no assumptions about the number or shape of clusters.

1. A random observation, $x_i$, is selected.
2. If $x_i$ has a minimum of close neighbors, we consider it part of a cluster.
3. Step 2 is repeated recursively for all of $x_i$'s neighbors, then neighbors' neighbors etc... These are the cluster's core members.
4. Once Step 3 runs out of observations, a new random point is chosen.

Afterwards, observations not part of a core are assigned to a nearby cluster or marked as outliers.

Chris Albon
Comparing Clustering algorithms

Scikit Learn — Plot Cluster Comparison
DBSCAN Example

This gif (in ppt) shows how DBSCAN grows four clusters and identifies the remaining points as outliers.
DBSCAN, (Density-Based Spatial Clustering of Applications with Noise), captures the insight that clusters are dense groups of points. The idea is that if a particular point belongs to a cluster, it should be near to lots of other points in that cluster.