Clustering using Python

The objective of this article is to show how to classify observations using Python when we don't have an explicit classification criterion. In machine learning, this is known as unsupervised learning.
Unsupervised learning uncovers hidden structure from unlabeled data. It encompasses a variety of techniques in machine learning, from clustering to dimension reduction to matrix factorization. One example could be to classify customer based on their purchase.

Clustering and its application - Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group than those in other groups. Clustering has application in recommendation engines, market segmentation, image segmentation, anomaly detection and many more areas.

Clustering Algorithm

Let's look at some key clustering algorithms in detail:

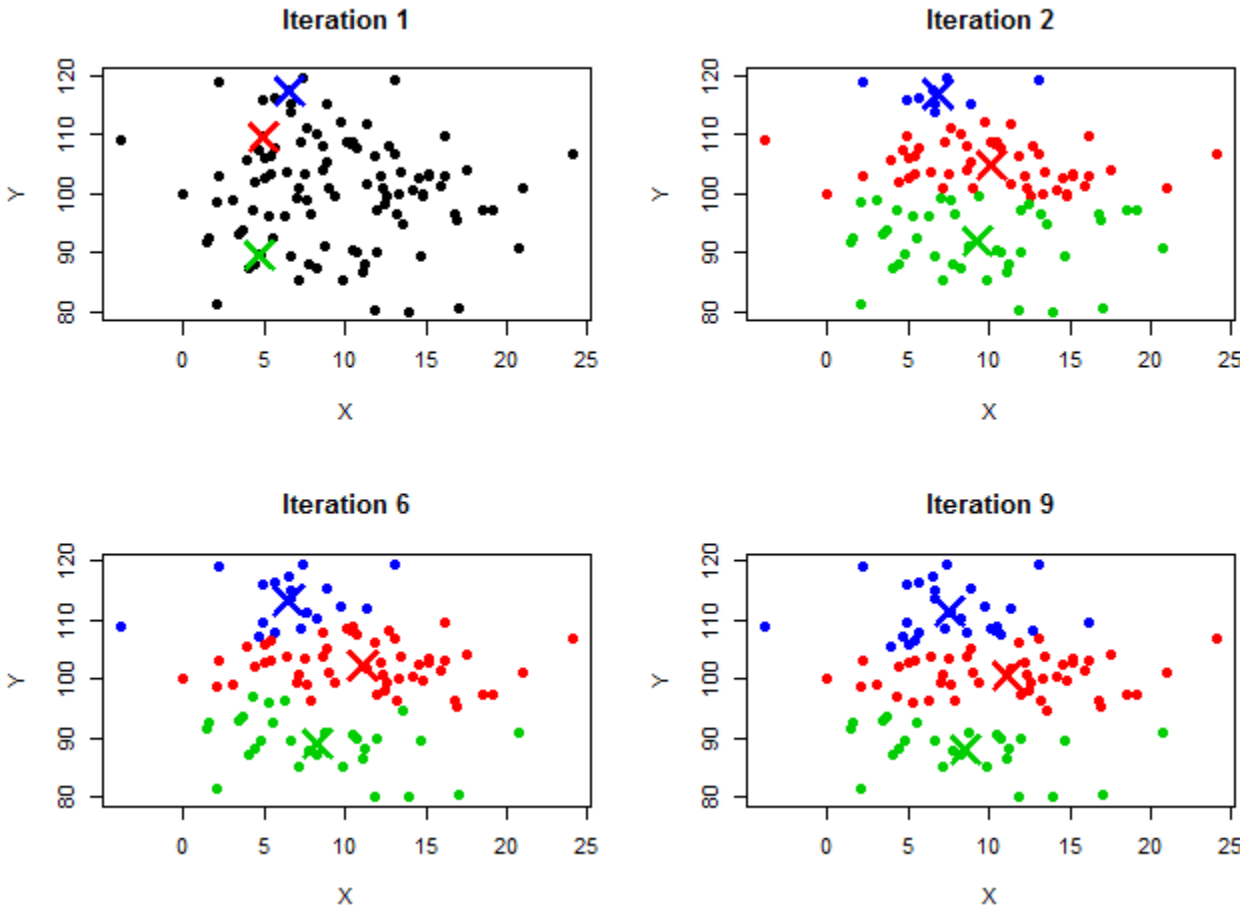
- Centroid models: These are iterative clustering algorithms in which the notion of similarity is derived by the closeness of a data point to the centroid of the clusters. In this approach, the no. of clusters required at the end should be mentioned beforehand, which makes it important to have prior knowledge of the dataset. Example K-Means clustering algorithm

- Connectivity models: These are based on the notion that the data points closer in data space exhibit more similarity to each other than the data points lying farther away. These models can follow two approaches. In the first approach, they start with classifying all data points into separate clusters & then aggregating them as the distance decreases. In the second approach, all data points are classified as a single cluster and then partitioned as the distance increases. Also, the choice of the distance function is subjective. These models are very easy to interpret but lack scalability for handling big datasets. Examples hierarchical clustering algorithm.

- Distribution models: These are based on the notion of how probable is it that all data points in the cluster belong to the same distribution (For example: Normal, Gaussian). These models often suffer from overfitting. Example Expectation-maximization algorithm which uses multivariate normal distributions.

Now I will explain two of the most popular clustering algorithms in detail – K-Means clustering and Hierarchical clustering.

In K-means feature variance impact feature influence and hence it is recommended to standardize the variable. K-Means is found to work well when the shape of the clusters is hyperspherical.



K-Means clustering iterations

Interesting read on non-convex sets with k means. https://pafnuty.wordpress.com/2013/08/14/non-convex-sets-with-k-means-and-hierarchical-clustering/

In Python, to standardize, import StandardScaler from sklearn.preprocessing and to Normalize Import Normalizer from sklearn.preprocessing. While StandardScaler() standardizes features by removing the mean and scaling to unit variance, Normalizer() rescales each sample - independently of the other. In K-Means clustering, since we start with the random choice of clusters, the results produced by running the algorithm multiple times might differ.

How to choose a good number of clusters for a dataset using the k-means inertia graph

Inertia measures how spread out the clusters are (lower the better). In other words, it measures the distance from each sample to the centroid of its cluster. In the graph of inertia vs a number of clusters look at elbow points to find out the optimum number of clusters.

```
# Import pyplot
import matplotlib.pyplot as plt
# Import KMeans
from sklearn.cluster import KMeans
ks = range(1, 10)
inertias = []
for k in ks:
    # Create a KMeans instance with k clusters: model
    model = KMeans(n_clusters=k)
    # Fit model to sample_dataset
    model.fit(sample_dataset)
    # Append the inertia to the list of inertias
    inertias.append(model.inertia_)
# Plot ks vs inertias
plt.plot(ks, inertias, '-o')
plt.xlabel('number of clusters, k')
plt.ylabel('inertia')
plt.xticks(ks)
plt.show()
```

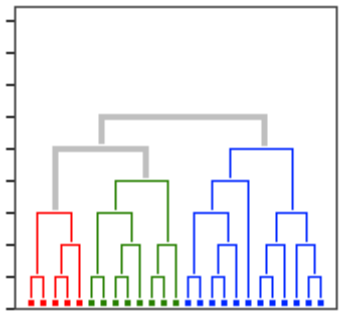
Data standardization and clustering

```
# Perform the necessary imports
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import pandas as pd

# Create scaler: scaler
scaler = StandardScaler()

# Create KMeans instance: kmeans
kmeans = KMeans(n_clusters=5)

# Create pipeline: pipeline
```

```
pipeline = make_pipeline(scaler,kmeans)

# Calculate the cluster labels: labels
labels = pipeline.predict(samples)

# Create a DataFrame with labels and species as columns: df
df = pd.DataFrame({'labels':labels, 'species':species})

# Create crosstab: ct
ct = pd.crosstab(df['labels'],df['species'])

# Display ct
print(ct)
```

Hierarchical clustering

In Hierarchical clustering bottom-up approach, each variable begins in a separate cluster and at each step, the two closer cluster are merged. This process continues until all variables are in the single cluster. Hierarchical clustering is more intuitive and we are not required to give the number of clusters beforehand. The disadvantage of Hierarchical clustering is that it can handle only a few data-points and takes exponential time for the high number of observations.



Hierarchical clustering

How to choose a good number of clusters for a dataset using the Hierarchical clustering

The best choice of the number of clusters is the number of vertical lines in the dendrogram cut by a horizontal line that can transverse the maximum distance vertically without intersecting a cluster.
In Python, SciPy linkage () function performs hierarchical clustering on an array of samples with the method='complete' or 'single'. In the complete linkage, the distance between clusters is the distance between the furthest points of the clusters. In single linkage, the distance between clusters is the distance between the closest points of the clusters. We can use dendrogram () to visualize the result. It may be noted that height on dendrogram specifies the maximum distance between merging clusters. We can use

the fcluster() function to extract the cluster labels for this intermediate clustering and compare the labels with the grain varieties using a cross-tabulation

```
# Perform the necessary imports
from scipy.cluster.hierarchy import linkage, dendrogram


import matplotlib.pyplot as plt
# SciPy hierarchical clustering doesn't fit into a sklearn pipeline, s
o we need to use the
# normalize() function from sklearn.preprocessing instead of Normalize
r.


# Import normalize
from sklearn.preprocessing import normalize
# Normalize the movements: normalized_movements
normalized_movements = normalize(sample_data)
# Calculate the linkage: mergings
mergings = linkage(normalized_movements, method='complete')



# Plot the dendrogram, using varieties as labels
dendrogram(mergings,labels=varieties, leaf_rotation=90,leaf_font_size=
6)
plt.show()
```

Another method of clustering is t-SNE. t-SNE maps the data samples into 2d space so that the proximity of the samples to one another can be visualized. “t-SNE stands for t-distributed stochastic neighbor embedding”
t-SNE provides great visualizations when the individual samples can be labeled.

```
# Import TSNE
from sklearn.manifold import TSNE
# Create a TSNE instance: model
model = TSNE(learning_rate=200)


# Apply fit_transform to samples: tsne_features
tsne_features = model.fit_transform(sample_data)


# Select the 0th feature: xs
```

```
xs = tsne_features[:,0]


# Select the 1st feature: ys
ys = tsne_features[:,1]
# Scatter plot, coloring by variety_numbers
plt.scatter(xs,ys,c=variety_numbers)
plt.show()
```

Recommended Reading:
http://scikit-learn.org/stable/modules/clustering.html#homogeneity-completeness-and-v-measure
http://www.learnbymarketing.com/tutorials/k-means-clustering-in-r-example/
http://varianceexplained.org/r/kmeans-free-lunch/