ST1511
AI & MACHINE LEARNING

Practical 6
Unsupervised Learning

What you will learn / do in this lab

1. *Explore Unsupervised Learning concepts and applications*
2. *Conduct Clustering experiment using Python*
3. *Conduct Anomaly Detection experiment using Python*

TABLE OF CONTENTS

1.
OVERVIEW

In this practical we will use python with scikit-learn to apply unsupervised learning to some typical problems encountered. Scikit-learn comes with estimators that help to solve the clustering and anomaly detection problems.

INTRODUCTION TO UNSUPERVISED LEARNING

Unsupervised learning is where you only have input data (**X**) and no corresponding output variables.

The goal for unsupervised learning is to **model the underlying structure** or distribution in the data in order to learn more about the data.

These are called unsupervised learning because unlike supervised learning above there is **no correct answers** and there is **no teacher**. Algorithms are left to their own devises to discover and present the interesting structure in the data.

APPLICATIONS OF UNSUPERVISED LEARNING

Unsupervised learning problems can be further grouped into clustering, anomaly detection, dimensionality reduction and association problems.

- ***Clustering****: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.*
- ***Anomaly Detection****: A anomaly detection problem is where you want to discover unusual data points. This is related to clustering,*

but what you want to discover are the points that are far from any cluster.

- ***Dimensionality Reduction**: A dimensionality reduction problem is where you want to discover the features or transformation of features that gives the best representation the data in relation to the problem. For reducing the number of attributes in data for summarization, visualization and feature selection such as Principal component analysis.*
- ***Association**: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.*

Some popular examples of unsupervised learning algorithms are:

- *k-means for clustering problems.*
- *Apriori algorithm for association rule learning problems.*

Some examples of use cases are:

- *Behavioral segmentation:*
    - *Segment by purchase history*
    - *Segment by activities on application, website, or platform*
    - *Define personas based on interests*
    - *Create profiles based on activity monitoring*
- *Inventory categorization:*
    - *Group inventory by sales activity*
    - *Group inventory by manufacturing metrics*
- *Sorting sensor measurements:*
    - *Detect activity types in motion sensors*
    - *Group images*
    - *Separate audio*
    - *Identify groups in health monitoring*
- *Detecting bots or anomalies:*
    - *Separate valid activity groups from bots*
    - *Group valid activity to clean up outlier detection*
    - *In addition, monitoring if a tracked data point switches between groups over time can be used to detect meaningful changes in the data.*

2.
CLUSTERING

In this section we would use clustering algorithms from scikit-learn to discover groups or clusters in the data.

K-MEANS

We would look into using the k-means algorithms to find clusters in the iris data set. We assume that the class label is removed from the data set first (by ignoring the iris.target).

K-means clustering is a type of unsupervised learning, which is used when you have unlabeled data (i.e., data without defined categories or groups). The goal of this algorithm is to find groups in the data, with the number of groups represented by the variable K.

The algorithm works iteratively to assign each data point to one of K groups based on the features that are provided. Data points are clustered based on feature similarity.

The results of the K-means clustering algorithm are:

- *The centroids of the K clusters, which can be used to label new data*
- *Labels for the training data (each data point is assigned to a single cluster)*

Algorithm

The K-means clustering algorithm uses iterative refinement to produce a final result. The algorithm inputs are the number of clusters K and the data set. The data set is a collection of features for each data point.

The algorithms starts with initial estimates for the K centroids, which can either be randomly generated or randomly selected from the data set.

The algorithm then iterates between two steps:

1. *Data assignment step:*

   Each centroid defines one of the clusters. In this step, each data point is assigned to its nearest centroid, based on the squared Euclidean distance.

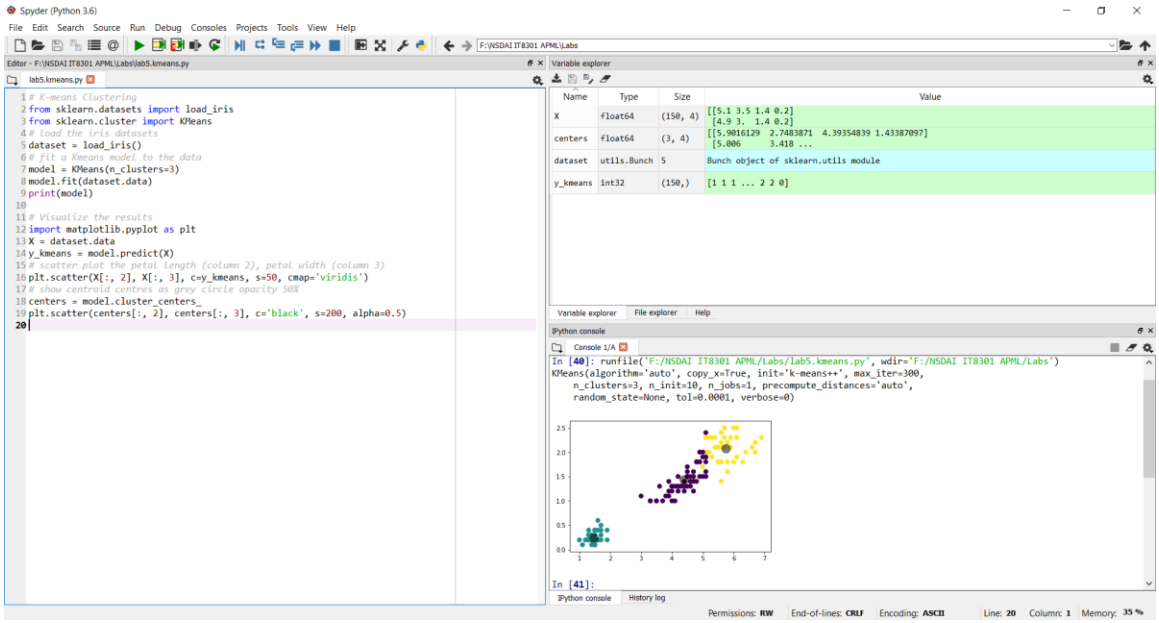
2. *Centroid update step:*

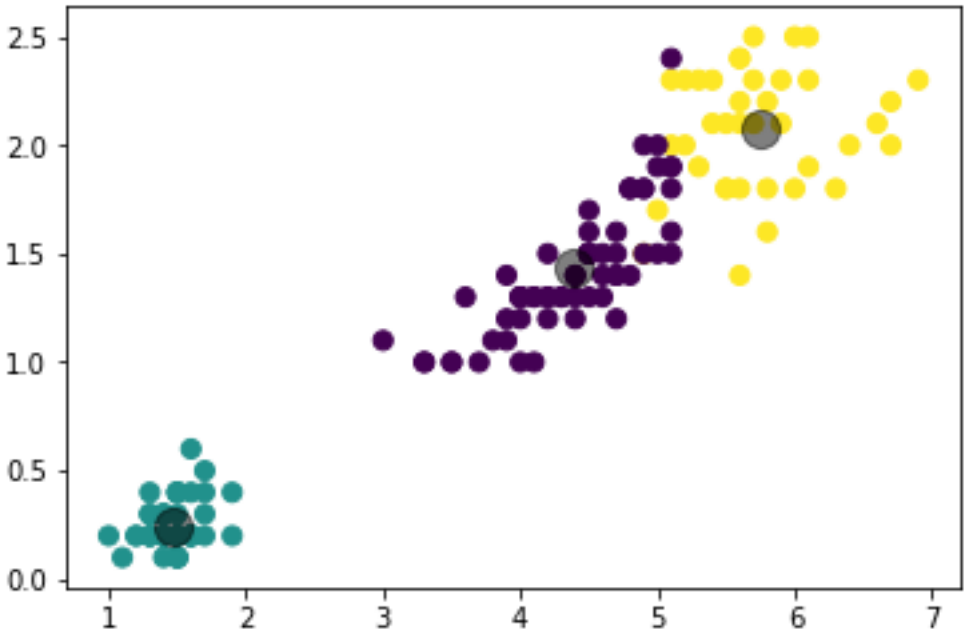
   In this step, the centroids are recomputed. This is done by taking the mean of all data points assigned to that centroid's cluster.

The algorithm iterates between steps one and two until a stopping criterion is met (i.e., no data points change clusters, the sum of the distances is minimized, or some maximum number of iterations is reached).

```
# K-means Clustering
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
# load the iris datasets
dataset = load_iris()
# fit a Kmeans model to the data
model = KMeans(n_clusters=3)
model.fit(dataset.data)
print(model)

# Visualize the results
import matplotlib.pyplot as plt
X = dataset.data
y_kmeans = model.predict(X)
# scatter plot the petal length (column 2), petal width (column 3)
plt.scatter(X[:, 2], X[:, 3], c=y_kmeans, s=50, cmap='viridis')
# show centroid centres as grey circle opacity 50%
centers = model.cluster_centers_
plt.scatter(centers[:, 2], centers[:, 3], c='black', s=200, alpha=0.5)
```


Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Editor - F:\NSDAI IT8301 APML\Labs\lab5.kmeans.py
lab5.kmeans.py
# K-means Clustering
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
# load the iris datasets
dataset = load_iris()
# Fit a Kmeans model to the data
model = KMeans(n_clusters=3)
model.fit(dataset.data)
print(model)
# Visualize the results
import matplotlib.pyplot as plt
X = dataset.data
y_kmeans = model.predict(X)
# scatter plot the petal length (column 2), petal width (column 3)
plt.scatter(X[:, 2], X[:, 3], c=y_kmeans, s=50, cmap='viridis')
# show centroid centres as grey circle opacity 50%
centers = model.cluster_centers_
plt.scatter(centers[:, 2], centers[:, 3], c='black', s=200, alpha=0.5)
Variable explorer
Name Type Size Value
X float64 (150, 4) [[5.1 3.5 1.4 0.2]
[4.9 3. 1.4 0.2]
centers float64 (3, 4) [[5.9016129 2.7483871 4.39354839 1.43387097]
[5.006 3.418 ...
dataset utils.Bunch 5 Bunch object of sklearn.utils module
y_kmeans int32 (150,) [1 1 1 ... 2 2 0]
Variable explorer File explorer Help
IPython console
Console 1/A
In [40]: runfile('F:/NSDAI IT8301 APML/Labs/lab5.kmeans.py', wdir='F:/NSDAI IT8301 APML/Labs')
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
n_clusters=3, n_init=10, n_jobs=1, precompute_distances='auto',
random_state=None, tol=0.0001, verbose=0)
In [41]:
IPython console History log
Permissions: RW End-of-lines: CRLF Encoding: ASCII Line: 20 Column: 1 Memory: 35 %


2.5
2.0
1.5
1.0
0.5
0.0
1
2
3
4
5
6
7

CHOOSING THE NUMBER OF CLUSTERS K

From URL:
http://scikit-learn.org/stable/modules/clustering.html#silhouette-coefficient ,

A **higher Silhouette Coefficient score** relates to a model with better-defined clusters.

The Silhouette Coefficient is defined for each sample and is composed of two scores:

- ***a**: The mean distance between a sample and all other points in the same class.*
- ***b**: The mean distance between a sample and all other points in the next nearest cluster.*

The Silhouette Coefficient is for a single sample is then given as:

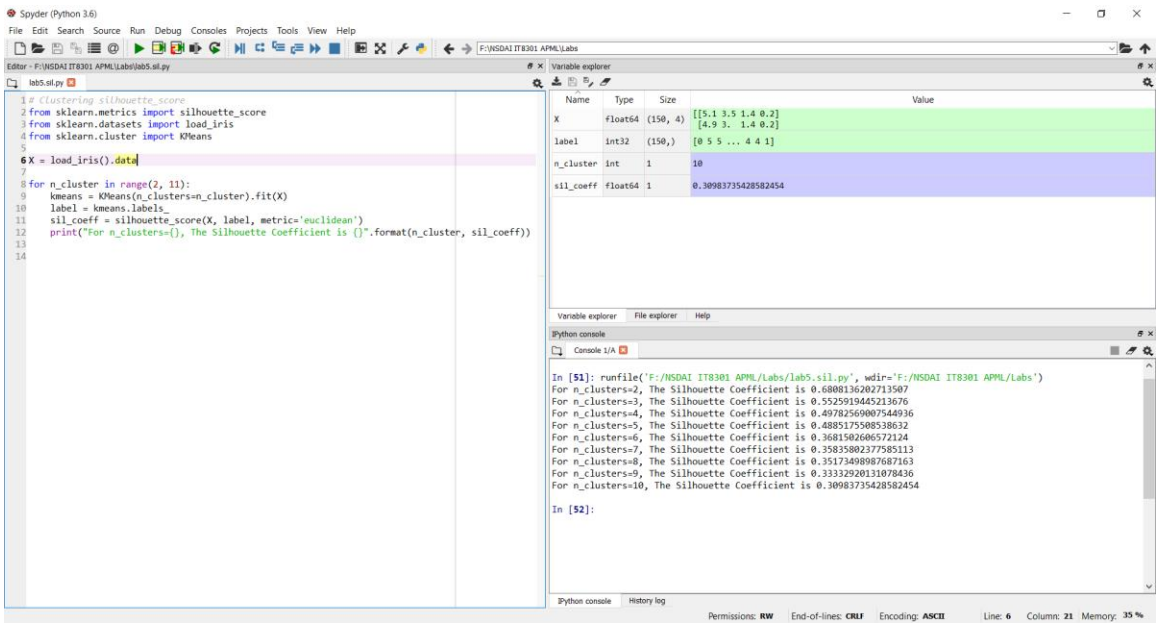
$$s = \frac{b - a}{max(a, b)}$$

Assuming that we don't know the number of iris clusters initially, we can use this coefficient (s) to determine the K to use.

```
# Clustering silhouette_score
from sklearn.metrics import silhouette_score
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans

X = load_iris().data

for n_cluster in range(2, 11):
    kmeans = KMeans(n_clusters=n_cluster).fit(X)
    label = kmeans.labels_
    sil_coeff = silhouette_score(X, label, metric='euclidean')
    print("For n_clusters={}, The Silhouette Coefficient is {}".format(
            n_cluster, sil_coeff))_
```



For n_clusters=2, The Silhouette Coefficient is 0.6808136202713507
For n_clusters=3, The Silhouette Coefficient is 0.5525919445213676
For n_clusters=4, The Silhouette Coefficient is 0.49782569007544936
For n_clusters=5, The Silhouette Coefficient is 0.4885175508538632
For n_clusters=6, The Silhouette Coefficient is 0.3681502606572124
For n_clusters=7, The Silhouette Coefficient is 0.35835802377585113
For n_clusters=8, The Silhouette Coefficient is 0.35173498987687163
For n_clusters=9, The Silhouette Coefficient is 0.33332920131078436
For n_clusters=10, The Silhouette Coefficient is 0.30983735428582454

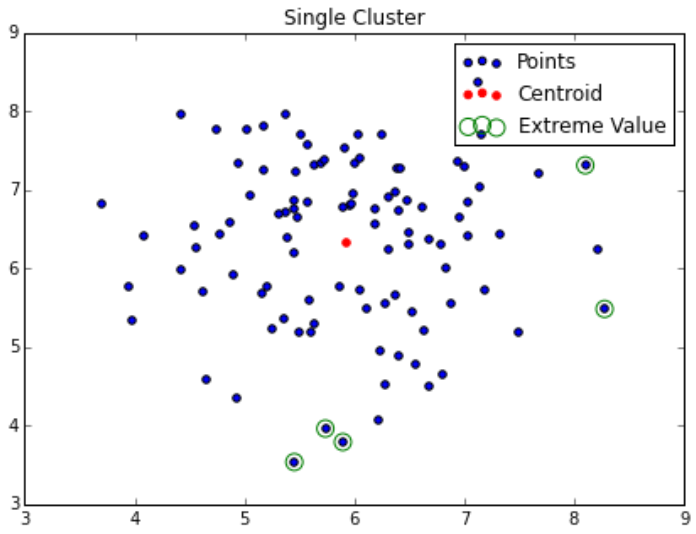
For the iris data set, s is highest with K=2, this is because the third species is in the middle overlapping the other 2 species. So, we used K=3 for the iris data set because of other sources of knowledge about the problem domain.

3.
ANOMALY DETECTION

In this section, we will detect anomalies or unusual data occurences

ANOMALY DETECTION USING K-MEANS

We can use K-means clustering algorithm to perform anomaly detection and outlier detection. If we set the K=1, we effectively generate only one centroid for the data, points that are far away from the centre are considered to be outliers or anomalies.



Finding a cluster with one center is similar to an SVM with one class (One-class SVM is an anomaly detection algorithm http://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html)

In this experiment, we will use the synthetic dataset found in sklearn called make_blob
(see http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html

```
# Anomaly Detection using K-means
from sklearn.datasets import make_blobs
import numpy as np
from sklearn.cluster import KMeans
# generate the data
X, label = make_blobs(100, centers = 1)
# get the K-means model
kmeans = KMeans(n_clusters=1)
kmeans.fit(X)
print(kmeans)
# Visualize the results
import matplotlib.pyplot as plt
y_kmeans = kmeans.predict(X)
# scatter plot the data points
f, ax = plt.subplots(figsize=(7, 5))
ax.set_title('Blob')
ax.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis',
           label='Points')
# show centroid centres as red circle opacity 50%
centers = kmeans.cluster_centers_
ax.scatter(centers[:, 0], centers[:, 1], c='red', s=200, alpha=0.5,
           label='Centroid')
ax.legend()
plt.show()
# identify the 5 closest points
# argsort returns an array of indexes which will sort the array in
# ascending order
# so we reverse it via [::-1] and take the top five with [:5]
distances = kmeans.transform(X)
sorted_idx = np.argsort(distances.ravel())[::-1][:5]
# Now let's see which are the top 5 points furthest away
f, ax = plt.subplots(figsize=(7, 5))
ax.set_title("Single Cluster")
ax.scatter(X[:, 0], X[:, 1], label='Points')
centers = kmeans.cluster_centers_
ax.scatter(centers[:, 0],
           centers[:, 1],
           label='Centroid', color='r')
ax.scatter(X[sorted_idx][:, 0], X[sorted_idx][:, 1],
           label='Extreme Value', edgecolors='g',
           facecolors='none', s=100)
ax.legend(loc='best')
```

```
plt.show()
# It's easy to remove these points if we like:
new_X = np.delete(X, sorted_idx, axis=0)
# Or to extract the anomalies
anomaly_X = X[sorted_idx,:]
print(anomaly_X)
```