# Machine Learning
# Esercitazione di laboratorio sul K-means

1) MATERIALE NECESSARIO (contenuto nel file .ZIP scaricabile dal sito del corso)

   - common.py
   - Kmeans.py
   - Kmeans_SingleExecution.py
   - Kmeans_ClusterCountEstimation.py
   - PerfumeData.txt

2) CONOSCENZE UTILI

   - Per la generazione di numeri pseudo-casuali sfruttare la libreria `random`.

3) ALGORITMO K-MEANS

   Implementare l'algoritmo K-means per raggruppare un insieme di pattern acquisiti analizzando una serie di profumi con dei sensori di odore. Effettuare alcune prove al variare del numero di cluster.

   a. Il dataset PerfumeData.txt contiene 560 pattern bidimensionali non etichettati acquisiti analizzando un insieme di profumi con due sensori di odore.
   b. Il file Kmeans.py è incompleto e contiene puntini di sospensione "…" laddove è richiesto di completare il codice.
   c. Analizzare nel dettaglio la funzione `kmeans_execution`. La funzione implementa lo pseudocodice riportato a pag. 8 delle dispense "Clustering".
   d. Implementare la funzione `select_random_centroids` che scelga casualmente `s` pattern (`data`) da utilizzare come centroidi iniziali. I centroidi dovranno essere memorizzati per riga in un Numpy array bidimensionale (`centroids`). Utilizzare il seme `seed` per l'inizializzazione del generatore di numeri casuali (`random`).
   e. Implementare la funzione `assign_patterns_to_clusters` che restituisca un Numpy array monodimensionale (`labels`) contenente l'indice del cluster a cui è stato assegnato ogni pattern. Per individuare il centroide più vicino ad ogni pattern utilizzare la funzione `compute_square_euclidean_distance` già disponibile nel codice. Inoltre si dovrà calcolare (e restituire) la somma dei quadrati delle distanze di tutti i pattern dai centroidi a cui sono stati assegnati memorizzandola in `sumSquareDistances` (indicato con $J_e$ a pag. 4 delle dispense "Clustering").
   f. Implementare la funzione `compute_centroids` che dovrà restituire, sotto forma di un Numpy array bidimensionale (`centroids`), i nuovi centroidi (memorizzati per riga) calcolati come media dei pattern appartenenti ad ogni cluster.
   g. Utilizzare il file Kmeans_SingleExecution.py per verificare il funzionamento dell'algoritmo al variare del numero di cluster (`s`), del seme per la scelta casuale dei centroidi iniziali (`seed`) e del massimo numero di iterazioni consentite (`maxIterationCount`). Assegnare alla variabile `datasetFilePath` il percorso del file PerfumeData.txt.

4) TROVARE IL NUMERO DI PROFUMI PRESENTI

   Stimare il numero di profumi presenti utilizzando l'algoritmo K-means sviluppato al punto precedente.

   a. Il file Kmeans_ClusterCountEstimation.py è incompleto e contiene puntini di sospensione "…" laddove è richiesto di completare il codice.

b. Assegnare alle variabile `datasetFilePath` il percorso del file PerfumeData.txt.
c. Il codice presente esegue l'algoritmo K-means variando il numero di cluster (`s`) nell'intervallo [`sMin`; `sMax`].
d. Per evitare che il risultato ottenuto possa essere condizionato da una scelta errata dei centroidi iniziali si consiglia di eseguire più volte (`executionCount`) il K-means per ogni valore di `s`.
e. Il grafico mostrato al termine dell'esecuzione riporta il valore minimo di $J_e$ (su `executionCount` esecuzioni per ogni `s`) al variare di `s`.
f. Cercare di individuare il numero di profumi presenti nel dataset analizzando il grafico. Si faccia attenzione al fatto che, per come è definito $J_e$, valori grandi di `s` producono più facilmente valori piccoli di $J_e$.