kMeans

Deadline: 31 december 2023, 23:59
Total number of points: 1.7

1. Preprocessing

a. Provide a brief description of the dataset. What are the attributes? What is the purpose of the dataset? Specify which attributes are discrete and continuous.
b. Identify the NaN's (Not a Number) in your dataset. Remove the rows that contain such values.
c. Calculate the mean and variance for each numerical attribute.
d. Remove the target attribute from your dataset.

2. Distances

a. Convert the discrete attributes that are not numeric (such as strings or boolean values) into numerical. If this doesn't apply to your dataset, provide a short explanation on how you would proceed.
b. Write a function `distance_points` that calculates the distance between two points. The function should take three parameters: the two points and `p`, where `p` indicates the order of the Minkowski distance (remember that `p=1` is the equivalent for the Manhattan distance, and `p=2` for the Euclidean one).
c. Write a function `generate_random_points` that generates `n` `d`-dimensional points using the uniform distribution. The values should be greater than `left_range` and lower than `right_range`.
d. Write a function `distance_to_df` that calculates the distance between a point `X` and a dataframe `df`. The function should return a vector with `n` values that contains the distance between `X` and each instance belonging to `df` (`n` represents the number of instances of the dataframe).
Hint: Check the norm calculation function from the `numpy` module.

3. kMeans

a. Write a function `distance_to_centroids` that calculates the distance between the points from a dataset and a list of centroids. The function will take as parameters the dataframe `df` and the list of centroids `centroids` and will return a `n x m` matrix, where `n` is the number of points from `df` as `m` the number of centroids.

b. Write a function `closest_centroid` that, using the output from the previous function, determines the closest centroid for each point. The function should return a list that for each point contains the index of the closest centroid.

c. Write a function `get_clusters` that uses the closest centroid list to create the list of clusters. The function will return a dictionary

```
{ index_centroid_1 : [index point for which centroid 1 is the closest]}
```

d. Using the list of clusters and the dataframe, write a function `update_centroids` that will recalculate the centroids as the arithmetic mean of the points from each cluster. The function should return a list with the new coordinates of the centroid.
*Note*s:

1. Treat the case when a cluster is empty, i.e. there is a centroid that is not considered the closest for any of the points from the dataframe.
2. Keep the order of the old indices, meaning the new centroid of the cluster 2 should be the third in the list (assuming the indexing starts with 0).

e. Write a function that performs the kMeans++ initialisation. The function should take as parameters the dataframe `df`, the desired number of clusters `nclusters` and the random seed (for reproducibility) and should return the list of centroids.

f. Write the implementation of the kMeans algorithm. The function should have the following parameters: the dataframe `df`, the desired number of clusters, the number of iterations, the initalisation type (random or kmeans++) and the random seed. The function should return a dictionary with the following fields:

- `clusters`: the membership vector (for each point, the index of the cluster it belongs to)
- `centroids`: the coordinates of the centroids

g. Write a function that, given a dataframe `df`, a membership vector `mb` and the list of `centroids`, calculates the J score.

h. Write a function that enables multiple initialisations. Besides the parameters specified at *f*, you will add the number of initialisations.
By multiple initialisation we understand running kmeans multiple times with different random seeds.
The function will return the clustering with the best J score.
The output will be a dictionary with the fields `clusters`, `centroids` and `J`.

i. Run the kmeans implementation from `h` on your dataset with the following parameters: `ninit` = 100, `niter` = 30, `init` = "kmeans++" and `nclusters` varying from 2 to 30. Plot the evolution of the J score as the number of clusters increases.
What is the natural number of clusters in your case? Justify your reasoning.

Notes:

- make sure you include the functions implemented in the previous points!
- the implementation of the kMeans should be done by you, do not use framework such as `sklearn` to build the model (except for the comparison).
- the Assignment should be written in a Jupyter Notebook that will be sent via email