TP2 clustering en C++

nom : Bouymedj
prénom : wissam
CHPS M1
2019/2020

Conditions expérimentales

Les algorithmes sont testés sous linux sur un PC avec un processeur Intel Core i3 2.4Ghz, 4Go de RAM. Les programmes sont compilés avec cmake .

//clustering

// la fonction make_uniform_dataset : Générer de manière aléatoire deux vecteurs x et y (nombre de point) et on donne les bornes de ces vecteurs (bound_min et bound_max)

```
auto make_uniform_dataset(int nb_point, double bound_min , double bound_max)
{

auto x = std::vector<double>(nb_point);
auto y = std::vector<double>(nb_point);
auto uniform = std::uniform_real_distribution<double>(bound_min,bound_max);
auto rg= std::default_random_engine();

    for(auto i =0 ; i< nb_point ; ++i)
    {
    x[i]=uniform(rg);
    y[i]=uniform(rg);
    }

return std::make_pair(x,y);
}
```

//la fonction make_blob_dataset :
générer de maniéré aléatoire le nombre de blobs et le nombre de point dans un chaque blob

```
auto make_blob_dataset(int nb_point_per_blob,int nb_blobs ,double bound_min , double bound_max )
{

    auto x = std::vector<double>();
    x.reserve(nb_point_per_blob*nb_blobs);
    auto y = std::vector<double>();
    y.reserve(nb_point_per_blob*nb_blobs);
        auto rg= std::default_random_engine();

    for(auto i=0;i<nb_blobs;++i){

    std::normal_distribution<double> generateurdex (xblob_centers[i],0.3);
    std::normal_distribution<double> generateurdey (yblob_centers[i],0.3);
        for(auto j=0; j<nb_point_per_blob; ++j){

            auto xx= generateurdex(rg);
            auto yy= generateurdey(rg) ;
            x.push_back(xx);
            y.push_back(yy);
        }
    }

return std::make_pair(x,y);
}
```

//K-means

est un algorithme non supervisé de clustering . Il permet de regrouper en **K** clusters distincts les observations du data set. Ainsi les données similaires se retrouveront dans un même cluster. Une même observation, ne pourra donc, appartenir à deux clusters différents.

Pour pouvoir regrouper un jeu de données en K cluster distincts, l'algorithme K-Means a besoin d'un moyen de **comparer le degré de similarité** entre les différentes observations. Ainsi, deux données qui se ressemblent, auront une **distance de dissimilarité** réduite, alors que deux objets différents auront une distance de séparation plus grande.

- **La distance Euclidienne** : C'est la distance géométrique qu'on apprend au collège. Soit une matrice X à n variables quantitatives. Dans l'espace vectoriel E^n. La distance euclidienne d entre deux observations x_1 et x_2 se calcule comme suit :

$$d(x_1, x_2) = \sqrt{\sum_{j=1}^{n} (x_{1n} - x_{2n})^2}$$

```
auto k_means(std::vector<double>x,std::vector<double>y,
    double nb_cluster,
    double bound_min,double bound_max,
    int nombre.déterations,int nb_point,
    const DataFrame& data)
{
    auto centers=make_uniform_dataset(nb_cluster,bound_min,bound_max);
    auto xcenters=centers.first;
    auto ycenters=centers.second;
  std::vector<size_t> assignments(data.size());
  for (size_t M = 0; M < nombre.diterations; ++M) {
    for(size_t k=0 ;k<nb_point ;++k){
      auto meilleur.distance = std::numeric_limits<double>::max();
      auto meilleur.cluster = 0;
      for (auto l = 0; l <nb_cluster; ++l) {
        point v{xcenters[l], ycenters[l]};
        const double distance = distance(data[k],v);
        if (distance < meilleur.distance) {
          meilleur.distance = distance;
          meilleur.cluster = l;
        }
      }
      assignments[k] = best_cluster;
```

Fonctionnement de l'algorithme K-Means

k-means est un algorithme itératif qui minimise la somme des distances entre chaque individu et le centroïd. Le **choix initial des centroïdes conditionne le résultat final**.
Admettant un nuage d'un ensemble de points, K-Means change les points de chaque cluster jusqu'à ce que la somme ne puisse plus diminuer. Le résultat est un ensemble de clusters compacts et clairement séparés, sous réserve de choisir la bonne valeur K du nombre de clusters .

Algorithme K-means

Entrée :

- K le nombre de cluster à former
- dataframe(matrice de données)

DEBUT

Choisir aléatoirement K points (une ligne de la matrice de données). Ces points sont les centres des clusters (nommé centroïd).

REPETER

Affecter chaque point (élément de la matrice de donnée) au groupe dont il est le plus proche au son centre

Recalculer le centre de chaque cluster et modifier le centroide

JUSQU'A CONVERGENCE

//conclusion :

En analysant la façon de procéder de l'algorithme de K-means, on remarque que pour un même jeu de données, **on peut avoir des partitionnements différents**. En effet, L'initialisation des tous premiers K centroids est complétement aléatoire. Par conséquent l'algorithme trouvera des clusters différents en fonction de cette première initialisation aléatoire. De ce fait, la configuration des clusters trouvés par K-Means peut ne pas être la plus **optimale.**