Middle East Technical University | Department of Computer Engineering

CENG 499

Introduction to Machine Learning

Fall '2018-2019
Assignment 2

Yusuf Mücahit Çetinkaya
{yusufc@ceng.metu.edu.tr}
Due date: December 16, 2018, Sunday, 23:59

1 Overview

In this assignment, you are going to implement three popular clustering and classification algorithms with Python programming language. K-means (MacQueen, 1967) [1] is one of the simplest unsupervised learning algorithms that solve the well-known clustering problem. The procedure follows a simple and easy way to classify a given data set to a predefined, say K number of clusters.[4] K-Medoids is another clustering algorithm which is less sensitive to outliers.[2][3] Instead of random points, it select real samples as centroids. The k-Nearest-Neighbours (kNN) is a non-parametric classification method, which is simple but effective in many cases. For a data record t to be classified, its k nearest neighbours are retrieved, and this forms a neighbourhood of t. [5]

Template Python files are given to you. You are allowed to use numpy library and distance methods defined in scipy library. However, you are not allowed to use K-means, K-medoids or K-nearest neighbors implementations inside scikit-learn or any similar library in these implementations.

Keywords: *kMeans, kMedoids, k-Nearest Neighbor, clustering, classification, color quantization, data visualization*

2 Tasks

2.1 MyKMeans (25 Pts.)

For this task, you will implement *MyKMeans.py* class. This class is similar to *KMeans class in the scikit-learn* but simpler.

The model has several attributes as follows;

- n_clusters: The number of clusters to form as well as the number of centroids to generate.
- init_method: Method for choosing initial centroids. May be one of 'random', 'kmeans++' or 'manual'
- max_iter: Maximum number of iterations of the k-means algorithm for a single run.
- random_state: The seed for the python random library
- cluster_centers: if init_method is 'manual', then this array is used for initial values.

It has *initialize, fit, predict, fit_and_predict* methods.
"initialize" method initializes cluster centers with given init_method. If initialization method is randomized, cluster centers are chosen randomly from given data set. If initialization method is set to "kmeans++", it selects first centroid randomly from given samples. The other centroids are also selected from the samples according to their distance to previously selected centroids. For each centroid, the the sample with highest total euclidean distance to previous centroids is chosen. If initialization of centroids is manual, cluster_centers are to be the initial values of centroids.
"fit" method gets a data set as input and model gets trained by calculating cluster centers. "predict" method gets a data set as input and returns the predicted labels for each sample in the data set according to the distance to cluster centers. If initialization method is manual, initial centers can be used for predicting without fitting.
"fit_and_predict" method gets a data set as input, the model gets trained like in the "fit" method, returns predicted labels for each sample like in the "predict" method.

After the implementation is done, one should be able to simply use MyKMeans class as follows;

```
from MyKMeans import MyKMeans
import numpy as np
X =np.array([[1, 2], [1, 4], [1, 0],
             [4, 2], [4, 4], [4, 0]])
kmeans =MyKMeans(n_clusters=2, random_state=0, init_method='kmeans++')
print kmeans.initialize(X)
# [[4. 4.]
# [1. 0.]]
kmeans =MyKMeans(n_clusters=2, random_state=0, init _method='random')
print kmeans.initialize(X)
# [[4. 0.]
# [1. 0.]]
kmeans.fit(X)
print kmeans.labels
# array([1, 1, 1, 0, 0, 0])
print kmeans.predict([[0, 0], [4, 4]])
```

```
# array([1, 0])
print kmeans.cluster_centers
# array([[4, 2],
# [1, 2]])
```

Note: Be careful! If a seed is given, initial and final cluster_centers should be same with the scikit learn implementation with same seed.

2.2 MyKMedoids (25 Pts.)

You will implement *MyKMedoids.py* class in this task. The model has several attributes as follows;

- n_clusters: The number of clusters to form as well as the number of medoids to determine.
- max_iter: Maximum number of iterations of the k-medoids algorithm for a single run.
- method : Implies which K-Medoids algorithm to apply. If it is pam, it applies pam algorithm to whole dataset 'pam'. If it is 'clara' it selects number of samples with sample_ratio and applies pam algorithm to the samples. Returns best medoids of all trials according to cost function.
- sample_ratio: float, default: .2 It is used if the method is 'clara'
- clara_trials: int, default: 10, It is the number of trials for 'clara' method.
- random_state: The seed for the python random library

It has *sample, pam, generate_clusters, calculate_cost, fit, predict, fit_and_predict* methods. "sample" method returns a random sample with predefined sample ratio from given data.
"pam" method simply applies PAM algorithm to data. Details for the implementation can be found inside the given source code.
"generate_clusters" method assigns each sample to the closest medoid's cluster and returns list of clusters. "calculate_cost" method calculates total squared euclidean cost on the given configurations of medoids and clusters.
"fit" method gets a data set as input and model gets trained by applying given algorithm. Medoids are randomly initialized.
"predict" method gets a data set as input and returns the predicted labels for each sample in the data set according to the distance to medoid.
"fit_and_predict" method gets a data set as input, the model gets trained like in the "fit" method, returns predicted labels for each sample like in the "predict" method.

After the implementation is done, one should be able to simply use MyKMedoids class as follows;

```
from MyKMedoids import MyKMedoids
import numpy as np
X =np.array([np.array([2., 6.]),
             np.array([3., 4.]),
             np.array([3., 8.]),
             np.array([4., 7.]),
             np.array([6., 2.]),
             np.array([6., 4.]),
```

```
              np.array([7., 3.]),
              np.array([7., 4.]),
              np.array([8., 5.]),
              np.array([7., 6.])
              ])
kmedoids =MyKMedoids(n_clusters=2, random_state=0)
print kmedoids.fit_predict(X)
# [1 1 1 1 0 0 0 0 0 0]
print kmedoids.best_medoids
# [array([7., 4.]), array([2., 6.])]
print kmedoids.min_cost
# 28.0
```

2.3 MyKNeighborsClassifier (30 Pts.)

Just like in part 2.1, you will implement MyKNeighborsClassifier. This class is similar to *KNeighborsClassifier class in the scikit-learn* but simpler. The model has several attributes as follows;

- n_neigbors: Number of neighbors to use
- method: 'classical', 'weighted' or 'validity'. 'classical' weights has equal weights on voting for each neighbor. In 'weighted' version, weights are decreased by the distance. In this case, closer neighbors of a query point will have a greater influence than neighbors which are further away. If 'validity' method is chosen, in fit method validity of each training sample should be calculated. This value is multiplied with the weight calculated like in weighted method. In this case, outliers have less impact on prediction. validity for each training sample is the weight of its' true label on its' n nearest neighbors with weighted distance.
- norm: 'l1' or 'l2'. Distance norm. 'l1' is Manhattan distance. 'l2' is euclidean distance.

It has fit, predict, predict_proba methods. "fit" method gets a data set and labels as input and model does not get trained since KNN is a lazy classifier. It only saves the data and labels inside. However, if 'validity' method is chosen, then validity for each training sample is calculated in "fit" method.
"predict" method gets a data set as input, determines k-nearest neighbor to assign and return the label of each sample.
"predict_proba" behaves like "predict" function, however, it returns probabilities of each class label. Sum of the probabilities should be equal to 1. If method is 'classical', then probabilities for each class label is simply [# of neighbors member of this class/ n_neighbors]. However, if method is 'distance' or 'validity', then probabilities for each class is calculated by the vote formula and then normalized.

After the implementation is done, one should be able to simply use MyKNeighborsClassifier class as follows;

```
X =[[0], [1], [2], [3]]
y =[0, 0, 1, 1]
from MyKNeighborsClassifier import MyKNeighborsClassifier
neigh =MyKNeighborsClassifier(n_neighbors=3, method="validity")
```

```
neigh.fit(X, y)
print(neigh.predict([[1.1]]))
# [0]
print(neigh.predict_proba([[0.9]], method='classical'))
# [[0.66666667 0.33333333]]
print(neigh.predict_proba([[0.9]], method='weighted'))
# [[0.75089392 0.24910608]]
print(neigh.predict_proba([[0.9]], method='validity'))
# [[0.75697674 0.24302326]]
```

Note: The results should be as close as possible to scikit-learn library result.

2.4 ColorQuantizer (20 pts.)

This class performs a pixel-wise Vector Quantization (VQ) of the given image like in *scikit-learn examples*. This class has 4 methods. 3 of them will be used in "quantize_image". You will use your implementation of MyKMeans for this task. You will read a JPEG image and store it as numpy array. After quantization operation is done, you will save the cluster centers (selected colors) and quantized image in png format to the given path. It has three attributes;

- n_colors: The number of colors that wanted to exist at the end.
- random_state: random_state for the MyKMeans.

After the implementation is done, one should be able to simply use ColorQuantizer class as follows;

```
from ColorQuantizer import ColorQuantizer
import numpy as np
quantizer =ColorQuantizer(n_colors=64)
quantizer.quantize_image('metu.jpg', 'cluster_centers.txt', 'my_quantized_metu.png')
```

After quantizing colors, please submit calculated metu_cluster_centers.txt ankara_cluster_centers.txt files under Docs directory. You will get 10 points of grade according to how close your calculation is to the given image. You may think about sampling more informative colors from the image by any technique.

3 Submission & Evaluation

1. You will have **Source** and **Docs** directories. All of your .py files will be in Source directory. Others will be in the Docs directory.If your directory structure is messy, you will get **penalty**. If you have done something for bonus, explicitly write it in a readme file.

2. Your source codes will be tried with different configurations and various size data sets.

3. We have zero tolerance policy for **cheating**. People involved in cheating will be punished according to the university regulations and will get 0. You can discuss algorithmic choices, but sharing code between each other or using third party code is **strictly forbidden**. Even if you take a "part" of the code from somewhere/somebody else - this is also cheating. Please be aware that there are "very advanced tools" that detect if two codes are similar. So please do not think you can get away with by changing a code obtained from another source.

4. Your codes should be tested on inek machines before submit.

5. Do not put your .pyc file, IDE related files etc. Only .py files will be submitted.

6. Zip all directories and name it as <ID> _ <FullNameSurname> and submit it through COW. For example:
   e1234567_YusufMucahitCetinkaya.zip

References

[1] MacQueen, J. (1967, June). Some methods for classification and analysis of multivariate observations. In Proceedings of the fifth Berkeley symposium on mathematical statistics and probability (Vol. 1, No. 14, pp. 281-297).

[2] Rousseeuw, P. J., Kaufman, L. (1990). Finding groups in data. SeriesinProbabilityMathematicalStatistics 1990-34 (1), 111-112.

[3] Ng, R. T., Han, J. (1994, September). Efficient and Effective Clustering Methods for Spatial Data Mining. In Proceedings of VLDB (pp. 144-155).

[4] Kolay, S., Ray, K. S. (2017). K+ Means: An Enhancement Over K-Means Clustering Algorithm. arXiv preprint arXiv:1706.02949.

[5] Guo, G., Wang, H., Bell, D., Bi, Y., Greer, K. (2003, January). KNN model-based approach in classification. In CoopIS/DOA/ODBASE (Vol. 2003, pp. 986-996)