Hybrid Classic-Quantum Systems - Second Assignment

November 29, 2023

1 Hybrid Classic-Quantum Systems - Second Assignment

You should fill in this Jupyter notebook in order to complete the assignment. Here a small recap of the rules:

- You should **NOT** modify the functions that are already provided in the notebook, unless it is strictly necessary;
- If you want to modify the given functions, please, provide an explanation of why you had to;
- You can define new functions, as soon as they are well commented;
- You can import all libraries you want, as soon as you justify their utilization;
- You can add new cells, as soon as you do not remove the cells where you are supposed to comment your solution;
- You are supposed to work by yourself - **plagiarism will not be tolerated**;
- Your solution should be commented and accompanied by a small description - you can add additional cells if needed;
- For any issue and doubt, please do not hesitate to use the forum or to write me an email.

1.1 Exercise 1: Classic K-Means (25 points)

The first task you have to do for the assignment is the implementation of a classic version of k-means. This version will be used as a reference for your next exercises.

Some useful references: * https://en.wikipedia.org/wiki/K-means_clustering * NumPy reference: https://numpy.org/doc/stable/reference/ * Matplotlib reference: https://matplotlib.org/

1.1.1 Creation of the dataset

We create a simple multiclass dataset with normally distributed clusters of points. To do this, we use the *make_blobs* function from *scikit-learn*

[ ]:
```
from sklearn.datasets import make_blobs

def create_dataset(number_of_points, number_of_centers):
    data = make_blobs(n_samples=number_of_points, n_features=2,␣
 ↪centers=number_of_centers, cluster_std=2, random_state=100)
    points = np.array(data[0])
    centers = np.array(data[1])

    return data, centers
```

```
dataset, centers = create_dataset(100,4)
```

1.1.2 Normalize Input Points

We need to rescale the input in a specific interval.

[ ]:
```
import numpy as np

def preprocess(data_points):
    data_p = np.array(data_points[0])
    n = len(data_p)
    print(n)
    x = 30.0 * np.sqrt(2)
    for i in range(0,n):
        data_p[i][:] += 15
        data_p[i][:] /= x

    return data_p
```

1.1.3 Plot points

We can use this function to plot the points in the space during algorithm execution.

[ ]:
```
import matplotlib.pyplot as plt

def draw_plot(points,centers,label=True):
    if label==False:
        plt.scatter(points[:,0], points[:,1])
    else:
        plt.scatter(points[:,0], points[:,1], c=centers, cmap='viridis')
    plt.xlim(0,1)
    plt.ylim(0,1)
    plt.show()
```

1.1.4 Plot Centroids

You can use this function to check the update of the centroids.

[ ]:
```
def plot_centroids(centers):
    plt.scatter(centers[:,0], centers[:,1])
    plt.xlim(0,1)
    plt.ylim(0,1)
    plt.show()
```

1.1.5 Initialize Centroids

The first step of the algorithm is to initialize the centroids in the space, that will be updated in further iterations of the algorithm.

[ ]:
```
def initialize_centroids(points,k):

```

EDIT THIS CELL TO COMMENT YOUR SOLUTION

1.1.6 Find nearest neighbors

Now we need to find the closest center for each point. This computation depends on the distance function that we want to use.

[ ]:
```
def dist_func(points, centroids):


def find_nearest_neighbour(points, centroids, dist_func):

```

EDIT THIS CELL TO COMMENT YOUR SOLUTION

1.1.7 Find centroids

Finally, we update centroids based on the nearest centers for each point.

[ ]:
```
def find_centroids(points, centers):

```

EDIT THIS CELL TO COMMENT YOUR SOLUTION

1.1.8 Main k-means loop

[ ]:
```
n = 100  # number of data points
k = 4  # Number of centers


points, o_centers = create_dataset(n, k)  # dataset
points = preprocess(points)  # Normalize dataset

plt.figure()
draw_plot(points, o_centers, label=False)

centroids = initialize_centroids(points, k)  # Intialize centroids

# run k-means algorithm
### EDIT FROM HERE ###
```

EDIT THIS CELL TO COMMENT YOUR SOLUTION

1.1.9 Bonus question

- Modify your implementation into k-medians.
- Compare the results for the two algorithms: can you see any difference? **ADD MORE CELLS BELOW TO ANSWER THESE QUESTIONS**

1.2 Exercise 2: Quantum K-Means (45 points)

Now you have to modify your implementation of k-means and move some of the component on quantum.

1.2.1 Encoding

First step is to find out the best encoding for the input data points. References about data encoding: * Weigold et al., "Data encoding patterns for quantum computing" (https://dl.acm.org/doi/10.5555/3511065.3511068) * P. Niemann, R. Datta, and R. Wille, Logic Synthesis for Quantum State Generation, IEEE 46th International Symposium on Multiple-Valued Logic, Springer (2016) pp. 247-252. (https://ieeexplore.ieee.org/abstract/document/7515556) * M. Mttnen, J. Vartiainen, V. Bergholm, and M. M. Salomaa, Transformation of quantum states using uniformly controlled rotations, Quantum Information and Computation, 5 (2005) pp. 467-473. (https://dl.acm.org/doi/abs/10.5555/2011670.2011675)

[ ]:
```
def encode(point):

```

1.2.2 EDIT THIS CELL TO COMMENT YOUR SOLUTION

1.2.3 Identify which tasks can be executed on quantum devices

In this part, you should describe which part of the computation be executed on quantum devices. For each task, you should * Identify the corresponding algorithm * Provide a possible design for each algorithm, including possible data encoding

EDIT HERE TO COMMENT YOUR SOLUTION

1.2.4 Implement quantum tasks

In the next cells, you should implement **AT LEAST ONE** identified quantum tasks. For the ones you do not implement, you should justify why you decided to not implement it in quantum version. Justification can be related to the limit of available quantum hardware. Add as many cells you need for your code and describe your implementation.

1.2.5 Compare executions

Execute both classic and quantum version of k-means and compare results. Add as many cells you need to run your code, describe your implementation and comment your results.

1.2.6 Bonus question (15 points)

- What would you need to change to implement the k-medians?
- Are there additional quantum tasks that you can have if you implement k-medians?

-

1.3 Exercise 3: Quantum SVM (30 points)

In this exercise, you will apply quantum SVM to perform some basic classification task. You should perform training and evaluation for one of the selected datasets. Bonus points (up to 15) will be awarded if you perform dataset preprocessing.

1.3.1 Data analysis with Quantum SVM

[ ]:
```
from sklearn.datasets import load_iris, load_wine, load_breast_cancer
from qiskit_machine_learning.kernels import FidelityQuantumKernel
from qiskit.primitives import Sampler
from qiskit_algorithms.state_fidelities import ComputeUncompute
from qiskit_machine_learning.algorithms import QSVC
from sklearn.model_selection import train_test_split
### EDIT FROM HERE ###
```

EDIT HERE TO COMMENT YOUR SOLUTION

1.3.2 Try different Feature Maps

Check the results of your model by training with different Feature Maps. Which one gives the best results?

[ ]:
```
### EDIT FROM HERE ###
```

1.3.3 Compare with classic SVM

Compare your results with classic SVM.

[ ]:
```
### EDIT FROM HERE ###
```

EDIT HERE TO COMMENT YOUR SOLUTION

[ ]:
```

```