M 124 Coursework 3

Y. Panagakis and **K. Panousis**

Due Date: May 16, 2021, 23:59

1 Introduction

For the third coursework, we focus on K-Means, Kernels and Support Vector Machines.

In the seventh lecture, we explored the Unsupervised Learning scenario. Contrary to the supervised learning task that we had explored up to that point, in Unsupervised Learning, we assume that the training dataset is of the form $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, with $\boldsymbol{x}_i \in \mathbb{R}^d,\ i = 1, \ldots, N$; that is, we assume that we only have available training points without any corresponding ground truth label.

We introduced two basic rationales of Unsupervised Learning, Clustering and Dimensionality Reduction Methods, like PCA, NMF and ICA. In this exercise, you'll focus on clustering methods and specifically on one of the most popular approaches, K-Means. You will implement and experiment with the heuristic KMeans algorithm as described in the Lecture, and explore a more sophisticated and popular initialization method.

In the eighth Lecture, we focused on Reproducing Kernel Hilbert Spaces and showed how we can employ the *kernel trick* in order to map a non-linear task in a higher dimension to solve it in a linear way. We then introduced one of the most well known approaches in Machine Leaning, the Support Vector Machines. In this exercise you are asked to delve deeper into kernels and SVMs, exploring some theoretical aspects of the former, while focusing on a hands-on implementation of the latter in the context of Classification.

2 K-means Clustering **(60 Points)**

In this task, you will implement the K-means algorithm as defined in the respective Lecture and explore the various aspects of the objective as well as a more sophisticated initialization method.

As a reminder, given a set $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}_{n=1}^N$, where $\boldsymbol{x}_n \in \mathbb{R}^d$, the goal of K-means is to partition the data into K distinct clusters. Specifically, we aim to find a set of centers $\boldsymbol{c}_1, \ldots, \boldsymbol{c}_K \in \mathbb{R}^d$ to minimize the K-means objective:

$$\sum_{j=1}^{n} \min_{i \in \{i, \ldots, K\}} ||\boldsymbol{x}_j - \boldsymbol{c}_i||^2 \tag{1}$$

Partitioning the data into K distinct clusters is an NP-hard problem. To this end, we introduced a heuristic algorithm that alternates between two steps: (i) Computing the centroid for each cluster based on the observations assigned to it, and (ii) Re-assigning each point to the cluster with the nearest center. This process is repeated until the algorithm converges, i.e., the partitioning remains unchanged. Thus, the conventional K-Means algorithm can be summarized as follows:

Algorithm 1 K-Means Algorithm

1: Select the number of clusters K.
2: Randomly select a number from 1 to K for each observation. This will server as initial cluster assignments.
3: Repeat until there is no change in cluster assignments:
(a) For each of the K clusters, compute the *centroid*. The centroid is the vector of the p feature means for the observations assigned in that cluster.
(b) Assign each observation to the cluster whose centroid is closest (using Euclidean distance)

2.1 Implementing K-Means in Python (30 Points)

In this task, you are asked to implement the K-Means algorithm **from scratch**. You can use only native Python and numpy/scipy commands and not ready-made implementations of sklearn. Your implementation should follow an Object Oriented rationale, as the one showed in the Kernel Ridge Regression approach.

You are asked to create a KMeans class and your code should implement the following functionality:

- **Init** function: The initialization function of the class. Takes as input two arguments, an integer K that denotes the number of classes and a string argument init_method that denotes the initialization method to be used (in this exercise you are asked to implement two different initialization schemes).
- **Fit** function: A class method that takes as input the data matrix $N \times D$ with N denoting the number of examples and D the dimensionality of the data and a constant integer num_restarts that denotes the number of restarts. As mentioned in the respective Lecture, the clustering results of the heuristic KMeans algorithm will largely depend on the initialization. Thus, num_restarts denotes how many times we run the algorithm from random starting points, retaining the best performing one among these runs.
- **Predict** function: A class method that takes as input a data matrix $M \times D$ and returns the closest cluster for each of the M datapoints.

For debugging and assessing the performance of your implementation, you'll use a simple toy dataset provided in the corresponding Datasets path and illustrated in Figure 1. Use the given data to obtain cluster assignments using your KMeans implementation and plot the corresponding clusters with different colors.

2.2 Further Investigation (30 Points)

2.2.1 The effect of K (10 Points)

In this part, you will investigate the effect of the number of clusters K in KMeans while evaluating a heuristic approach for choosing a good value for K. To this end, you'll need to implement the following:

- Construct a function plot_objective_per_k which plots the k-means objective function evaluated by running the core k-means function that you implemented above for various values of $k \in \{1, 2, \ldots, 20\}$.
- A heuristic approach to selecting a value for k is to investigate the resulting and select the value at the "bend" of the plot; that is, the point where after increasing k beyond this value yields relatively little decrease in the corresponding objective (also called the "elbow" of the plot). Based on the produced plot what value would you choose? Does this value agree with your intuition about the number of clusters in the dataset?

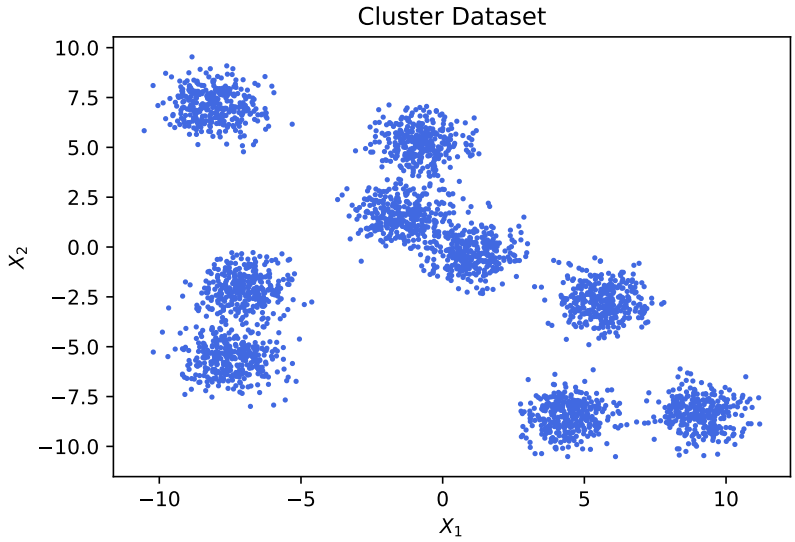

Figure 1: The dataset to be used to evaluate and investigate the KMeans approach.

2.2.2 The effect of initialization (20 Points)

In this part, you will explore the effect of the initialization in the Kmeans method. We discussed in the Lecture that the resulting clustering results heavily depend on the initialization of the clusters' centroids. In the conventional definition of the algorithm, this initialization is performed in a random way, randomly assigning a cluster for each observation. Using this simple method, requires running KMeans multiple times, retaining the best performing run according to the value of the objective function. Various methods have been proposed to overcome the simplicity and avoid the poor clusterings found from the standard random initialization approach. One of these methods is k-means++ which we describe below:

Algorithm 2 K-Means++ Initialization

1: Choose one centroid $\boldsymbol{c}_1$ uniformly at random from the datapoints.
2: For each datapoint $\boldsymbol{x} \in X$, compute $D(\boldsymbol{x})$: the distance between $\boldsymbol{x}$ and the nearest center that has already been chosen.
3: **repeat**
4: Choose a new data point to be a new center $\boldsymbol{c}_i$, using a weighted probability distribution, where a point x is chosen with probability proportional to $D^2(x)$.
5: **until** we have K centers
6: Proceed with the standard kmeans algorithm.

Thus, you must implement and explore the following:

- Implement the aforementioned k-means++ initialization algorithm in your KMeans class definition.
- For both type of initializations, i.e. "random" and "kmeans++":
    1. Plot the corresponding initial centroids and the datapoints colored based on their class label (a plot similar to Fig. 1).

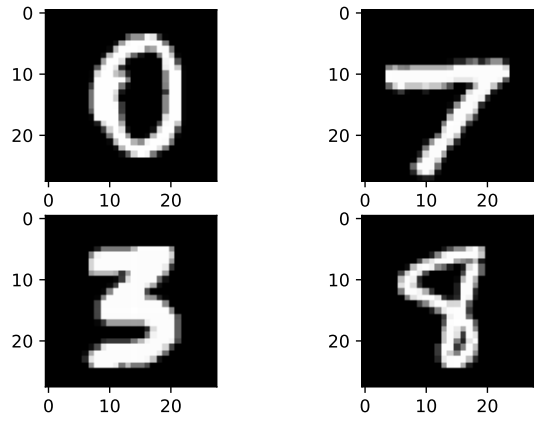

Figure 2: Example digits from the MNIST dataset.

2. Report the mean and standard deviation of the K-means objective value using $K = 9$ and num_restarts $= 1$ over 800 runs. Since both initializations are random, the results will vary each time you perform this task but there should be a clear performance difference.

- In general, what is the issue with the random initialization in k-means? How does k-means++ address this issue?

3 Kernel Construction (10 Points)

Suppose that $K_1(\boldsymbol{u}, \boldsymbol{v}) \in \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ and $K_2(\boldsymbol{u}, \boldsymbol{v}) \in \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ are both *valid kernel functions*. Prove the following:

- $c_1 K_1(\boldsymbol{u}, \boldsymbol{v}) + c_2 K_2(\boldsymbol{u}, \boldsymbol{v})$ is a valid kernel function., where $c_1, c_2 \geq 0$ are real constants.
- $K_1(\boldsymbol{u}, \boldsymbol{v}) K_2(\boldsymbol{u}, \boldsymbol{v})$ is a valid kernel function.
- Given a polynomial $p : \mathbb{R} \to \mathbb{R}$ with positive coefficients, the function:

  $$K(\boldsymbol{u}, \boldsymbol{v}) = p(K_1(\boldsymbol{u}, \boldsymbol{v}))$$

  is a valid kernel function.
- $K(\boldsymbol{u}, \boldsymbol{v}) = e^{K_1(\boldsymbol{u}, \boldsymbol{v})}$ is a valid kernel function.

4 Support Vector Machines (30 Points)

In this question you will explore the Support Vector Machines in the Classification task. Specifically, you'll use the MNIST dataset, one of the most well known benchmark data in Image Processing. The dataset consists of 70000 examples, where a typical split comprises 50000 points for training, 10000 for validation and 10000 for testing. Each data point represents a 28×28 image of handwritten digits as seen in Fig. 2.

Many approaches cannot deal with two-dimensional data; hence, in these approaches, we "vectorize" the data to obtain a 1-dimensional representation. After this procedure, each example is described by $28 \times 28 = 784$ features.

In this task you are asked to:

- Load the MNIST dataset and normalize the data. This task should be performed using both $(0, 1)$ normalization and $(-1, 1)$ to compare the results.
- In SVMs there are various choices that may affect the resulting performance. For example the type of kernel used and the values of its parameters or the value for C. Thus, you will need to examine multiple values of these parameters and report the resulting performances, highlighting the values selected for the best performing run. For this task, consider a train/test split using 60000 examples for training and 10000 examples for testing [Hint: For easier implementation you may use a function that we saw in the corresponding Lab.].
- Report the parameters that correspond to the best performing model, i.e. kernel type, C and gamma. Use these to compute the classification accuracy for both the training and the test set.
- Perform a PCA transformation on the data and rerun the SVM method using the best parameters of the previous question. To this end, use the PCA implementation of sklearn, selecting 3 different values for the retained variance (0 < n_components < 1.)
- For each run, report the number of components retained and the obtained classification accuracy. Record the times between the different runs and draw some conclusions about a potential trade-off between accuracy, dimensionality reduction and complexity.

General Notes

- In this coursework, you are asked to implement basic approaches to ML problems. As was the case in the previous courseworks, I highly recommend to take your time and understand the fundamental techniques and rationale behind the introduced approaches and avoid searching ready-made solutions online.
- Individual Coursework.
- Must be submitted via e-class. E-mailed submissions will not be accepted.
- Programming:
  - For the programming part, you can turn to StackOverflow if you bump into some kind of bug or problem, but again I strongly advise against copying code for these simple tasks.
  - As mentioned in the first tutorial, it is best practice to use Virtual Environments for your projects to avoid breaking the system. You can use either Virtual Environments of native python venv or install an Anaconda/Miniconda platform. Or simply use Google Colab.
  - Implement your approach using a Jupyter Notebook, with sufficient but not redundant comments.
- Typesetting:
  - I would suggest using LaTeXfor reporting the results. LaTeXis a very powerful typesetting system, most commonly employed in research papers and reports. It provides all the necessary tools for easily typesetting equations, arrays and more, while avoiding the hassles of other editors. This CS was written in LaTeX. You can use Overleaf for online editing.
  - A template will be provided.