10-315 Introduction to Machine Learning: Homework 6

Due: 11:59 pm, Monday, April 22, 2019

Instructions

- **Submit your homework on time electronically by submitting to Autolab by 11:59 pm, Monday, April 22, 2019**.

  We recommend that you use LaTeX, but we will accept other typesetting as well. To submit this homework, you should submit a pdf of your solutions on Autolab by navigating to Homework 6 and clicking the "Submit File" button.

- **Late homework policy**: Homework 6 is worth full credit if submitted before the due date. Up to 50% credit can be received if the submission is less than 48 hours late. The lowest homework grade at the end of the semester will be dropped. Please talk to the instructor in the case of extreme extenuating circumstances.

- **Collaboration policy**: You are welcome to collaborate on any of the questions with anybody you like. However, you must write up your own final solution, and you must list the names of anybody you collaborated with on this assignment.

1 k-means Clustering [60 pts]

In this problem you will implement Lloyd's method for the k-means clustering problem and answer several questions about the k-means objective, Lloyd's method, and k-means++.

Recall that given a set $S = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$ of n points in d-dimensional space, the goal of k-means clustering is to find a set of centers $c_1, \ldots, c_k \in \mathbb{R}^d$ that minimize the k-means objective:

$$\sum_{j=1}^{n} \min_{i \in \{1,\ldots,k\}} \|x_j - c_i\|^2, \tag{1}$$

which measures the sum of squared distances from each point x_j to its nearest center.

Consider the following simple brute-force algorithm for minimizing the k-means objective: enumerate all the possible partitionings of the n points into k clusters. For each possible partitioning, compute the optimal centers $c_1, \ldots, c_k$ by taking the mean of the points in each cluster and compute the corresponding k-means objective value. Output the best clustering found. This algorithm is guaranteed to output the optimal set of centers, but unfortunately its running time is exponential in the number of data points.

(a) [8 pts] For the case $k = 2$, argue that the running time of the brute-force algorithm above is exponential in the number of data points n.

In class we discussed that finding the optimal centers for the k-means objective is NP-hard, which means that there is likely no algorithm that can efficiently compute the optimal centers. Instead, we often use Lloyd's method, which is a heuristic algorithm for minimizing the k-means objective that is efficient in practice and often outputs reasonably good clusterings. Lloyd's method maintains a set of centers $c_1, \ldots, c_k$ and a partitioning of the data S into k clusters, $C_1, \ldots, C_k$. The algorithm alternates between two steps: (i) improving the partitioning $C_1, \ldots, C_k$ by reassigning each point to the cluster with the nearest center, and (ii) improving the centers $c_1, \ldots, c_k$ by setting c_i to be the mean of those points in the set C_i for $i = 1, \ldots, k$. Typically, these two steps are repeated until the clustering converges (i.e, the partitioning $C_1, \ldots, C_k$ remains unchanged after an update). Pseudocode is given below:

1. Initialize the centers $c_1, \ldots, c_k$ and the partition $C_1, \ldots, C_k$ arbitrarily.
2. Do the following until the partitioning $C_1, \ldots, C_k$ does not change:
   i. For each cluster index i, let $C_i = \{x \in S : x \text{ is closer to } c_i \text{ than any other center}\}$, breaking ties arbitrarily but consistently.
   ii. For each cluster index i, let $c_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$.

Implementing Lloyd's Method

In the remainder of this problem you will implement and experiment with Lloyd's method and the k-means++ algorithm on the two dimensional dataset shown in Figure 1.

(b) [8 pts] Complete the function `update_assignments(X, C)`. The input `X` is the $n \times d$ data matrix, `C` is the $k \times d$ matrix of current centers. The function returns a vector of current cluster assignments `assignments`, an array of length n. That is, `C[i]` is the center for cluster i and the j^{th} data point `X[j]`, is assigned to cluster `assignments[j]`. Your function should output a vector of cluster assignments of shape $(n,)$ so that each point is assigned to the cluster with the nearest center.

(c) [8 pts] Complete the function `[C] = update_centers(X, prev_C, assignments)`. The input arguments are as in part (b). Your function should output a $k \times d$ matrix `C` whose i^{th} row is the optimal center for those points in cluster i.

(d) [8 pts] Complete the function `lloyd_iteration(X, C0)`. This function takes a data matrix `X`, initial centers `C0` and runs Lloyd's method until convergence. Specifically, alternate between updating the assignments and updating the centers until the the assignments stop changing. Your function should output the final $k \times d$ matrix `C` of centers and final the vector `assignments` of assignments.

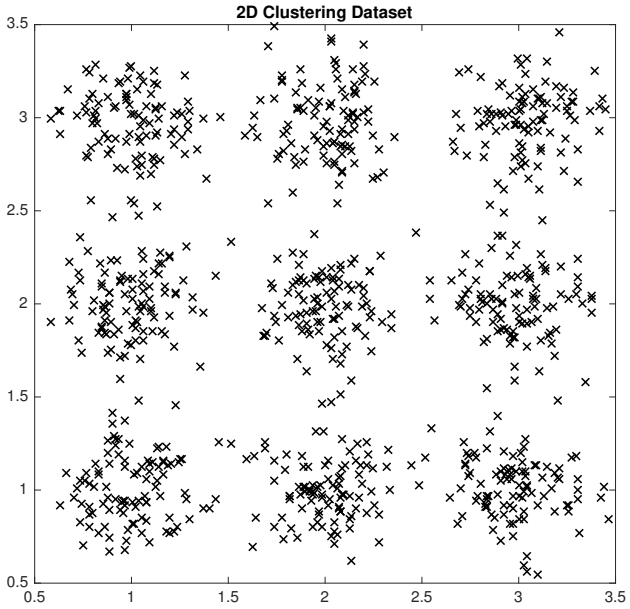

Figure 1: The 2D dataset used for problem 1.

(e) [8 pts] Complete the function `kmeans_obj(X, C, assignments)`. This function takes the $n \times d$ data matrix `X`, a $k \times d$ matrix `C` of centers, and a length n vector `assignments` of cluster assignments. Your function should output the value of the k-means objective of the provided clustering.

We provide you with a function `kmeans_cluster(X, k, init, num_restarts)` that takes an $n \times d$ data matrix `X`, the number `k` of clusters, a string `init` which must be either `'random'` or `'kmeans++'`, and a number of restarts `num_restarts`. This function runs Lloyd's method `num_restarts` times and outputs the best clustering it finds. You may use this function when answering the following questions. When `init` is set to `'random'` then the centers are initialized uniformly at random, and when `init` is set to `'kmeans++'` the centers are initialized using the D^2 weighting of k-means++.

Please go to the Handouts section of Homework 6 Code Autolab page to download the starter code and the data for both problems.

Experiment 1: The effect of k on Lloyd's method

In parts (f) and (g) of this problem you will investigate the effect of the number of clusters k on Lloyd's method and a simple heuristic for choosing a good value for k. You can load the data for this problem by `load_data()`. This will introduce a 900×2 matrix `X`, where each row corresponds to a single point in the dataset shown in Figure 1.

(f) [5 pts] Write a short script to plot the k-means objective value obtained by running `kmeans_cluster(X, k, 'random', 10)` for each value of `k` in $\{1, \ldots, 20\}$. The x-axis of your plot should be the value of `k` used, and the y-axis should be the `k`-means objective. Include both the plot and script in your written solutions.

(g) [5 pts] One heuristic for choosing the value of k is to pick the value at the "elbow" or bend of the plot produced in part (f), since increasing k beyond this point results in relatively little gain. Based on your plot, what value of k would you choose? Does this agree with your intuitions when looking at the data?

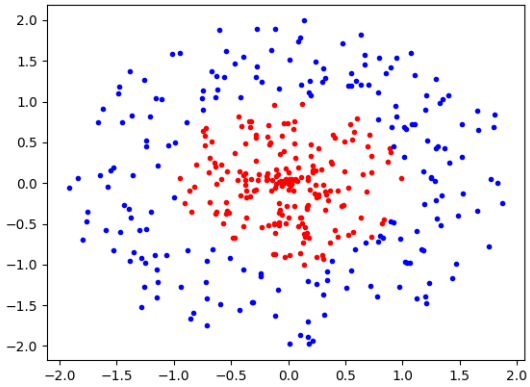

Figure 2: Dataset 1

Experiment 2: The effect of initialization on Lloyd's method

In parts (h) and (i) you will investigate the effect of the center initialization on Lloyd's method. In particular, you will compare random initialization with the k-means++ initialization. Note that the provided `kmeans_cluster` function has both kinds of initialization already implemented. In this problem, you just need to run the code and answer questions.

(h) [5 pts] For each value of `init` in [`'random'`, `'kmeans++'`], report the average k-means objective value returned by `kmeans_cluster(X, 9, init, 1)` over $N = 1000$ runs (note that the number of restarts is set to 1). Since the initialization is random, your answers will vary each time you perform this experiment. The variation in your outcomes should be small relative to the gap between the two objective values.

(i) [5 pts] Intuitively, a good center initialization for the 2D dataset shown in Figure 1 is when we pick one center from each of the 9 natural Gaussian clusters. When two centers land in the same Gaussian cluster, they will split that cluster in half and leave only 7 centers to cover the remaining 8 Gaussian clusters. Explain in 1–2 sentences why using k-means++ initialization is more likely to choose one sample from each cluster than random initialization.

2 AdaBoost [40 pts]

In this problem, you will explore some interesting properties of AdaBoost through intuitive examples. You will be asked to implement the AdaBoost algorithm with decision stumps as base learners to perform classification on a synthetic dataset. A decision stump is an axis-aligned linear separator that classifies samples to one side as positive, and samples to the other side as negative. Suppose the input is $\mathbf{x} = (x_1, x_2) \in \mathbb{R}^2$, we use the following decision stumps:

$$h(\mathbf{x}) = \begin{cases} y & x_i \geq k \\ -y & o.w. \end{cases}$$

or

$$h(\mathbf{x}) = \begin{cases} y & x_i \leq k \\ -y & o.w. \end{cases}$$

where y can be either $+1$ or -1 and k can be any real number.

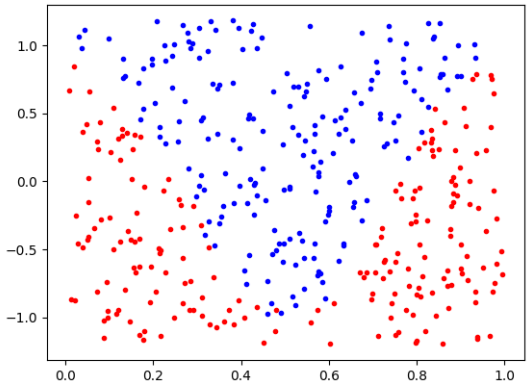

Figure 3: Dataset 2

An outline of AdaBoost algorithm is provided below.

Given: $(x_1, y_1), \cdots, (x_m, y_m)$ where $x_i \in X$, $y_i \in Y = \{-1, +1\}$

Initialize $D_1(i) = \frac{1}{m}$.

For $t = 1, \cdots, T$:

- Select a weak classifier with smallest weighted error $h_t = \arg\min_{h \in \mathcal{H}} \sum_{i=1}^{m} D_t(i) \mathbb{1}_{\{y_i \neq h(x_i)\}}$, where $\mathcal{H}$ is the set of all possible weak learners.
- Compute

$$\epsilon_t = \Pr_{i \sim D_t}(y_i \neq h_t(x_i)) = \sum_{i=1}^{m} D_t(i) \mathbb{1}_{\{y_i \neq h_t(x_i)\}}$$

- Compute

$$\alpha_t = \frac{1}{2} \log(\frac{1 - \epsilon_t}{\epsilon_t})$$

- Update

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalization factor.

Output the final classifier

$$H(x) = \text{sign}(\sum_{t=1}^{T} \alpha_t h_t(x))$$

.

2.1 Programming [20 pts]

In this section, you will need to implement the AdaBoost algorithm with decision stump as base learners on synthetic datasets. You will be provided with the `adaboost.py` where there are:

- A code skeleton that load two datasets and test your model on the data. The generated datasets have two entries X and y, where $X_{i,j} \in [-2, 2]$ corresponds to the j-th feature of the i-th sample and $y_i \in \{-1, 1\}$ corresponds to the label of the i-th sample.

- A helper function `plot_func` that is used to plot the synthetic datasets.

The two datasets are shown in Fig. 2 and Fig. 3.

You are free to design your own code, but you are required to implement the function `adaboost(X_train, y_train, X_test, num_iter)` that returns a vector `pred`.

Here, `X_train` is $m \times 2$ matrix where row i corresponds to the feature values of the i-th sample; `y_train` is a length m vector where the i-th entry corresponds to the label of the i-th sample; `X_test` is $n \times 2$ matrix where row i corresponds to the feature values of the i-th sample. m is the size of training set, n is the size of testing set; `num_iter` is the number of iterations your AdaBoost algorithm will run. The output `pred` should be a length n vector where the i-th entry is your label prediction on the ith test sample. Note that your prediction needs to be either $+1$ or -1.

Autograding The accuracy of our implementation is 0.985 on dataset 1 and 0.935 on dataset 2. You will get full score if your accuracy is above 0.96 on dataset 1 and above 0.90 on dataset 2.

Recommendations on Implementation:

- Start with a base learner function that takes in training data and weights of each sample, and outputs the specifications of the best decision stump under the current AdaBoost distribution D_t, e.g. the feature index that your decision stump is separating (x_1 or x_2), the position of the decision stump, and to which side your decision stump will label as 1.
- Write a function to update the AdaBoost distribution after each iteration.
- Write a function to predict the labels y given matrix X. This function should take in the decision stumps that you learned in previous iterations.

2.2 Report Questions [20 pts]

Using the functions you wrote, do the following experiments on **dataset 1** and report the results.

1. [5 pts] At each iteration t, compute the weighted error rate ϵ_t of your weak learner h_t on the training set. For $T = 20$, plot ϵ_t as a function of t. What is the highest weighted error rate a weak learner ever achieves?
2. [5 pts] At each iteration t, compute the error rate of your combined learner H_t on the training set. For $T = 20$, plot your training error as a function of current iteration number t.
3. [5 pts] At each iteration t, compute the error rate of your combined learner H_t on the testing set. For $T = 20$, on the same graph as the previous question, plot your testing error as a function of current iteration number t.
4. [5 pts] Set $T = [20, 50, 100, 200, 500, 1000, 2000, 4000]$ respectively. Plot the training error and the test error. Does the training error decrease with the number of iterations? Does the testing error increase with the number of iterations?

3 Submission Instructions

Please submit your codes for programming in a tar file and submit to Autolab. Make sure you put all your files in a directory called `src`, and run the following command in its top-directory enclosing the folder:

```
tar cvf src.tar src
```

Then submit src.tar