10-601 Machine Learning: Homework 5

Due 5:30 p.m. Thursday, March 31, 2016
TAs: Travis Dick and Han Zhao

Instructions

- **Late homework policy:** Homework is worth full credit if submitted before the due date, half credit during the next 48 hours, and zero credit after that. You *must* turn in at least $n-1$ of the n homeworks to pass the class, even if for zero credit.
- **Collaboration policy:** Homeworks must be done individually, except where otherwise noted in the assignments. "Individually" means each student must hand in their own answers, and each student must write and use their own code in the programming parts of the assignment. It is acceptable for students to collaborate in figuring out answers and to help each other solve the problems, though you must in the end write up your own solutions individually, and you must list the names of students you discussed this with. We will be assuming that, as participants in a graduate course, you will be taking the responsibility to make sure you personally understand the solution to any work arising from such collaboration.
- **Submission:** You must submit your solutions on time BOTH electronically by submitting to autolab AND by dropping off a hardcopy in the bin outside Gates 8221 by 5:30 p.m. Thursday, March 31, 2016. We recommend that you use LaTeX, but we will accept scanned solutions as well. On the Homework 5 autolab page, you can click on the "download handout" link to download the submission template, which is a tar archive containing a blank placeholder pdf for your written solutions, and an Octave `.m` file for each programming question. Replace each of these files with your solutions for the corresponding problem, create a new tar archive of the top-level directory, and submit your archived solutions online by clicking the "Submit File" button.

  DO NOT change the name of any of the files or folders in the submission template. In other words, your submitted files should have exactly the same names as those in the submission template. Do not modify the directory structure.

Problem 1: k-means Clustering (Travis) [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.

Solution: There are $2^n/2$ ways to partition a set of n points into 2 clusters and we do n work for each way. Therefore, the running time is $O(2^n n)$.

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 `[a] = update_assignments(X, C, a)`. The input `X` is the $n \times d$ data matrix, `C` is the $k \times d$ matrix of current centers, and `a` is the $n \times 1$ vector of current cluster assignments. That is, `C(i,:)` is the center for cluster i and the j^{th} data point, `X(j,:)`, is assigned to cluster `a(j)`. Your function should output a new $n \times 1$ vector of cluster assignments so that each point is assigned to the cluster with the nearest center.

Solution:

```
function a = update_assignments(X, C, a)
  % Compute the matrix of all pairwise distances between rows of X and C
  n = size(X, 1);
```

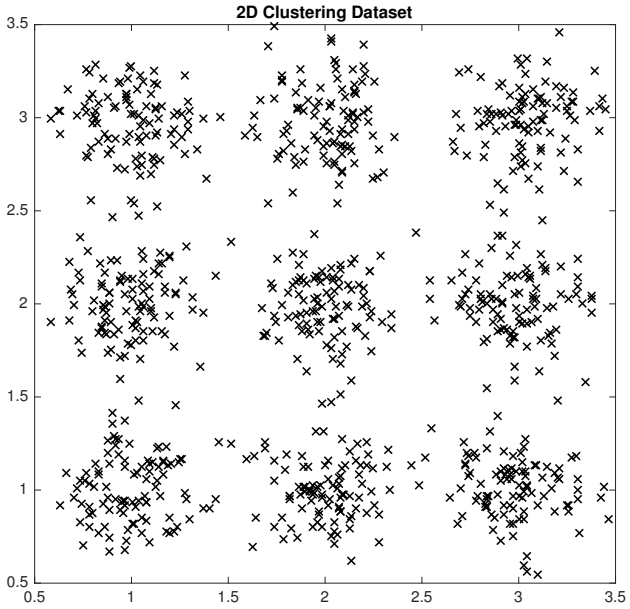


Figure 1: The 2D dataset used for problem 1.

```
  k = size(C, 1);
  D = zeros(n, k);
  for i = 1:k
    D(:, i) = sum(bsxfun(@minus, X, C(i,:)) .^2, 2);
  end
  % Update the assignments vector to be the argmin of each row of D.
  [~, a] = min(D, [], 2);
end
```

(c) [8 pts] Complete the function `[C] = update_centers(X, C, a)`. 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.

Solution:

```
function C = update_centers(X, C, a)
  k = size(C, 1);
  for i = 1:k
    % Set C(i,:) to be the mean of those rows of X for which a == i.
    C(i,:) = mean(X(a == i,:), 1);
  end
end
```

(d) [8 pts] Complete the function `[C, a] = lloyd_iteration(X, C)`. This function takes a data matrix `X`, initial centers `C` 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 $n \times 1$ vector `a` of assignments.

Solution:

```
function [C, a] = lloyd_iteration(X, C)
  a = zeros(size(X,1), 1);
  while 1
    % Compute the new assignments
    a_new = update_assignments(X, C, a);
    % If none of them have changed since the last iteration then halt
    if all(a_new == a)
      break
    end
    % Update the assignments
    a = a_new;
    % Update teh centers
    C = update_centers(X, C, a);
  end
end
```

(e) [8 pts] Complete the function `[obj] = kmeans_obj(X, C, a)`. This function takes the $n \times d$ data matrix `X`, a $k \times d$ matrix `C` of centers, and a $n \times 1$ vector `a` of cluster assignments. Your function should output the value of the k-means objective of the provided clustering.

Solution:

```
function obj = kmeans_obj(X, C, a)
  k = size(C, 1);
  obj = 0.0;
  for i = 1:k
    % Compute X(j,:) - C(i,:) for each row of X with a(j) == i.
    deltas = bsxfun(@minus, X(a == i, :), C(i, :));
    % Add the sum of squared norms of the rows of delta to the objective.
    obj = obj + sum(sum(deltas.^2));
  end
end
```

We provide you with a function `[C, a, obj] = 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++.

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

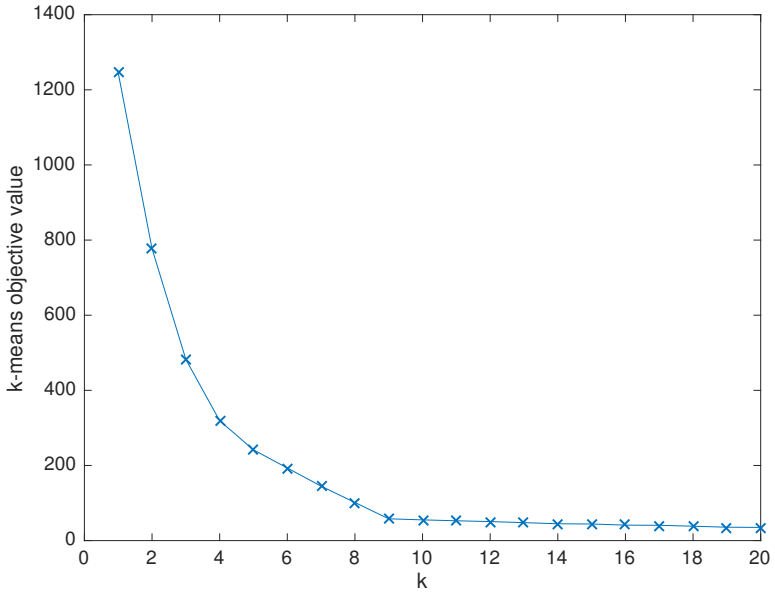
In parts (f) and (g) of this problem you will investigate the effect of the number of clusters k on Lloyd's method and a simple heuristic for choosing a good value for k. You can load the data for this problem by running the command `load kmeans_data` in Matlab or Octave. This will introduce a 900×2 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.

Solution:

```
load kmeans_data
objs = [];
```

```
for k = 1:20
  [~, ~, obj] = kmeans_cluster(X, k, 'random', 10);
  objs = [objs, obj];
end
plot(objs, 'x-')
```



(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?

Solution: The elbow in the plotted curve occurs at $k = 9$. Looking at the plot of the data, we see that there are 9 reasonably well separated Gaussian clusters. So the value of $k = 9$ matches the intuition.

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.

Solution:

```
load kmeans_data

num_trials = 1000;
for init = {'random', 'kmeans++'}
  total_obj = 0;
```

```
  for i = 1:num_trials
    [~, ~, obj] = kmeans_cluster(X, 9, init{1}, 1);
    total_obj = total_obj + obj;
  end
  fprintf('Average for init=%s: %f\n', init{1}, total_obj / num_trials)
end
```

The output of the above code gives average objective value of 89.2 for random initialization and 77.4 for k-means++ initialization. These values are random, so we will accept any output for which the random initialization objective is approximately 90 and the k-means++ initialization objective is approximately 78.

(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.

Solution: When using k-means++ initialization, each point is chosen to be the j^{th} center with probability proportional to its squared distance to the first $j-1$ centers. For this dataset, once a center has been chosen from a given Gaussian cluster, points belonging to any other Gaussian cluster will be further away than points in the same Gaussian cluster, so k-means++ will be more likely to sample them.

Problem 2: Disagreement-based Active Learning (Han) [40 pts]

T or F [10 pts]

Answer the following questions. If your answer is **T**, provide 1-2 sentences of justification. If your answer is **F**, give a counter-example if possible. Recall from the lecture notes that a classification problem is **realizable** if the true labeling function h^* is in our hypothesis class H. Given a set of labeled examples $\{(x_t, y_t)\}_{t=1}^n$, where $y_t = h^*(x_t), \forall t = 1, \ldots, n$, the version space $H' \subseteq H$ is a subset of H in which the hypotheses are consistent with the labels seen so far. The region of disagreement with respect to a version space H', denoted as $\text{DIS}(H')$, is defined to be the part of data space about which there is still some uncertainty. More specifically, $\text{DIS}(H') = \{x \in X : \exists h_1, h_2 \in H', h_1(x) \neq h_2(x)\}$.

(a) Assume the problem is realizable. Given a concept class H and a sequence of training instances $\{(x_t, y_t)\}_{t=1}^n$. Let $H_t \subseteq H$ be the version space after the learner has received the t-th instance. Then $H_t \neq \varnothing$, $\forall t = 1, \ldots, n$.

Solution: T. Since the problem is realizable, then the true labeling function $h^* \in H$ will be consistent with every instance seen so far, which implies that $h^* \in H_t$. Hence $H_t \neq \varnothing$.

(b) Assume the problem is realizable. Given a concept class H and a sequence of training instances $\{(x_t, y_t)\}_{t=1}^n$. Let $H_t \subseteq H$ be the version space after the learner has received the t-th instance. After all n examples have been observed we have $|H_n| = 1$.

Solution: F. Let $H = \{[a, \infty) : a \in \mathbb{R}\}$ and for all the instances $\{(x_t, y_t)\}_{t=1}^n$ we have $y_t = +1$. Let $x_0 = \min_t x_t$, then $H_n = \{[a, \infty) : a \leq x_0\}$ where $|H_n| = \infty > 1$.

(c) Let $X = \{(x_t, y_t)\}_{t=1}^n$ be a set of n training instances and H be the hypothesis space. Assume our current version space is H'. Pick $x_0 \in \text{DIS}(H')$. Let $\hat{H}$ be the version space of X after querying x_0, i.e., the version space of $X \cup \{(x_0, y_0)\}$, where y_0 is provided by the true labeling function, then it must be that $\hat{H} \neq H'$.

Solution: T. For $\forall x_0 \in \text{DIS}(H'), \exists h_1, h_2 \in\in H', h_1(x) \neq h_2(x)$. So if we query x_0 and make a refinement to H' based on the query result, then at least one of h_1 and h_2 will be removed from the current version space, hence $\hat{H}$ will be a proper subset of H'.

An Example [30 pts]

Consider a binary classification problem where the sample space is $\Omega = \{(\mathbf{x}, y) : ||\mathbf{x}||_2 \leq 2, y \in \{+1, -1\}\}$. Let the concept class $H = \{\mathbf{w} : \mathbf{w} \in \mathbb{R}^2, ||\mathbf{w}||_2 = 1\}$, i.e., the set of linear classifiers where the decision rule is given by:

$$h_{\mathbf{w}}(\mathbf{x}) = \begin{cases} +1 & \mathbf{w}^T\mathbf{x} \geq 0 \\ -1 & \mathbf{w}^T\mathbf{x} < 0 \end{cases}$$

In this problem we assume that H is realizable. Suppose after receiving a sequence of training instances we know that the entire sector AOB shown in Fig. 2 is +1 and the entire sector COD is -1. Both sectors AOB and COD are symmetric with respect to the x_1-axis and $\angle AOB = \angle COD = \frac{\pi}{3}$.

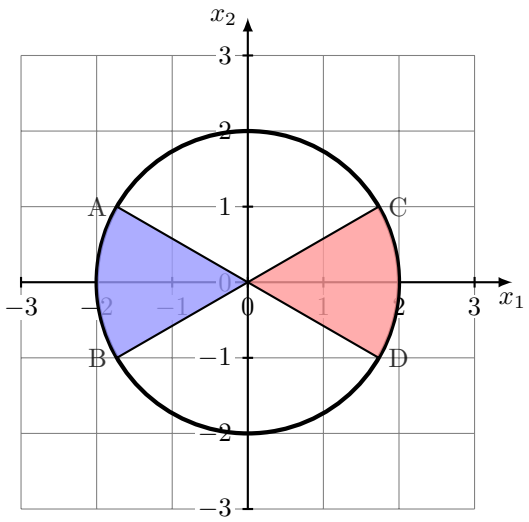


Figure 2: All hypotheses in the version space label the blue region as +1 and the red region as -1.

(d) Explicitly write down the version space $H_1 \subseteq H$ that is consistent with all the instances in sectors AOB and COD and draw H_1 in Fig. 2. Specifically, in Fig. 2, shade in the region that corresponds to the set of vectors $\mathbf{w}$ such that $h_{\mathbf{w}} \in H_1$.

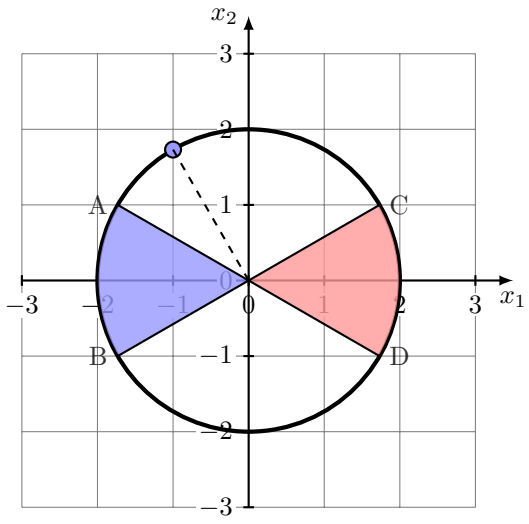
Solution: $H_1 = \{\mathbf{w} : \mathbf{w} = (\cos\theta, \sin\theta), \frac{2\pi}{3} \leq \theta \leq \frac{4\pi}{3}\}$.

(e) Suppose we receive a new instance $(\mathbf{x}, y) = ((-1, \sqrt{3}), +1)$. Write down the new version space $H_2 \subseteq H$ that is consistent with all the instances in both sectors as well as the new instance. Draw H_2 in Fig. 3a.

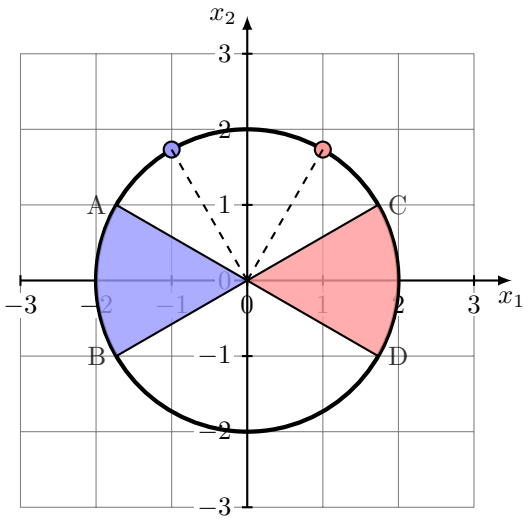
Solution: $H_2 = \{\mathbf{w} : \mathbf{w} = (\cos\theta, \sin\theta), \frac{2\pi}{3} \leq \theta \leq \frac{7\pi}{6}\}$.

(f) This time another new instance with negative label $(\mathbf{x}, y) = ((1, \sqrt{3}), -1)$ has been received. Write down the new version space $H_3 \subseteq H$ that is consistent with all the instances in both sectors along with the two new instances. Draw H_3 in Fig. 3b.

Solution: $H_3 = \{\mathbf{w} : \mathbf{w} = (\cos\theta, \sin\theta), \frac{5\pi}{6} \leq \theta \leq \frac{7\pi}{6}\}$.



(a) A new instance $((-\frac{1}{2}, \frac{\sqrt{3}}{2}), +1)$ is received.



(b) Another new instance $((1, \sqrt{3}), -1)$ is received.

Figure 3

Problem 3: Parity Functions (Han) [Extra Credit] [20 pts]

Let the domain set X be the Boolean hypercube $\{0,1\}^n$, i.e., $\forall x \in X$, x is an n-bit binary string. For a set $S \subseteq \{1, 2, \ldots, n\}$ we define a parity function $h_S(x)$ as follows:

$$h_S(x) = \left(\sum_{i \in S} x_i \right) \bmod 2$$

That is, h_S computes the parity of bits in S. Define the concept class $H_{\textbf{parity}} = \{h_S : S \subseteq \{1, 2, \ldots, n\}\}$.

VC-dimension of Parity Functions [10 pts]

In this problem we will calculate the VCdim of $H_{\textbf{parity}}$ by computing a matching upper and lower bound.

(a) Show that $\text{VCdim}(H_{\textbf{parity}}) \leq n$.

Solution: Note that $H_{\textbf{parity}}$ is a finite concept class with 2^n elements, hence we have $\text{VCdim}(H_{\textbf{parity}}) \leq \log_2 |H_{\textbf{parity}}| = n$.

(b) Show that there exists a finite set of n-bit binary string with size n that can be shattered by $H_{\textbf{parity}}$ and conclude that $\text{VCdim}(H_{\textbf{parity}}) = n$.

Solution: Let $H_n = \{u_t : t \in [n]\}$ where each u_t is a binary string with all 0s except the t-th bit. It can be shown that any subset H of H_n can be shattered by $H_{\textbf{parity}}$ by setting $S = \{t : u_t \in H\}$.

Active vs Passing Learning [10 pts]

In this problem we will explore how active learning can drastically change the number of examples needed to learn a concept. More specifically, we will design an active learning algorithm where the learner has the luxury of specifying a data point and querying its label, and compare it with a passive learning algorithm where labeled examples are drawn at random. Note that for this problem we are in a realizable case where the consistent labeling function is in $H_{\textbf{parity}}$.

(c) In the active learning setting, the learning algorithm can query the label of an unlabeled example. Assume that you can query any possible example. Show that, starting with a single example, you can exactly learn the true hypothesis h_{S^*} using n queries.

Solution: For any n-bit binary string x with label y, let $x_t, t \in [n]$ be the copy of x and then take the complement at the t-th bit. Query a new label y_t for x_t. If $y_t = y$, then $t \notin S$ otherwise $t \in S$. Hence we only need n exact queries to determine the consistent labeling function h_{S^*}.

(d) In the passive learning setting, the examples are drawn i.i.d from a uniform distribution over all the possible 2^n binary strings. According to PAC learning theory, how many examples (in big-O notation) are required to guarantee a generalization error less than ε with probability $1 - \delta$?

Solution: The result follows directly from the sample complexity in the realizable case: $O((n \log(1/\varepsilon) + \log(1/\delta))/\varepsilon)$.

Problem 4: k-means on the real line (Travis) [Extra Credit] [20 pts]

In this problem you will show that the k-means objective (1) can be minimized in polynomial time when the data points are single dimensional ($d = 1$), despite the fact that in general finding the optimal centers is NP-hard.

(a) [5 pts] Consider the case where $k = 3$ and we have 4 data points $x_1 = 1$, $x_2 = 3$, $x_3 = 6$, $x_4 = 7$. What is the optimal clustering for this data? What is the corresponding value of the objective (1).

Solution: The optimal centers are $c_1 = 1$, $c_2 = 3$, and $c_3 = 6.5$. The cost is $0+0+(0.5)^2+(0.5)^2 = 0.5$.

(b) [5 pts] One might be tempted to think that Lloyd's method is guaranteed to converge to the global minimum when $d = 1$. Show that there exists a suboptimal cluster assignment for the data in part (a) that Lloyd's algorithm will not be able to improve (to get full credit, you need to show the assignment, show why it is suboptimal *and* explain why it will not be improved).

Solution: $c_1 = 2$, $c_2 = 6$, $c_3 = 2$. Points x_1 and x_2 are closest to center c_1, point x_3 is closest to center c_2, and point x_4 is closest to c_3. Computing the mean of each cluster to update the centers leaves them unchanged and the assignments will also be unchanged, so Lloyd's method has converged. The cost of this set of centers is $1^2 + 1^2 + 0 + 0 = 2$.

(c) [5 pts] Assume we sort our data points such that $x_1 \leq x_2 \leq \cdots \leq x_n$. Prove that an optimal cluster assignment has the property that each cluster corresponds to some interval of points. That is, for each cluster j, there exits i_1 and i_2 such that the cluster consists of $\{x_{i_1}, x_{i_1+1}, \ldots, x_{i_2}\}$.

Solution: We saw in class that for a given set of centers, the optimal cluster assignment is to assign each point to its nearest center. This implies that if the optimal assignment assigns point x to the center c, then every point between x and c must also be assigned to c, since otherwise c could not be the closest center to x. Now let x_i and x_j be the smallest and largest points assigned to center c. Then every point between them is also assigned to center c, and therefore the points assigned to c form an interval. Since this is true for all centers c, this implies that all clusterings are intervals. In particular, this implies that for the optimal centers, the resulting clusters are intervals.

(d) [5 pts] Develop an $O(kn^2)$ dynamic programming algorithm for single dimensional k-means. (Hint: from part (c), what we need to optimize are $k - 1$ cluster boundaries where the i^{th} boundary marks the largest point in the i^{th} cluster.)

Solution: For any $k' \leq k$ and $n' \leq n$, let $Q(k', n')$ be the k-means cost of the optimal clustering of the first n' points using k' clusters. For indices $i \leq j$, also define

$$c(i,j) = \sum_{t=i}^{j} (x_t - \mu(i,j))^2 \qquad \text{where} \qquad \mu(i,j) = \frac{1}{j-i+1} \sum_{t=i}^{j} x_t$$

to be the k-means cost of a cluster containing points $x_i, x_{i+1}, \ldots, x_j$.

Our goal is to calculate $Q(k, n)$ and the corresponding optimal clustering. For the case of $k' = 1$, we know that the optimal clustering simply puts all points in the same cluster, which gives the following base case:

$$Q(1, n') = c(1, n') \text{ for all } n' \leq n.$$

For values of k' bigger than 1, we know that the optimal clustering of the first n' points can be constructed by taking the optimal clustering for some prefix of $m < n'$ of the points with $k' - 1$ clusters and setting the final cluster to contain the points with indices $m + 1$ through n' (this is where we used the fact that the clusters are all intervals). This gives us the following recurrence relation on Q:

$$Q(k', n') = \min_{1 \leq m < n'} Q(m, k' - 1) + c(m + 1, n') \text{ for all } k' > 1 \text{ and } n' \leq n$$

We can use the above observations to efficiently compute the value of $Q(k, n)$ as follows: imagine a table with k rows and n columns, where the $(k', n')^{\text{th}}$ entry of the table contains $Q(k', n')$. We can fill in the first row of the table using the base case for $Q(1, n')$. Once row k' has been filled in, we can fill in row $k' + 1$ using the recurrence relation. We need to fill in $O(nk)$ table entries, and filling in each entry requires us to evaluate the c function $O(n)$ times (i.e., we loop over all the values $m < n'$ and for each value we lookup an entry in the table and compute the value of $c(m + 1, n')$). If we recompute $c(i, j)$ each time we need it, the total running time is $O(kn^3)$, since we we have $O(nk)$ table entries to fill, each involves $O(n)$ evaluations of c, and each evaluation of c costs $O(n)$ time.

We can reduce the running time to $O(kn^2)$ by precomputing the values of $c(i, j)$ using the following trick. By expanding the square in the definition of $c(i, j)$, we have the following alternative expression:

$$c(i, j) = \left(\sum_{t=i}^{j} x_t^2 \right) + \frac{1}{j - i + 1} \left(\sum_{t=i}^{j} x_t \right)^2 - 2 \left(\sum_{t=i}^{j} x_t \right) \cdot \left(\frac{1}{j - i + 1} \sum_{t=i}^{j} x_t \right).$$

The key observation is that given the two sums $\sum_{t=i}^{j} x_t$ and $\sum_{t=i}^{j} x_t^2$, we can compute $c(i, j)$ in constant time. So, imagine filling in an $n \times n$ table where the $(i, j)^{\text{th}}$ entry contains $c(i, j)$. We only need to fill in the entries where $i \leq j$. Moreover, we can fill in each row i from left to right in $O(n)$ by keeping track of a running sum for $\sum_{t=i}^{j} x_i$ and $\sum_{t=i}^{j} x_i^2$. The total cost of precomputing c in this way is $O(n^2)$. Combining this with the above dynamic programming results in running time $O(kn^2)$, as required.

Finally, we can recover the optimal clustering from the above dynamic programming algorithm by remembering which value of m minimized the recurrence relation in each entry of the able. Starting from $Q(k, n)$, we can follow the minimizing values of m back through the table to find the break points between each of the interval clusters.