CS 475 Machine Learning: Homework 4
Clustering, EM, Dimensionality Reduction, Graphical Models

Due: Friday April 27, 2018, 11:59pm

125 Points Total Version 1.0

Make sure to read from start to finish before beginning the assignment.

1 Programming (65 points)

In this assignment you will implement two variants of the K-means clustering algorithm, λ-means clustering and the stochastic k-means algorithm.

1.1 λ-Means

The K-means clustering algorithm groups instances into K clusters. Each cluster is represented by a prototype vector μ_k, which represents the mean of the examples in that cluster. Cluster assignments are "hard," meaning that an instance can belong to only a single cluster at a time.

The K-means algorithm works by assigning instances to the cluster whose prototype μ_k has the smallest distance to the instance (based on, for example, Euclidean distance). The mean vectors μ_k are then updated based on the new assignments of instances to clusters, and this process is repeated using an EM-style iterative algorithm.

A limitation of K-means is that we must choose the number of clusters K in advance, which can be difficult in practice. There is a variant of K-means called λ-means where the number of clusters can change as the algorithm proceeds. This algorithm is very similar to K-means but with one key difference: an instance $\mathbf{x}_i$ is assigned to the nearest existing cluster **unless** all of the cluster prototypes have a distance larger than some threshold λ. In that case, $\mathbf{x}_i$ is assigned to a new cluster (cluster $K + 1$ if we previously had K clusters). The prototype vector for the new cluster is simply the same vector as the instance, $\mu_{K+1} = \mathbf{x}_i$. The idea here is that if an instance is not similar enough to any of the existing clusters, we should start a new one.[1]

1.1.1 Implementation

Your `LambdaMeans` class should inherit from `Model`, as you have done all semester. You will still implement the `fit` and `predict` methods. However, the behavior will be slightly different than in previous projects.

In unsupervised learning, the learner does not have access to labeled data. However, the datasets you have been using contain labels in the training data. You should not use these labels at all during learning. Additionally, since the clustering algorithm cannot read the correct labels, we must be clear about what "label" you are returning with `predict`.

Your implementation should include the following functionality:

[1]This algorithm (called "DP-means" here) is described in: B. Kulis and M.I. Jordan. Revisiting k-means: New Algorithms via Bayesian Nonparametrics. 29th International Conference on Machine Learning (ICML), 2012.

- The `fit` method should learn the cluster parameters based on the training examples. While the labels are available in the provided data, the clustering algorithms should not use this information. The result of the `fit` method are the cluster parameters: the means μ_k and the number of clusters K.
- The `predict` method should label the examples by assigning them to the closest cluster. The value of the label should be the cluster index, i.e., an example closest to kth cluster should receive label k. While this document will describe the algorithm as if k is in the set $\{1, \ldots, K\}$, it is fine for your code to use the set $\{0, \ldots, K-1\}$ because the evaluation scripts (see Section 1.8) will give the same results whether you use 0-indexing or 1-indexing.

1.2 Inference with EM

The K-means algorithm and the λ-means algorithm are based on an EM-style iterative approach. On each iteration, the algorithm computes 1 E-step and 1 M-step.

In the E-step, cluster assignments r_{nk} are determined based on the current model parameters μ_k. r_{nk} is an **indicator** variable that is 1 if the nth instance belongs to cluster k and 0 otherwise.

Suppose there are currently K clusters. You should set the indicator variable for these clusters as follows. For $k \in \{1, \ldots, K\}$:

$$r_{nk} = \begin{cases} 1 & \text{if } \; k = \arg\min_j ||\mathbf{x}_n - \mu_j||_2 \text{ and } \min_j ||\mathbf{x}_n - \mu_j||_2 \leq \lambda \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where $||\mathbf{x} - \mathbf{x}'||_2$ denotes the Euclidean distance, $\sqrt{\sum_{f=1}^{m}(x_f - x'_f)^2}$. When computing this distance, assume that if a feature is not present in an instance $\mathbf{x}$, then it has value 0.

Additionally, you must consider the possibility that the instance $\mathbf{x}_n$ is assigned to a new cluster, $K+1$. The indicator for this is:

$$r_{n,K+1} = \begin{cases} 1 & \text{if } \; \min_j ||\mathbf{x}_n - \mu_j||_2 > \lambda \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

If $r_{n,K+1} = 1$, you should immediately set $\mu_{K+1} = \mathbf{x}_n$ and $K = K+1$, before moving on to the next instance $\mathbf{x}_{n+1}$. You should not wait until the M-step to update μ_{K+1}.

Be sure to iterate through the instances in the order they appear in the dataset.

After the E-step, you will update the mean vectors μ_k for each cluster $k \in \{1, \ldots, K\}$ (where K may have increased during the E-step). This forms the M-step, in which the model parameters μ_k are updated using the new cluster assignments:

$$\mu_k = \frac{\sum_n r_{nk}\mathbf{x}_n}{\sum_n r_{nk}} \tag{3}$$

This process will be repeated for a given number of iterations (see Section 1.6).

1.3 Cluster Initialization

As we discussed in class, clustering objectives are non-convex. Therefore, different initializations will lead to different clustering solutions. In order for us to test the submitted code, we must have everyone use the same initialization method.

In K-means, the standard initialization method is to randomly place each instance into one of the K clusters. However, in λ-means you can initialize the algorithm with only one cluster ($K = 1$). You should initialize the prototype vector to the mean of all instances: $\mu_1 = \bar{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}_n$.

1.4 Lambda Value

The default value of λ will be the average distance from each training instance to the mean of the training data:

$$\lambda^{(\text{default})} = \frac{1}{N}\sum_{i=1}^{N} ||\mathbf{x}_i - \bar{\mathbf{x}}||_2, \tag{4}$$

where $\bar{\mathbf{x}}$ is the mean vector of the training instances (also the initialization of μ_1 in 1.3).

You *must* add a command line argument to allow this value to be adjusted via the command line. Do this by adding the following to the `get_args` function in `classify.py`:

```
parser.add_argument("--cluster_lambda", type=float,
help="The value of lambda in lambda-means", default=0.0)
```

How you implement the λ value is up to you, but it probably makes more sense to set its value in your `LambdaMeans` rather than in `get_args`. You can do this by setting the input value to 0 by default (as in the above code example), then setting the value to the default in Eq. 4 if the value passed to the `fit` function is 0. You can assume that any value we supply through the command line will be > 0.

We highly encourage you to experiment with different values of λ to see how it affects the number of clusters that are produced. In particular, think about how to choose λ in order to have K match the true number of classes in the dataset.

1.5 Stochastic K-means

The Stochastic K-means algorithm (SKA) associates the data point $\mathbf{x}_i$ to a cluster based on a probability distribution that depends on the distance between the data point and the cluster centers. SKA is less dependent than K-means on the initial cluster choices.

The cluster initialization is the same as that of the K-means algorithm; i.e. typically done randomly. Because we want to standardize results and need large spacing, let's do the following. For $K = 1$, initialize by taking the mean of the data (as in Section 1.3). For larger K, choose centers that divide the range of each dimension into $K - 1$ equal pieces. For instance, with $K = 2$, choose one center with all of the minima and one center with all of the maxima. For $K = 3$, choose a center with all maxima, one with all minima, and one where $x_{c,i} = (\max(x_i) + \min(x_i))/2,\ \forall i$. For $K = 4$, choose the maxima, the minima, and prototypes at $x_{c1,i} = 1/3 * \max(x_i) + 2/3 * \min(x_i),\ \forall i$ and $x_{c2,i} = 2/3 * \max(x_i) + 1/3 * \min(x_i),\ \forall i$. Continue this way for larger K.

At the beginning of each iteration, assign each data point $\mathbf{x}_n$ to the closest mean. You should set the indicator variable for the K clusters as follows. For $k \in \{1, \ldots, K\}$,

$$r_{nk} = \begin{cases} 1 & \text{if } \ k = \arg\min_j d(\mathbf{x}_n, \mu_j) \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

The probability that $\mathbf{x}_n$ belongs to cluster k is defined as follows:

$$p_{nk}^{(i)} = \frac{\exp\left(\frac{-\beta^{(i)} d(x_n, \mu_k)}{\hat{d}(x_n)}\right)}{\sum_{j=1}^{K} \exp\left(\frac{-\beta^{(i)} d(x_n, \mu_j)}{\hat{d}(x_n)}\right)} \tag{6}$$

where $d(\mathbf{x}_n, \mu_k)$ is the Euclidean distance between $\mathbf{x}_n$ and μ_k and $\hat{d}(\mathbf{x}_n)$ is the mean Euclidean distance between $\mathbf{x}_n$ and all μ_j, $j \in \{1, \ldots, K\}$

$$\hat{d}(\mathbf{x}_n) = \frac{1}{K} \sum_{j=1}^{K} ||\mathbf{x}_n - \mu_j||_2 \tag{7}$$

The model parameters μ_k should then be updated as

$$\mu_k = \frac{\sum_n p_{nk} \mathbf{x}_n}{\sum_n p_{nk}} \tag{8}$$

To implement this, you should create another class that inherits from the `Model` class called `StochasticKMeans`. Note that you will not use a λ value for this algorithm. Instead, please add a command line argument to allow the value of K to be adjusted via the command line. Do this by adding the following to the `get_args` function in `classify.py`:

```
parser.add_argument("--number_of_clusters", type=int,
help="The number of clusters (K) to be used.")
```

* Since this was added late, we will only grade your SKA results for $K = 2$ on all data sets, except for the iris data set. On that one we are interested in $K = 3$. It may be interesting for you to see how the results vary with different K, however.

1.5.1 How to choose β

The parameter β increases linearly with iteration, as follows:

$$\beta^{(i)} = c * i, \tag{9}$$

where i is the iteration number and c is a constant. You may want to experiment with different values of c, but use $c = 2$ in your final implementation. Note that in the limit, as $\beta \to \infty$, SKA behaves similarly to the K-means algorithm.

1.6 Number of Iterations

The default number of clustering iterations is 10. An iteration is defined as computing 1 E-step and 1-M step of the algorithm (in that order).

You *must* add a command line argument to allow this value to be adjusted via the command line. Do this by adding the following to the `get_args` function in `classify.py`:

```
parser.add_argument("--clustering_training_iterations", type=int,
help="The number of training EM iterations")
```

1.7 Implementation Details

1.7.1 Tie-Breaking

When assigning an instance to a cluster you may encounter ties, where the instance is equidistant to two or more clusters. When a tie occurs, select the cluster with the lowest cluster index.

1.7.2 Empty Clusters

It is possible (though unlikely) that a cluster can become empty while training the algorithm. That is, there may exist a cluster k such that $r_{nk} = 0, \forall n$. In this case, you should set $\mu_k = \mathbf{0}$. Do not remove empty clusters.

1.7.3 Expanding the Cluster Set

In standard K-means, you typically create arrays of size K to store the parameters. Since the number of clusters can grow with λ-means, you will need to set up data structures that can accommodate this. One option is to simply use arrays of a fixed size, expanding the size of the array if you run out of space (by creating a larger array and copying the contents of the old array). Alternatively, you can expand the size of the data structure as you go along (though this will likely run slower).

1.8 Evaluation

For clustering we cannot simply use accuracy against true labels to evaluate the output, since your prediction outputs are simply the indices of clusters (which could be arbitrary). Instead we will evaluate your output using an information-theoretic metric called *variation of information* (VI), which can be used to measure the dependence of the cluster indices with the true labels. The variation of information between two random variables Y and $\hat{Y}$ is defined as: $H(Y|\hat{Y}) + H(\hat{Y}|Y)$. Lower is better. This will have a value of 0 if there is a one-to-one mapping between the two labelings, which is the best you can do. We have provided a python script to compute this metric: `cluster_accuracy.py`. Additionally, we have provided a script that displays the number of unique clusters: `number_clusters.py`.

We will evaluate your implementation on the standard data sets you have been using (`speech`, `bio`, etc.) as well as the three-class `Iris` dataset, included here. We have removed the `nlp` data set from this assignment for computational efficiency.

2 Analytical (60 points)

1) Clustering (14 points) We want to cluster the data set $\mathbf{x}_1, \dots, \mathbf{x}_n$ using the K-means algorithm. The K-means algorithm partitions the n observations into k sets ($k < n$) $\mathbf{S} = \{S_1, S_2, \dots, S_k\}$ so as to minimize the within-cluster sum of squares:

$$\underset{\mathbf{S}=\{S_1,\dots,S_k\}}{\text{argmin}} \sum_{j=1}^{k} \sum_{\mathbf{x}_i \in S_j} \|\mathbf{x}_i - \mu_j\|_2^2$$

where μ_j is the mean of the examples in S_j.

(a) Prove that the objective is non-increasing after each E/M step.

(b) One variant is called K-medoids. K-medoids is similar to K-means: both K-means and K-medoids minimize the squared error. However, unlike K-means, K-medoids chooses a training example as a cluster center (medoid). It is more robust to noise and outliers as compared to k-means because it minimizes a sum of pairwise dissimilarities instead of a sum of squared Euclidean distances.

Give an example to illustrate that K-medoids will be more robust to outliers than K-means.

(c) Another variant of K-means is to change the distance metric to the L1 distance, which is more robust to outliers. In this case, we are minimizing:

$$\min_{S=\{S_1,\ldots,S_k\}} \sum_{j=1}^{k} \sum_{\mathbf{x}_i \in S_j} \|\mathbf{x}_j - \mu_j\|_1 \tag{10}$$

where μ_j is the center of S_j w.r.t. to L1 distance. It turns out that computing the medians is involved when computing the cluster center. That is why this algorithm is call K-medians. Does the cluster center have to be an actual instance from the dataset? Explain why.

2) Expectation-Maximization (12 points) As we discussed in class, clustering objectives are non-convex. Therefore, different initializations will lead to different clustering solutions.

As an example, take Gaussian mixture model (GMM) clustering. Suppose all the parameters of the GMM are initialized such that all components/clusters have the same mean $\mu_k = \hat{\mu}$ and same covariance $\Sigma_k = \hat{\Sigma}$ for all $k = 1, \ldots, K$.

(a) Prove that the EM algorithm will converge after a single iteration for any choice of the initial mixing coefficients π.

(b) Show that this solution has the property $\mu_k = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}_n$ and $\Sigma_k = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \mu_k)(\mathbf{x}_n - \mu_k)^T$, where N is the total number of examples. Note that this represents a degenerate case of the mixture model in which all of the components are identical, and in practice we try to avoid such solutions by using an appropriate initialization.

3) Dimensionality Reduction (9 points) Show that in the limit $\sigma^2 \to 0$, the posterior mean for the probabilistic PCA model becomes an orthogonal projection onto the principal subspace, as in conventional PCA.

4) Probabilistic PCA (10 points) Draw a directed probabilistic graphical model representing a discrete mixture of probabilistic PCA models in which each PCA model has its own values of $\mathbf{W}$, $\boldsymbol{\mu}$, and σ^2. Then draw a modified graph in which these parameter values are shared between the components of the mixture. The graph should represent the model for a single data point $\mathbf{x}$. (Hint: refer to slide 24 of the Dimensionality Reduction lecture as a starting point.)

5) Graphical Models (15 points) Consider the Bayesian Network given in Figure 1. Are the sets $\mathbf{A}$ and $\mathbf{B}$ d-separated given set $\mathbf{C}$ for each of the following definitions of $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$? Justify each answer.

a. $\mathbf{A} = \{X_3\}$, $\mathbf{B} = \{X_4\}$, $\mathbf{C} = \{X_1, X_2\}$

b. $\mathbf{A} = \{X_5\}$, $\mathbf{B} = \{X_4\}$, $\mathbf{C} = \{X_2, X_7\}$

c. $\mathbf{A} = \{X_4\}$, $\mathbf{B} = \{X_6\}$, $\mathbf{C} = \{X_8\}$

d. $\mathbf{A} = \{X_5\}$, $\mathbf{B} = \{X_4\}$, $\mathbf{C} = \{X_2\}$

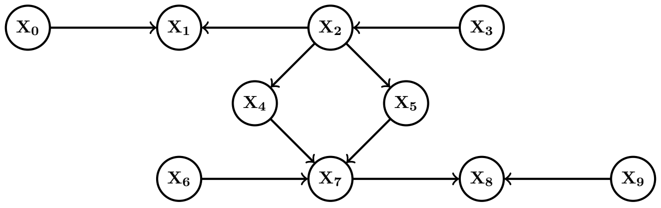

Figure 1: A directed graph

e. $\mathbf{A} = \{X_6, X_9\}$, $\mathbf{B} = \{X_8\}$, $\mathbf{C} = \{X_3, X_1\}$

Now assume that Figure 1 is a Markov Random Field, where each edge is undirected (just drop the direction of each edge.) Re-answer each of the above questions with justifications for your answers.

3 What to Submit

You will need to create an account on gradescope.com and signup for this class. The course is `https://gradescope.com/courses/13142`. Use entry code `974Z3W`. See this video for instructions on how to upload a homework assignment: `https://www.youtube.com/watch?v=KMPoby5g_nE`. In each assignment you will submit two things to gradescope.

1. **Submit your code (.py files) to gradescope.com as a zip file**. **Your code must be uploaded as code.zip with your code in the root directory**. By 'in the root directory,' we mean that the zip should contain `*.py` at the root (`./*.py`) and not in any sort of substructure (for example `hw4/*.py`). One simple way to achieve this is to zip using the command line, where you include files directly (e.g., `*.py`) rather than specifying a folder (e.g., `hw4`):

   ```
   zip code.zip *.py
   ```

   We will run your code using the exact command lines described earlier, so make sure it works ahead of time, and make sure that it doesn't crash when you run it on the test data. Remember to submit all of the source code, including what we have provided to you. We will include `requirements.txt` but nothing else.

2. **Submit your writeup to gradescope.com**. **Your writeup must be compiled from latex and uploaded as a PDF**. It should contain all of the answers to the analytical questions asked in the assignment. Make sure to include your name in the writeup PDF and to use the provided latex template for your answers.

4 Questions?

Remember to submit questions about the assignment to the appropriate group on Piazza: `https://piazza.com/class/jatranyaky957s`.