CS 475 Machine Learning: Homework 4
EM and Clustering
Due: Thursday Nov 3, 2016, 11:59pm
100 Points Total Version 1.0

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

1 Programming (60 points)

In this assignment you will implement λ-means clustering and Naive Bayes clustering.

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, e.g. 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 to choose in practice. There is a variant of K-means, which we call λ-means, in which the number of clusters can change as the algorithm proceeds. This algorithm is very similar to K-means, with one difference: when assigning an instance $\mathbf{x}_i$ to a cluster, we will assign it to the lowest-distance cluster **unless** all of the clusters have a distance larger than some threshold λ. In this case, we then assign $\mathbf{x}_i$ to a new cluster, which is denoted cluster $K+1$ if we previously had K clusters. The prototype vector for the new cluster will simply be the same vector as the instance, $\mu_{K+1} = \mathbf{x}_i$. The idea is that if an instance is not similar to any of the clusters, we should start a new one.[1]

The command line `algorithm` argument should be `lambda_means`.

1.1.1 Clustering as a `Predictor`

Your new class will implement the `train` and `predict` methods. However, the behavior will be slightly different than usual.

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 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 `train` 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 `train` 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 1.8) will give the same results whether you use 0-indexing or 1-indexing. You may only predict a cluster that was known during training. For an algorithm that can adjust the number of clusters based on the data, you should not increase the number of available clusters at prediction time.

1.2 Inference with EM

The K-means algorithm and the λ-means algorithm are based on an EM-style iterative algorithm. 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 $||\mathbf{x} - \mathbf{x}'||$ denotes the Euclidean distance, $\sqrt{\sum_{f=1}^{m}(x_f - x'_f)^2}$, and $||\mathbf{x} - \mathbf{x}'||^2$ is this distance squared. 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 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 to test submitted code, everyone must 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 simply initialize the algorithm with only one cluster, i.e. $K = 1$. 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 based on 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: `cluster-lambda`. Add this command line option by adding the following code 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 default value is up to you, but it probably makes more sense to set the default value of λ inside your new clustering method/class rather than `classify.py`. You can do this by setting the value to 0.0 by default (as in the above code example), then set the value to the default in Eq. 4 if the value passed to the constructor is 0.0. You can assume that any value we supply through the command line will be > 0.0.

We highly encourage you to experiment with different values of λ and see how it effects the number of clusters which are produced.

1.5 Naive Bayes Clustering

The second algorithm you will implement is unsupervised Naive Bayes. It is different from a supervised Naive Bayes classifier in that the label y is a latent variable. Therefore, we will apply the EM algorithm to learn in the absence of training data with y.

The command line `algorithm` argument will be `nb_clustering`.

1.5.1 Naive Bayes Generative Story

In Naive Bayes, the conditional likelihood of the example i given its label y_i factors into the distribution over its components (the naive assumption):

$$p(\mathbf{x}_i \mid y_i) \propto \prod_{j=1}^{M} p(x_{ij} \mid y_i) \tag{5}$$

The decision rule for Naive Bayes is:

$$\arg\max_{y_i} \; p(y_i \mid \mathbf{x}_i) = \arg\max_{y_i} \; p(y_i) \prod_{j=1}^{M} p(x_{ij} \mid y_i) \tag{6}$$

In this assignment we will assume a Naive Bayes model with K clusters (labels). For a given cluster k and with respect to each dimension of $\mathbf{x}_i$ we will assume:

$$p(x_{ij} \mid y_i = k) \sim Normal(\mu_j^k, \sigma_j^k), \tag{7}$$

i.e. each example is drawn from a cluster specific Normal (Gaussian) distribution. $p(y)$ is simply a categorical distribution:

$$p(y_i = k) = \phi^k \text{ where } \phi^k \geq 0 \text{ and } \sum_{k=1}^{K} \phi^k = 1 \tag{8}$$

1.5.2 Expectation Maximization

Your task will be to determine where the cluster centers (μ^k) are and which cluster each point i belongs to ($\arg\max_{y_i} \; p(y_i) \prod_{j=1}^{M} p(x_{ij} \mid y_i)$), where $y_i \in \{1, 2, \ldots, K\}$ is the index of the cluster that $\mathbf{x}_i$ belongs to. Note that y_i is a latent variable which is not observed during training. Therefore we will use the Expectation Maximization (EM) algorithm.

This is a *hard assignment* for each $\mathbf{x}_i$ rather than a soft one that you saw in class for GMMs. This hard assignment will make your code run faster and simplify your program.

In your E-step (Expectation), you calculate the *posterior* of your latent variables y_i given your observed variables $\mathbf{x}_i$ and current model parameters μ, σ:

$$\begin{aligned} y_i &= \arg\max_{k=1}^{K} p(y_i = k \mid \mathbf{x}_i; \mu^k, \sigma^k, \phi^k) \\ &= \arg\max_{k=1}^{K} \left(p(y_i = k; \phi^k) \prod_{j=1}^{M} p(x_{ij} \mid y_i = k; \mu_j^k, \sigma_j^k) \right) \end{aligned} \tag{9}$$

In your M-step (Maximization), you will update ϕ^k for each cluster k by simple frequency estimation:

$$\phi^k = \frac{|\{y_i : y_i = k\}|}{N} \tag{10}$$

and also μ_j^k and σ_j^k for each cluster k and component j:

$$\mu_j^k = \frac{1}{|\{y_i : y_i = k\}|} \sum_{x_{ij} : y_i = k} x_{ij} \tag{11}$$

$$\sigma_j^k = \frac{1}{|\{y_i : y_i = k\}| - 1} \sum_{x_{ij} : y_i = k} ||x_{ij} - \mu_j^k||_2^2 \tag{12}$$

Note that the formula for σ^k involves a denominator of $N^k - 1$ whereas the formula for μ^k divides by N^k, where N^k is the number of points that belong to cluster k. This is *the*

unbiased sample variance estimate. It turns out that if you divide by N^k you will underestimate the true variance of the cluster, which can be bad for unseen data. Showing this algebraically is left as an (optional) exercise for you, but intuitively it deals with the fact that before calculating the variance we chose the mean (μ^k), and we chose the mean that *minimizes* the variance of the observed data.

Additionally, this formula for σ^k does not work for clusters with size 0 or 1. Also for small clusters with very similar points, components of σ^k can underflow, so we will take an extra precaution any time we calculate σ^k. We will say that no σ^k_j can go below S_j and define σ^k_j for a cluster of size 0 or 1 to be S_j. S_j is a number calculated as 0.01 times *the unbiased sample variance* of the N training examples (already defined above):

$$S_j = 0.01 \; Var(\{x_{1j}, x_{2j}, \ldots, x_{Nj}\}) \tag{13}$$

1.5.3 Cluster Initialization

Again, everyone must use the same initialization method.

Begin by dividing your data into K folds. The ith example is in the kth fold as $k = i \;\%\; K$, where % is the mod operator. Make sure to use 0 based indexing for folds and examples. For example, if we have $K = 4$ folds, then the 200-th example is placed in the 0-th fold.

Initialize the mean μ^k and σ^k for the kth cluster based on the examples in the kth fold. This means that if you are using 0-indexing for cluster indexes, initialize the parameter μ^k to be the mean of the examples in the k-th fold, and σ^k to be the variance for the examples in the k-th fold (or if you are using 1-indexing, initialize the parameter μ^k to be the mean of the examples in the $(k-1)$-th fold). ϕ^k is easily computed based on the number of examples assigned to each cluster. You can then proceed with running the clustering algorithm by computing an E-step based on these initial parameters. Keep the data in the order it was loaded to avoid any inconsistencies in running the algorithm.

1.5.4 Numerical Accuracy

In Naive Bayes, and many other probabilistic models, you end up multiplying many small probabilities together. In theory this is not a problem, but in practice (i.e. using 64 bit floating point numbers), it can be. This is due to the fact that you can only represent a number so small using 64 bits at a given level of precision.[2] Because probabilities in our model can get *very* small, we need to use a small trick to ensure that it will not be rounded to 0 in 64 bit floating point number space. This trick is simply to take the log of our probabilities. Given that we are taking the log of all probabilities, you will have to adapt some formulas:

$$p(\mathbf{x}_i \mid y_i) = \prod_{j=1}^{M} p(x_{ij} \mid y_j)$$

[2]Floating point numbers in most major languages are, even though they represent a continuous (real) number, discrete. Almost all computer scientists feel that floating point numbers are evil, but most accept that they are a *necessary* evil. For details on how floats work, see the Wikipedia article on `doubles`.

becomes

$$\log(p(\mathbf{x}_i \mid y_i)) = \sum_{j=1}^{M} \log(p(x_{ij} \mid y_i)) \tag{14}$$

In general, we will not give you the log-probability form. You, as statistically-enlightened computer scientists, will take our notation and translate it into the appropriate code in log space.

1.5.5 How many clusters?

For a given run of your algorithm, you will be given the number of clusters, k. You will use `num-clusters` to indicate the value for k. The default should be 3.

You *must* define a new command line option for this parameter: `num-clusters`.

```
parser.add_argument("--num-clusters", type=int, help="The number of clusters in Naive Bayes clustering.",
                    default=3)
```

1.6 Number of Iterations

For both λ-Means and Naive Bayes Clustering, 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* define a new command line option for this parameter: `clustering-training-iterations`

```
parser.add_argument("--clustering-training-iterations", type=int,
                    help="The number of clustering iterations.", default=10)
```

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. In case of a tie, select the cluster with the lowest cluster index.

1.7.2 Empty Clusters

In λ-Means, it is possible (although unlikely) that a cluster can become empty during the course of 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. Similarly, set $\mu^k = \mathbf{0}$ if a cluster is empty in Naive Bayes Clustering.

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. 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 multi-class data, `speech.mc`.

1.8.1 Deliverables

You need to implement λ-Means and Naive Bayes Clustering. Your predictor will be selected by passing the string `lambda_means` or `nb_clustering` as the argument for the algorithm parameter.

1.8.2 How Your Code Will Be Called

You may use the following commands to test your algorithms.

```
python classify.py --mode train --algorithm lambda_means --model-file speech.lambda_means.model \
                   --data speech.train --cluster-lambda 0.0 --clustering-training-iterations 10
python classify.py --mode train --algorithm nb_clustering --model-file speech.nb_clustering.model \
                   --data speech.train --num-clusters 3 --clustering-training-iterations 10
```

To run the trained model on development data:

```
python classify.py --mode test --algorithm lambda_means --model-file speech.lambda_means.model \
                   --data speech.dev --predictions-file speech.dev.predictions
python classify.py --mode test --algorithm nb_clustering --model-file speech.nb_clustering.model \
                   --data speech.dev --predictions-file speech.dev.predictions
```

2 Analytical (40 points)

1. Semi-supervised EM algorithm (10 points) We consider a d-dimensional mixture model with the following probability density function

$$f(x; \theta_1, ..., \theta_k) = \sum_{i=1}^{K} \pi_i f_i(x; \theta_i),$$

where $f_i(x; \theta_i)$ is a probability density function. Suppose that we observe $n + m$ samples independently generated from the above mixture model, and m of the observed samples are labelled. Specifically, we have $x_1, ..., x_n \in \mathbb{R}^d$ and $x_{n+1}, ..., x_{n+m} \in \mathbb{R}^d$. Meanwhile, we know the labels corresponding to $x_{n+1}, ..., x_{n+m}$, i.e., $y_{n+1}, ..., y_{n+m} \in \{1, ..., K\}$. Design an EM algorithm to cluster the data.

[Hint: For $x_{n+1}, ..., x_{n+m}$, the corresponding labels are no longer missing values]

(a) Write the likelihood objective for this model.

(b) Write the update rules in each iteration.

2) Deep Neural Networks (15 points)

(a) Consider a 2-layer neural network, with M input nodes, Z nodes in the hidden layer and K nodes in the output layer. The network is fully connected, i.e. every node in the $n-1$th layer is connected to every node in the nth layer. However, for your application of interest, you suspect that only some of the nodes in the input are relevant. How would you modify the objective function to reflect this belief?

(b) Consider a N layer neural network. We could (a) train the entire network at once using back-propagation or (b) pre-train each layer individually, and then tune the final network with back-propagation. Will (a) and (b) converge to the same solution? Why would we favor strategy (a) vs. strategy (b)?

(c) Consider a $N \geq 2$ layer neural network with a single node in the output layer. We wish to train this network for binary classification. Rather than use a cross entropy objective, we want to take a max-margin approach and ensure a margin of $\gamma = 1$. Describe the structure of the last layer of the network, including the final activation function, and the training objective function that implements a max-margin neural network. What are the benefits of this network compared to one trained with cross entropy? Will a max-margin trained neural network learn the same decision boundary as an SVM?

3) Neural Networks (15 points) Suppose we have two inputs: x_1 and x_2. Both x_1 and x_2 are real numbers and their values are restricted such that $-1 \leq x_1 \leq 1$ and $-1 \leq x_2 \leq 1$. You may use activation functions of the form:

$$\theta_r(z) = \begin{cases} 1 & \text{if } z \leq r \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

where r determines the activation function. You may select a different r for each node in an artificial neural network. Create artificial neural networks for the following functions.

(a) Create a multi-layer network to recognize when $x_2 \geq \max(x_1, 1 - x_1)$.

(b) Create a multi-layer network to recognize when $|x_2| + |x_1| \leq 1$.

(c) Suppose we wanted to build a multi-layer network that approximates (with some error) the decision $x_1^2 + x_2^2 \leq 1$. Explain such an approximation.

For each network, describe the network structure, the value of each weight, and the activation function for each node. You may do this in words, or by drawing a picture. Please keep your answers clear and concise, i.e. we do not need a long explanation of how the network works; we only need the network definition.

3 What to Submit

In each assignment you will submit two things.

1. **Code:** Your code as a zip file named `code.zip`. **You must submit source code (.py files)**. We will run your code using the exact command lines described above, so make sure it works ahead of time. Remember to submit all of the source code, including what we have provided to you. We will include the libraries specific in `requirements.txt` but nothing else.

2. **Writeup:** Your writeup as a **PDF file** (compiled from latex) containing answers to the analytical questions asked in the assignment. Make sure to include your name in the writeup PDF and use the provided latex template for your answers.

Make sure you name each of the files exactly as specified (code.zip and writeup.pdf).

To submit your assignment, visit the "Homework" section of the website (http://www.cs475.org/.)

4 Questions?

Remember to submit questions about the assignment to the appropriate group on Piazza: https://piazza.com/class/it1vketjjo71l1. .