CMSC 726 : Machine Learning *First Last, UID, email*

# HW4

**The deadline for submission is 11:59:59PM Tuesday, November 9.**

## 0.1 Instructions

- Submit your assignment via the submit server, at `http://submit.cs.umd`. (Full instructions will be provided before the due date.)
- Please also **turn in a hardcopy** of your write-up. You may either turn it in on Tuesday during class, or slip it under Prof. Getoor's office door. The hardcopy should exactly match the electronic PDF submitted, and should be turned in within 12 hours of the electronic submission.
- Your electronic submission should be a single .ZIP (or .TAR.GZ) file, containing the PDF for your write-up and any code that you implemented. It should unzip to a folder named `hw4_<last_name>_<first_name>`.
- Your write-up should be named `hw4_<last_name>_<first_name.pdf>`.
- Do *not* include code in the write-up, unless asked to.
- Do *not* submit the .tex file.
- Do *not* submit any data files that we give to you.
- All MATLAB code should be documented with inline comments in the code. This includes things like a brief script/function description, input/output arguments and any implementation details that seem important to document. If you have any instructions on the high-level organization of your code, or how to run it, please create a README file.
- You may use LaTex or Word to create your write-up, but **you must submit in PDF format**.
- For your convenience, you can use the included .tex file. To do so,
  1. Edit the header information with your own personal information. Be sure to include your name and email address.
  2. Uncomment the `\begin{solution} ... \end{solution}` blocks and fill in your solution. (Note: you don't *need* to use the solution blocks if you find the formatting too restrictive. If you find that your solution spans multiple pages, you can break the solution block into multiple parts. Also, feel free to use `\begin{proof} ... \end{proof}` for proofs.)
- To add an image to this tex file, use the command `\includegraphics{filename}` (search the LaTeX documentation for additional arguments).
- There are two extra-credit questions. **These questions are optional.** Your grade will not be adversely affected if you do not do them.

# 1 Learning Theory

## 1.1 Conversion from Online Learning to PAC Learning

In class, we learned that, if a concept class is efficiently learnable in the online model, then it is also efficiently learnable in other models. In particular, we will prove that efficient online learnability implies efficient PAC learnability.

Suppose that we have a **conservative**[1] online learning algorithm $A$, with mistake bound $M$. To convert this into a PAC algorithm, for any $\epsilon, \delta \in [0, \frac{1}{2})$, we propose the following procedure:

1. Train $A$ on $m$ samples drawn from some distribution $\mathcal{D}$.
2. If at any point during training $A$ has correctly classified $k = \frac{1}{\epsilon} \ln\left(\frac{M+1}{\delta}\right)$ consecutive examples, stop training and output the current hypothesis $h$.

**Exercise 1a (15 points)** Prove that the sample complexity $m$ is bounded by

$$m \leq \frac{M+1}{\epsilon} \ln\left(\frac{M+1}{\delta}\right).$$

**Exercise 1b (15 points)** Prove that this is a valid PAC learning algorithm.

Note: you must show that, for the specified $k$ and any value of $\epsilon$, there exists some value of $\delta$ that satisfies the PAC requirements. (You could also prove that, for a given $k$ and $\delta$, there exists some value of $\epsilon$ that satisfies them, but you do not need to do so for this assignment.)

## 1.2 VC Dimension

**Definition 1.** *A* **triangular classifier** *is a triangle that partitions a 2D instance space into points inside the triangle and outside the triangle. The points inside can be designated as either positive or negative and the points outside are labeled as the opposite.*

**Exercise 1c (20 points)** Given Definition 1, what is the VC dimension of triangular classifiers? Give a formal VC dimension proof, proving both lower bound and upper bound. In proving lower bound shattering, you should enumerate as many label permutations as is necessary, without being redundant. (i.e. Generalize intelligently.)

[1]A conservative algorithm only updates the hypothesis upon making a mistake.

## 2 Clustering

In this section, you will implement two popular clustering algorithms: k-means and EM for a Gaussian mixture model. You will then test your implementations on the provided toy data.

Note: You are required to implement these algorithms yourself; not just call MATLAB's built-in functions. (Warning: you will receive zero credit if you do.) You are allowed to use built-in functions for auxiliary subroutines. For example, you may use a built-in function to compute the Gaussian PDF, but not the built-in function for k-means. If you are unsure whether a built-in function is acceptable, please email the TA.

### 2.1 Implementation

**Exercise 2a (15 points)** Implement k-means as a MATLAB function. Your function prototype should look like this:

```
[c, clust, obj] = k_means(X, c_init)
```

INPUTS:

- `X` : the $m \times n$ input matrix, where each row is a data point.
- `c_init` : a $k \times n$ matrix of *seeds* (initial values) for the $k$ centroids, where each row is a centroid vector. (Note: you can determine $k$ from the size of this argument.)

OUTPUTS:

- `c` : a $k \times n$ matrix of converged centroids, where each row is a centroid vector.
- `clust` : a length-$m$ vector, where each entry $i$ is equal to the cluster index of data point $i$.
- `obj` : a length-$T$ vector corresponding to the sum of squared distances to the cluster centroids at time $t = 1 \ldots T$.

Briefly discuss the convergence criteria you used. You may add an optional argument to your function to parameterize this.

**Exercise 2b (15 points)** Implement EM for a Gaussian mixture model as a MATLAB function, using log-likelihood as the objective function. Your function prototype should look something like this:

```
[mu, sig, clust, ll] = em_gmm(X, mu_init, sig_init)
```

INPUTS:

- `X` : the $m \times n$ input matrix, where each row is a data point.
- `mu_init` : a $k \times n$ matrix of *seeds* for the $k$ mean vectors, where each row is a mean vector.
- `sig_init` : a $kn \times n$ matrix of vertically-stacked *seeds* for the $k$ covariance matrices, where each $n \times n$ block is a covariance matrix.

OUTPUTS:

- `mu` : a $k \times n$ matrix of converged mean vectors, where each row is a mean vector.
- `sig` : a $kn \times n$ matrix of vertically-stacked converged covariance matrices, where each $n \times n$ block is a covariance matrix.
- `clust` : a $m \times k$ matrix of membership probabilities, where each entry $i, j$ is equal to the membership probability of data point $i$ to Gaussian $j$.
- `ll` : a length-$T$ vector corresponding to the log-likelihood at time $t = 1 \ldots T$.

Note: the covariances matrices may quickly become singular. To overcome this, you can add a small regularization term to the diagonal,

$$\Sigma \leftarrow \Sigma + \lambda I$$

where $\lambda$ is some small value that is just enough to "fix" the matrix.

Briefly discuss the convergence criteria and regularization method you used. You may add an optional argument to your function to parameterize this.

## 2.2 Evaluation

You will now test your clustering algorithms on the provided toy data. There are two files, `data/datasetA.txt` and `data/datasetB.txt`. In each case, the number of clusters should be $k = 3$. (You can verify this by plotting the data.)

**Exercise 2c (10 points)** For each data set, perform the following experiment:

1. Call your k-means function 10 times. For each run, use random initial centroid vectors within some "reasonable" range.
2. Select the "best" and "worst" runs of the algorithm. For each,
   (a) Document the converged centroid values and objective (the sum of squared distance to centroids).
   (b) Create a plot of the objective function and add it to your write-up.
   (c) Create (separate) plots of the initial centroids and the converged centroids as red X's, overlaid on top of the data set as green dots. Include these in your write-up.
3. Call your EM function 10 times. For each run, create initial Gaussians using a random mean vector within some "reasonable" range and the identity matrix for covariance.
4. Select the "best" and "worst" runs of the algorithm. For each,
   (a) Document the converged centroid, covariance values and log-likelihood.
   (b) Create a plot of the log-likelihood and include it in your write-up.
   (c) Use the provided MATLAB function `plotGaussians` to create (separate) plots of the initial Gaussians and the converged Gaussians, overlaid on top of the data set as green dots. Include these in your write-up.

All in all, this will generate 12 plots for k-means (2 data sets $\times$ best/worst runs $\times$ objective function, init and converged centroids) and 12 plots for EM (2 data sets $\times$ best/worst runs $\times$ log-likelihood, init and converged centroids). No need to display these in full scale. To save paper, please use the scale argument of `\includegraphics`.

**Exercise 2d (10 points)** Briefly discuss the results. For each data set, which algorithm performed best? Why do you think this is the case? What is the effect of initialization on convergence?

## 3 Extra Credit

**EC1** Random seeding is a simple but ineffective initialization technique. There are many known techniques that converge faster and closer to optimal results. Implement some of these techniques and discuss the results. Note: you may want to use a more challenging/interesting data set to compare performance. If you do, please include this with your submission.

**EC2**

**Definition 2.** *A concept class $\mathcal{C}$ over an instance space $X$ is said to be* **linearly ordered** *if:*

1. *$\mathcal{C}$ contains at least two concepts.*
2. *for any $c_1, c_2 \in \mathcal{C}$ either $c_1 \subseteq c_2$ or $c_2 \subseteq c_1$.*

Prove that the VC dimension of a linearly ordered concept class is equal to 1. Provide a formal VC dimension proof, proving both lower and upper bounds. (For this exercise, concept class is synonymous with hypothesis space.)