CS5350: Machine Learning *Due November 3, 2011*

HW3: Probabilistic Models, Feature Selection, Learning Theory, Clustering

1 Written Exercises

1. The *Poisson* distribution is a distribution over *positive count values*. It has the form $P(X \mid \lambda) = \frac{1}{e^\lambda}\frac{\lambda^X}{X!}$, where X is the count and λ is the (single) parameter of the Poisson. $X!$ denotes the factorial of X. Here $P(X \mid \lambda)$ is the likelihood (probability of data given the parameters). Now suppose we have a bunch of count data (for instance, the number of cars to pass an intersection on a given day, measured on N-many days) called $X_1, X_2, \ldots, X_N$. Compute the maximum likelihood estimate (MLE) for λ given this data. (Hint: just as we did in the class, write down the likelihood over the N observations, then take the log. Do some algebra to simplify and then take the derivative with respect to λ.) (*10 points*)

   First simplify with logs.

$$\begin{aligned}\prod_{X=1}^{N}(\frac{1}{e^\lambda}\frac{\lambda^X}{X!}) &= \sum_{X=1}^{N} log(\frac{1}{e^\lambda}\frac{\lambda^X}{X!}) \\ &= -n\lambda + (\sum_{X=1}^{N} X)\log\lambda - \sum_{X=1}^{N}\log(X!)\end{aligned}$$

   Then take the derivative.

$$\frac{d}{d\lambda}L(\lambda) = -n + (\sum_{x=1}^{N} X)\frac{1}{\lambda}$$

   This results in our MLE.

2. Most feature selection algorithms consider adding (in forward search) or removing (in backward search) *one feature at a time*. Even the methods that are based on using some sort of score for each feature consider only that feature while computing the score. For what type of features, this strategy (i.e., looking at features in *isolation*) might be suboptimal in hindsight. (*5 points*)

   One problem is that, when the features are all very similar, it can be difficult to know when to stop adding or subtracting features. In such cases, it is frequently true that there is no performance heuristic that provides the best point-of-cutoff in all situations, and further, it is frequently true that when you consider other factors in the situation, you actually can make such a decision reasonably effectively.

3. We have seen the concept of shattering for hypothesis classes of infinite size (e.g., set of linear classifiers in 2 or higher dimensions). Now consider the case when the hypothesis class $\mathcal{H}$ is of *finite size* (e.g., all decision trees of a fixed depth). Show that if a sample of size N is shattered by $\mathcal{H}$ (assume binary classification) then the size of this hypothesis class is *at least* 2^N. Also, using this result, give an upper bound on the VC-dimension of a finite hypothesis class of size $|\mathcal{H}|$. (*10 points*)

If the sample size is N, and we are using binary classification, then combinatorially there are at least 2^N decision trees for the sample. We don't have a proof that this is a tight bound, so we can say that $|\mathcal{H}| \leq 2^N$. Solving for N, we get $|\mathcal{H}| = \log_2 N$ for the upper bound on the VC-dimension.

4. Assume we have a dataset in two dimensions with five data points at: {(1, 0), (-1, 0), (0, 1), (3, 0), (3, 1)}. Run two iterations of k-means by hand with cluster centers initialized at (-1, 0) and (3, 1) (so $K = 2$). For each iteration, write down which data points get assigned to which cluster center and also compute the cluster centers. Has the algorithm converged after two iterations? (*5 points*)

After our first iteration, $(1, 0)$, $(-1, 0)$, and $(0, 1)$ will be clustered around $(-1, 0)$ (let's call this cluster 1) while $(3, 0)$ and $(3, 1)$ will be clustered around $(3, 1)$ (let's call this cluster 2). At this point, the new centroid for cluster 1 will be $(0, \frac{1}{3})$, while the centroid of cluster 2 will be $(3, 1)$. For the second iteration, these clusters do not change at all. Thus, this cluster does converge.
For references, the distances I computed are as follows. For iteration 1:

	dist to (-1,0)	dist to (3,1)
(0,1)	2	$\sqrt{5}$
(-1,0)	0	$\sqrt{17}$
(0,1)	$\sqrt{2}$	3
(3,0)	4	1
(3,1)	17	0

For iteration 2:

	dist to $(0, \frac{1}{3})$	dist to $(3, \frac{1}{2})$
(0,1)	$\frac{\sqrt{10}}{3}$	$\frac{\sqrt{17}}{2}$
(-1,0)	$\frac{\sqrt{10}}{3}$	$\frac{\sqrt{65}}{2}$
(0,1)	$\frac{2}{3}$	$\frac{\sqrt{37}}{2}$
(3,0)	$\frac{\sqrt{82}}{3}$	$\frac{1}{2}$
(3,1)	$\frac{\sqrt{85}}{3}$	$\frac{1}{2}$

5. What advantages does the Gaussian Mixture Model (GMM) have over the k-means clustering. Note that in GMM, each mixture component is a Gaussian distribution $\mathcal{N}or(\mu_k, \Sigma_k)$ where μ_k and Σ_k are the mean and covariance of this Gaussian. Which special case of the GMM corresponds to k-means? (*10 points*)

If you think of k-means as a restricted GMM (as the question implies you should), then the main advantages sprout from the fact that it's not tied to Euclidean distance as a measure of similarity. Most noticeably, it is possible to model clusters using multiple membership. The presence of Σ_k and μ_k also provide a number of uses: one is that k-means can be viewed as a GMM with a built-in variance, which generally restricts its uses (*e.g.*, it is worse for clusters of very different sizes, it is more sensitive to outliers, it is worse for clusters of different densities, etc.).
To answer the second part of the question, the specific case of GMM that k-means corresponds to is the one where the covariance is restricted to the same as the Euclidean case (which is one over the diagonals, I think).

6. Give an intuitive explanation (100 words or less) of the *general* EM algorithm (i.e., what are the quantities being estimated in each step and *how* are they estimated?). Give 2 examples (not including GMM) of where the EM algorithm can be used. (*10 points*)

7. (*6350 only*) The k-means algorithm makes hard assignments of points to clusters, unlike the Gaussian mixture model which makes soft assignments (so each point has a nonzero probability of being assigned

to each cluster, with these probabilities summing to one). Show that you can accomplish something similar by slightly modifying the k-means algorithm. In particular, describe how the 2 steps of k-means (cluster assignments and cluster center computations) would need to be changed in this variant of the k-means algorithm. (Hint: The hard assignments of the original k-means are a result of assigning each point *fully* to the cluster center it is the *closest* to; in the soft assignments you would want to give a part of each point to each of the K clusters, based on how close it is to a particular cluster center). (*10 points*)

2 Programming Exercises

1. In this part, we will implement a feature selection method based on the Pearson correlation coefficient (PCC). The *estimate* of $\mathcal{R}(d)$ is given by

$$R(d) = \frac{\sum_{n=1}^{N}(X_{nd} - \bar{X}_d)(Y_n - \bar{Y})}{\sqrt{\sum_{n=1}^{N}(X_{nd} - \bar{X}_d)^2 \sum_{n=1}^{N}(Y_n - \bar{Y})^2}}$$

   X is the $N \times D$ data matrix where each row is an example and each column is a feature. X_d above denotes the d-th column of X, consisting of the d-th feature of all the N examples. Y is an $N \times 1$ vector of labels.

   In the above equation, the bar notation denotes an average over the index n (so $\bar{X}_d$ is basically the average of X_d).Using the *absolute* value of the PCC estimate ($|R(d)|$), for the provided training data (train.mat), rank the features in the order of decreasing relevance. Now run a KNN classifier on the provided dataset (train.mat and test.mat) using the m best features by varying m from 1 to 15. Plot the percentage accuracy on test data as a function of m. The skeleton code is in `feature_select.m` and the `run.m` with test it (but you will have to plot the results yourself :).

   Also answer why would you want to use the absolute PCC scores ($|R(d)|$) rather than the raw PCC scores ($R(d)$)? (*15 points*)

2. In this task, we implement K-means clustering. A shell is in `kmeans.m`. There are essentially two things you need to implement. First, is the actual K-means algorithm. Second is the initialization based on the "furthest point" heuristic. To remind you, the furthest point heuristic starts by choosing one of the data points as the first cluster center, then chooses the second cluster center which is furthest from the first center, then chooses the third cluster center which is furthest from both the first and the second centers (equivalently, from their mean), and so on.

   You should follow the comments in `kmeans.m` for hints on how to do the implementation.

   There is another script, `test_kmeans.m` for (not surprisingly!) testing your kmeans function. This function reads in some simple two dimensional data and then tries to cluster it. The figures it produces are:

   - A plot of the data, unclustered
   - The data clustered with K = 2 and random initialization
   - The data clustered with K = 3 and random initialization (this is run 16 times... you should see a small amount of variability in the outputs). The plots also include scores.
   - The data clustered with K = 3 and "furthest point" initialization (this is run 16 times... you should see a small amount of variability in the outputs). The plots also include scores.
   - A plot of K = {2; 3; 4; 5; 6; 8; 10; 15; 20} versus score.
   - The data clustered for K = {2; 3; 4; 5; 6} with together with scores.

   To verify that things seem to be going okay, aside from just checking to see if your clusters look reasonable in the plots, the scores that I get in Figure 6 are: 120, 58, 47, 38, 32. There will be a small amount of variation due to the random initialization, but they should be reasonably close. (*35 points*)