ISTA 421/521 - Final Take-home Assignment

Due: Monday, December 15, 5pm

24 pts total undergraduate / 30 pts total graduate

Emanuel Carlos de Alcantara Valente

Undergraduate

Instructions

You must work on the problems **INDEPENDENTLY**, not in groups.

The work on each problem must be your own.

Include in your final submission the pdf of written answers along with separate files for any python scripts that you write in support of answering the questions (although no scripts are needed for this assignment; if you do write scripts, clearly note in your pdf written answers which script filenames were used). You are required to create either a .zip or tarball (.tar.gz / .tgz) archive of all of the files for your submission and submit the archive to the D2L dropbox by the date/time deadline above.

NOTE: Problem 4 is **required for graduate students only**; undergraduates may complete them for extra credit equal to the point value.

(FCMA refers to the course text: Rogers and Girolami (2012), *A First Course in Machine Learning.*)

1. [4 points] Adapted from **Exercise 5.4** of FCMA p.204:

   Compute the maximum likelihood estimates of q_{mc} for class c of a Bayesian classifier with multinomial class-conditionals and a set of N_c, M-dimensional objects belonging to class c: $\mathbf{x}_1, ..., \mathbf{x}_{N_c}$.

   Solution.

   We will compute the maximum likelihood by taking the logarithmic function of the multinomial class-conditional, take the derivative and setting to zero, since maximizing $log(P)$ is equivalent to maximizing $L(P)$.

$$L = P(x_n|q) = \left(\frac{s_n!}{\prod_{m=1}^{M} x_{nm}!}\right) \prod_{m=1}^{M} q_m^{x_{nm}}$$

$$log(P(x_n|q)) = log(s_n!) - log(\prod_{m=1}^{M} x_{nm})! + log(\prod_{m=1}^{M} q_m^{x_{nm}}) =$$

$$= log(s_n!) - log(\prod_{m=1}^{M} x_{nm})! + \sum_{m=1}^{M} x_{nm} log(q_m)$$

$$\frac{\partial log(\mathbf{L}(q, \mathbf{x}))}{\partial q} = \frac{\partial \sum_{m=1}^{M} x_{nm} log(q_m)}{\partial q}$$

   Using lagrange multiplier to solve this derivate (maximaxing problem), we will have:

   The constraint is that:

   $\sum_i^M q_m = 1$

   The lagrange is then:

   $g(\lambda, q) = \sum_{m=1}^{M} x_{nm} log(q_m) - \lambda(\sum_{m=1} q_m - 1)$

   The another constraint is:

   $\frac{\partial g(\lambda,q)}{\partial q_m} = \frac{\sum_{n=1}^{N_c} x_{nm}}{q_m} - \lambda$

   Setting this derivative to zero and combining with the constraint $\sum_i^M q_m = 1$

   We will finally have:

$$q_{cm} = \frac{\sum_{n=1}^{N_c} x_{nm}}{\sum_{m'=1}^{M} \sum_{n=1}^{N_c} x_{nm'}}$$

2. [4 points] Adapted from **Exercise 5.5** of FCMA p.204:

   For a Bayesian classifier with multinomial class-conditionals with M-dimensional parameters $\mathbf{q}_c$, compute the posterior Dirichlet for class c when the prior over $\mathbf{q}_c$ is a Dirichlet with constant parameter α and the observations belonging to class c are the N_c observations $\mathbf{x}_1, ..., \mathbf{x}_{N_c}$.

   Solution. We will calculate the posterior by multiplying the likelihood (multinomial class-conditionals) and the given Dirichlet prior.

$$p(q_c|x_n) \propto p(x_n|r)p(q_c)$$

$$p(q_c|x_n) \propto \left(\frac{s_n!}{\prod_{m=1}^{M} x_{nm}!}\right) \prod_{m=1}^{M} q_{cm}^{x_{nm}} \frac{\Gamma(\sum_{m=1}^{M} \alpha_m)}{\prod_{m=1}^{M} \Gamma(\alpha_m)} \prod_{m=1}^{M} q_{cm}^{\alpha_m - 1}$$

But:

$$s_n! = (\sum_{m=1}^{M} x_{nm})! = \sum_{m=1}^{M} x_{nm} \Gamma(\sum_{m=1}^{M} x_{nm})$$

and

$$\prod_{m=1}^{M} x_{nm}! = \prod_{m=1}^{M} x_{nm} \Gamma(\prod_{m=1}^{M} x_{nm}) = \prod_{m=1}^{M} x_{nm} \Gamma(x_{nm})$$

Substituting these in our posterior:

$$p(q_c|x_n) \propto = \frac{\sum_{m=1}^{M} x_{nm} \Gamma(\sum_{m=1}^{M} x_{nm}) \Gamma(\sum_{m=1}^{M} \alpha_m)}{\prod_{m=1}^{M} x_{nm} \Gamma(x_{nm}) \Gamma(\alpha_m)} \prod_{m=1}^{M} q_{cm}^{x_{nm} + \alpha_m - 1}$$

As α_m is a constant, our final posterior will be:

$$p(q_c|x_n) \propto = \frac{\sum_{m=1}^{M} x_{nm} \Gamma(\sum_{m=1}^{M} x_{nm}) \Gamma(\alpha M)}{\prod_{m=1}^{M} x_{nm} \Gamma(x_{nm}) \Gamma(\alpha)} \prod_{m=1}^{M} q_{cm}^{x_{nm} + \alpha - 1}$$

3. [4 points] Adapted from **Exercise 6.1** of FCMA p.234:

   Derive the EM update for the variance of the dth dimension and the kth component, σ_{kd}^2, when the cluster components have a diagonal Gaussian Likelihood:

$$p(\mathbf{x}_n | z_{nk} = 1, \mu_{k1}, ..., \mu_{KD}, \sigma_{k1}^2, ..., \sigma_{kD}^2) = \prod_{d=1}^{D} \mathcal{N}(\mu_{kd}, \sigma_{kd}^2)$$

   Solution. <Solution goes here>

4. [6 points; **Required only for Graduates**] Adapted from **Exercise 6.6** of FCMA p.235:

   Derive an EM algorithm for fitting a mixture of Poisson distributions. Assume you observe N integer counts, $x_1, ..., x_N$. The likelihood is:

$$p(\mathbf{X}|\boldsymbol{\Delta}) = \prod_{n=1}^{N} \sum_{k=1}^{K} \pi_k \frac{\lambda_k^{x_n} \exp\{-\lambda_k\}}{x_n!}$$

   Solution. <Solution goes here>

5. [2 points]

   For a support vector machine, if we remove one of the support vectors from the training set, does the size of the maximum margin decrease, stay the same, or increase for that dataset? Why? Also justify your answer by providing a simple, hand-designed dataset (no more than 2-D) in which you identify the support vectors, draw the location of the maximum margin hyperplane, remove one of the support vectors, and draw the location of the resulting maximum margin hyperplane. You do not have to run any code – this can be done completely by hand and drawn schematically.

Solution.

In general, if we remove one of the support vectors from the training set, the size of the maximum margin increases or keeps the same. It happens because if we "remove" one support vector we are "removing" or "keeping" constraints to maximize the margin. In the example above, the maximum margin increased.

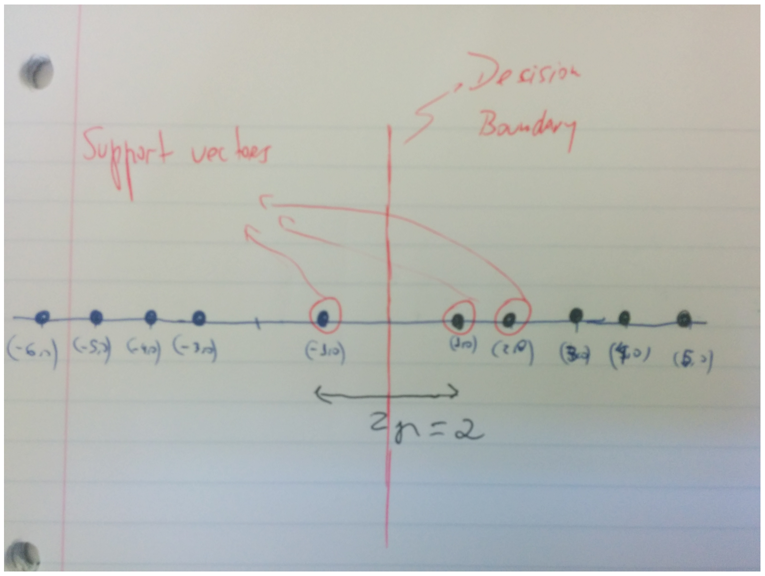

Figure 1: Before removing one support vector (point (1,0)) $2\gamma = 2$

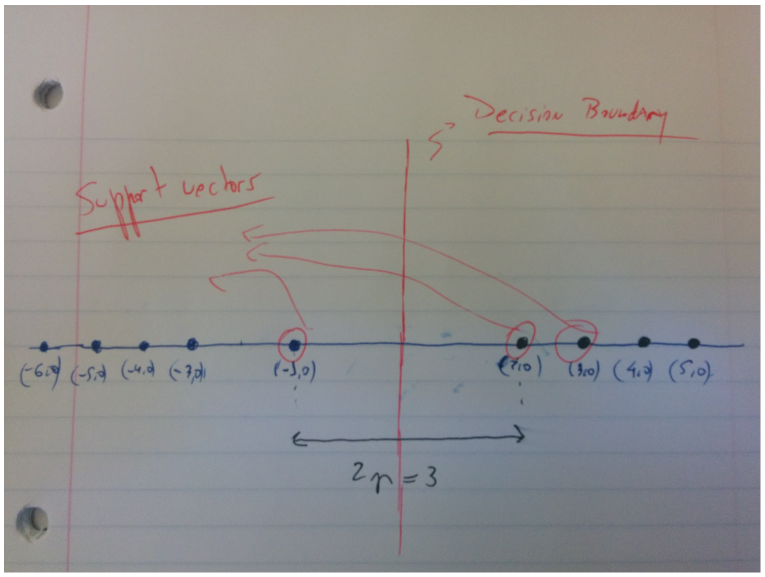

Figure 2: After removing one support vector (point (1,0)) $2\gamma = 3$

6. [3 points]

   Consider the 2-bit XOR problem for which the entire instance space is as follows: In each row, x_1

t	x_1	x_2
-1	-1	-1
$+1$	-1	1
$+1$	1	-1
-1	1	1

   and x_2 are the coordinates and t is the class for the point. These instances are not linearly separable, but they are separable with a polynomial kernel. Recall that the polynomial kernel is of the form $\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \top \mathbf{x}_j + c)^d$ where c and d are integers. Select (by hand!) values for c and d that yield a space in which the instances above are linearly separable. Write down the mapping Φ to which this kernel corresponds, write down $\Phi(x)$ for each instance above, and write down the parameters of a hyperplane in the expanded space that perfectly classifies the instances (there are a range of possible hyperplanes, just pick one set of hyperplane parameters that satisfies separating the points – you are not maximizing the margin here, just coming up with one possible separating hyperplane). Again, this can be done without writing code or deriving an analytic solution!

   Solution. <Solution goes here>

7. [2 points]

   Why does the kernel trick allow us to solve SVMs with high dimensional feature spaces without significantly increasing the running time?

   Solution.

   Because the terms representing the data only appear within inner (i.e., dot) products, like $x_n^T x_m$, $x_n^T x_{new}$. Therefore, we never see the data on its own. Moreover, these terms can be represented compactly by kernels.

8. [5 points] In this course we introduced the Metropolis-Hastings algorithm. Provide the following: (a) describe the problem it is designed to solve, including how it solves this problem by avoiding a potentially intractable problem; (b) describe the basic procedure; (c) describe the role of the proposal distribution.