CPSC 540 Assignment 3 (due February 23)

Duality and Densities

1. You can work in groups on 1-3 people on the assignments.
2. Place your names and student numbers on the first page.
3. Submit all answers as a single PDF file using the handin program (details on Piazza).
4. The PDF should answer the questions *in order*, and include relevant code in the appropriate place.
5. You will lose marks if answers are unclear or if the TA can't easily find the location of the answers.
6. Corrections/clarifications after the first version of this document is posted will be marked in red.
7. Acknowledge sources of help from outside of class/textbook (classmates, online material, papers, etc.).
8. All numbered questions are equally weighted.

Unnoffiial Course Evaluation

To help improve the course as we go along, or to suggest of how things could be done differently, please fill out the survey here:
`https://survey.ubc.ca/s/cs540-eval`

1 Fenchel Duality, Dual Coordinate Ascent

Recall that the Fenchel dual for the primal problem

$$P(w) = f(Xw) + g(w),$$

is the dual problem

$$D(z) = -f^*(-z) - g^*(X^Tz),$$

or if we re-parameterize in terms of $-z$:

$$D(z) = -f^*(z) - g^*(-X^Tz), \tag{1}$$

Re-formulating a problem in terms of the Fenchel dual can have numerous advantages: (i) if the primal is strongly-convex then the dual is guaranteed to be smooth (even if the primal is non-smooth), (ii) if g is an L2-regularizer then the dual allows us to use the kernel trick and in some cases is *sparse* in terms of the dual variables, and (iii) if f is a sum then the dual problem typically has a form that makes coordinate optimization efficient. In this question, you'll first get practice deriving Fenchel duals. Then you'll implement a dual coordinate ascent algorithm, and finally consider a kernelized version of the method.

1.1 Deriving Dual Problems

Convex conjugates are discussed in Section 3.3 of Boyd and Vandenberghe (`http://stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf`). Read this, then derive the Fenchel dual for the following problems:

1. $P(w) = \frac{1}{2}\|Xw - y\|^2 + \lambda\|w\|_1$ (L1-regularized least squares)
2. $P(w) = \frac{1}{2}\|Xw - y\|_1 + \frac{\lambda}{2}\|w\|^2$ (robust regression with L2-regularization)
3. $P(w) = \sum_{i=1}^N \log(1 + \exp(-y^i w^T x^i)) + \frac{\lambda}{2}\|w\|^2$ (regularized maximum entropy)

Hint: The structure of the primal and dual problem make this question much easier than taking a generic Lagrangian dual. A generic strategy to compute Fenchel duals is as follows:

- Determine X, f, and g to put the problem into the primal format.
- Determine the form of f^* and g^* (note that A here is not relevant).
- Evaluate f^* at $-z$ and g^* at $X^T z$ to get the final form.

Hints: For a differentiable f, you can often solve for the value achieving the sup inside of $f^*(z)$ by taking the gradient of $(x^T z - f(x))$ and setting it to zero (keeping in mind whether there are values of z where the sup might be infinity). Example 3.26 in the book gives the convex conjugate in the case where f is a norm. Section 3.3.2 of the book shows how the convex conjugate changes if you scale a function and/or compose a function with an affine transformation. For parts 1 and 2, the X in the primal will just be data matrix X. But for part 3, it will be easier if you define X as a matrix with row i is given by $y^i x^i$. For part 3 you'll want to use $f(x) = \sum_{i=1}^n \log(1 + \exp(z_i))$, which is a *separable* function (meaning that you can optimize each z_i independently).

1.2 Dual Coordinate Ascent

The dual of the SVM problem,

$$P(x) = \sum_{i=1}^{N} \max\{0, 1 - y^i w^T x^i\} + \frac{\lambda}{2}\|w\|^2,$$

is

$$D(z) = e^T z - \frac{1}{2\lambda} z^T \text{Diag}(y) X X^T \text{Diag}(y) z, \quad \text{s.t. } 0 \le z_i \le 1, \forall_i.$$

where e is a vector of ones, Diag(y) is diagonal matrix with the y^i values along the diagonal, and we have $w^* = \frac{1}{\lambda} X^T \text{Diag}(y) z^*$. Starting from *example_dual.m*, implement a dual coordinate optimization strategy to optimize the SVM objective. Hand in your code, report the optimal value of the primal and dual objectives with $\lambda = 1$.

Hint: the objective function is a quadratic and the constraints are just lower and upper bounds. This lets you derive the optimal update for one variable with the other held fixed: solve for the value of z_i with a partial derivative of zero, and if this violates the constraints then the solution must be either $z_i = 0$ or $z_i = 1$ (depending on which one is lower).

1.3 Kernel Trick

Removed.

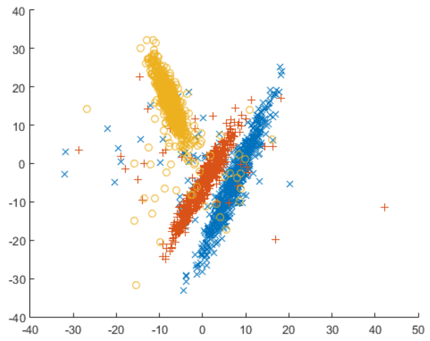
2 Density Estimation

Density estimation is an unsupervised task that is one of the fundamental problems in machine learning. In the case of continuous data, the simplest density estimation methods model the data as a particular parametric distribution (Gaussian, Laplace, ...). These simple models are easy to fit and are also the building blocks for many more-advanced methods.

In this question, we first look at deriving the MLE for generative classifiers that assume a Gaussian distribution of the features x given the class label y. You will then implement this method and explore a robust variation on it using the multivariate T distribution (where numerical optimization is needed to estimate the parameters).

2.1 Generative Classifiers with Gaussian Assumption

Consider the 3-class classification dataset in this image:



In this dataset, we have 2 features and each colour represents one of the classes. Note that the classes are highly-structured: the colours each roughly follow a Gausian distribution plus some noisy samples.

Since we have an idea of what the features look like for each class, we might consider classifying inputs x using a *generative classifier*. In particular, we are going to use Bayes rule to write

$$p(y = c|x, \Theta) = \frac{p(x|y = c, \Theta) \cdot p(y = c|\Theta)}{p(x|\Theta)},$$

where Θ represents the parameters of our model. To classify a new example $\hat{x}$, generative classifiers would use

$$\hat{y} = \underset{y \in \{1,2,\ldots,k\}}{\operatorname{argmax}} \; p(\hat{x}|y = c, \Theta)p(y = c|\Theta),$$

where in our case the total number of classes k is 3 (The denominator $p(\hat{x}|\Theta)$ is irrelevant to the classification since it is the same for all y.) Modeling $p(y = c|\Theta)$ is eays: we can just use a k-state categorical distribution,

$$p(y = c|\Theta) = \theta_c,$$

where θ_c is a single parameter for class c. The maximum likelihood estimate of θ_c is given by n_c/n, the number of times we have $y^i = c$ (which we've called n_c) divided by the total number of data points n.

Modeling $p(x|y = c, \Theta)$ is the hard part: we need to know the *probability of seeing the feature vector x given that we are in class c*. This corresponds to solve a density estimation problem for each of the k possible classes. To make the density estimation problem tractable, we'll assume that the distribution of x given that $y = c$ is given by a $\mathcal{N}(\mu_c, \Sigma_c)$ distribution for a class-specific μ_c and Σ_c,

$$p(x|y = c, \Theta) = \frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma_c|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x - \mu_c)^T\Sigma_c^{-1}(x - \mu_c)\right).$$

Since we are distinguishing between the probability under k different Gaussians to make our classification, this is called *Gaussian discriminant analysis* (GDA). In the special case where we have a constant $\Sigma_c = \Sigma$ across all classes it is known as *linear discriminant analysis* (LDA) since it leads to a linear classifier between any two classes (while the region of space assigned to each class forms a convex polyhedron as in k-means clustering). Another common restriction on the Σ_c is that they are a diagonal matrices, since this only requires $O(d)$ parameters instead of $O(d^2)$ (corresponding to assuming that the features are independent univariate Gaussians given the class label). Given a dataset $\mathcal{D} = \{(x^i, y^i)\}_{i=1}^n$, where $x^i \in \mathbb{R}^d$ and $y^i \in \{1, \dots, k\}$, the maximum likelihood estimate (MLE) for the μ_c and Σ_c in the GDA model is the solution to

$$\underset{\mu_1,\mu_2,\dots,\mu_k,\Sigma_1,\Sigma_2,\dots,\Sigma_k}{\operatorname{argmax}} \prod_{i=1}^n p(x^i|y^i, \mu_{y^i}, \Sigma_{y^i}).$$

This means that the negative log-likelihood will be equal to

$$\begin{aligned}-\log p(X|y, \Theta) &= -\sum_{i=1}^n \log p(x^i|y^i|\mu_{y^i}, \Sigma_{y^i}) \\ &= \sum_{i=1}^n \frac{1}{2}(x^i - \mu_{y^i})^T\Sigma_{y^i}^{-1}(x^i - \mu_{y^i}) + \frac{1}{2}\sum_{i=1}^n \log|\Sigma_{y^i}| + \text{const.}\end{aligned}$$

Derive the maximum likelihood likelihood estimate (MLE) for the GDA model under the following restrictions:

1. We have *common diagonal* covariance matrices, $\Sigma_c = D$ (d parameters).
2. We have *individual scaled-identity* covariance matrices, $\Sigma_c = \sigma_c^2 I$ (k parameters).

In each case you should assume that you have class-specific mean vectors μ_c, but note that the formula for μ_c will be the same in all cases.

It's painful to derive these from scratch, but you should be able to see a pattern that would allow other common restrictions. Without deriving the result from scratch (hopefully), give the MLE for the following two scenarios:

3. We have *individual full* covariance matrices Σ_c ($O(kd^2)$ parameters).
4. We have *individual diagonal* covariance matrices, $\Sigma_c = D_c$ (kd parameters).
5. We have *common full* covariance matrices, $\Sigma_c = \Sigma$ ($O(d^2)$ parameters).

Hints: you will be able to substantially simplify the notation if you use the notation $\sum_{i \in y_c}$ to mean the sum over all values i where $y^i = c$. Similarly, you can use n_c to denote the number of cases where $y_i = c$, so that we have $\sum_{i \in y_c} 1 = n_c$. Note that the determinant of a diagonal matrix is the product of the diagonal entries, and the inverse of a diagonal matrix is a diagonal matrix with the reciprocals of the original matrix along the diagonal. For part three you can use the result from class regarding the MLE of a general multivariate Gaussian.

2.2 Implementation of a Gaussian Generative Classifier

When you run *example_generative* it loads a variant of the dataset in the figure that has 12 features and 10 classes. This data has been split up into a training and test set, and the code fits a k-nearest neighbour classifier to the training set then reports the accuracy on the test data ($\sim 36\%$). The k-nearest neighbour model does poorly here since it doesn't take into account the Gaussian-like structure in feature space for each class label.

Write a function *generativeGaussian* that fits a GDA model to this dataset (using individual full covariance matrices). Hand in the function and report the test set accuracy.

2.3 Implementation of a T-distribution Generative Classifier

In this question we would like to replace the Gaussian distribution of the previous problem with the more robust multivariate-t distribution so that it isn't influenced as much by the noisy data. Unlike the previous case, we don't have a closed-form solution for the parameters. However, if you run *example_tdist* it generates random noisy data and fits a multivariate-t model (you will need to add the 'minFunc' directory to the Matlab path for the demo to work).

By using the *multivariateT* model, write a new function *generativeStudent* that implements a generative model that is based on the multivariate-t distribution instead of the Gaussian distribution. Report the test accuracy accuracy with this model.

3 Expectation Maximization

The Expectation maximization (EM) algorithm is one the most common way to fit models with missing data. It is used for things as diverse as filling in missing values, fitting mixture models, and semi-supervised learning. In this question you will get practice deriving EM algorithms as well as implementing one.

3.1 Semi-Supervised Gaussian Discriminant Analysis

Consider fitting a GDA model where some of the y^i values are missing at random. In particular, let's assume we have n labeled examples (x^i, y^i) and then another another t unlabeled examples (x^i). This is a special case of *semi-supervised learning*, and fitting generative models with EM is one of the oldest semi-supervised learning techqniues. When the classes exhibit clear structure in the feature space, it can be very effective even if the number of labeled examples is very small.

Derive the EM update for fitting the parameters of a GDA model (with individual full covariance matrices) in the semi-superivsed setting where we have n labeled examples and t unlabeled examples.

Hint: most of the work has been done for you in the EM notes on the course webpage. You can use the result (**) and the update of θ_c from those notes, but you will need to work out the update of the parameters of the Gaussian distribution $p(x^i|y^i, \Theta)$

3.2 Implementation

If you run the demo *example_SSL*, it will load a variant of the dataset from the previous question, but where the number of labeled examples is small and a large number of unlabeled examples are available. Because the number of labeled examples it quite small, the performance is worse than in Question 2. Write a function *generativeGaussianSSL* that fits the generative Gaussian model of the previous question using EM to incorporate the unlabeled data. Hand in the function and report the test error when training on the full dataset.

Hint: although EM often leads to simple updates, implementing them correctly can often be a pain. One way to help debug your code is to compute the observed-data log-likelihood after every iteration. If this number goes down, then you know your implementation has a problem. You can also test your updates of sets of variables in this way too. For example, if you hold the μ_c and Σ_c fixed and only update the θ_c, then the log-liklihood should not go down. In this way, you can test each of combinationso of updates on their to make sure they are correct.

3.3 Poisson Mixture Model

Consider a density estimation with examples $x^i \in \{\mathbb{Z}_{\geq 0}\}^d$ represent *counts* of d variables. In this setting, a natural way to model an individual variable x^i_j would be with a Poisson distribution,

$$p(x^i_j = \alpha | \lambda_j) = \frac{\lambda_j^\alpha e^{-\lambda_j}}{\alpha!}.$$

However, if we assume this structure for each variable then the variables would be independent. One way to model dependent count variables would be with a mixture of independent Poisson distributions,

$$p(x^i|\Theta) = \sum_{c=1}^{k} \theta_c \prod_{j=1}^{d} p(x^i_j|\lambda_{jc}),$$

where Θ contains all the θ_c and λ_{jc} values. Derive the EM update for this.

Hint: most of the work has been done for you in the EM notes on the course webpage.

4 Robust PCA

Robust PCA is a variant of PCA that allows outliers in the data matrix X. One of the applications of this model that people are currently exploring is background subtraction: the problem of separating out the background from the foreground in video sequences. In this model, we treat the background as a low-rank matrix A (as in standard PCA) and treat the foreground E is a sparse matrix. The standard formulation of robust PCA is the constrained optimization

$$\operatorname*{argmin}_{A,E} ||A||_* + \lambda ||E||_1 \text{ subject to } A + E = D$$

Here, $||A||_*$ is the nuclear norm (sum of the singular values of A). Minimizing the nuclear norm of A encourages it to be low rank. In contrast, $||E||_1$ is the (entry-wise) L1 norm of E and this encourages it to be sparse. We can approximate the above problem by unconstrained optimization by imposing a penalty if the constraint is violated. If we let μ be the parameter of the penalty term then the optimization becomes

$$\operatorname*{argmin}_{A,E} ||A||_* + \lambda ||E||_1 + \frac{1}{2\mu}||D - A - E||_2^2, \tag{2}$$

where $\|\cdot\|$ is the entrywise L2-norm of the matrix (also known as the Frobenius norm). If we want to solve the original problem, we can gradually decrease μ to increase the cost of violating the constraint, and the limit as μ goes to zero is the original problem. This problem is a sum of two non-smooth convex terms and a smooth convex term, but the non-smooth terms are *block separable* ($\|A\|_*$ only affects A and $\|E\|_1$ only affects E) and admit efficient proximal operators (provided that the matrices are not too large). This allows us to apply *block coordinate optimization.*

4.1 Block-Coordinate Update

Derive the optimal update of E in and the optimal update of A in (2). In other words, find a formula for the optimal value of E if we fix the value of A, and vice versa.

Hint: the proximal-operator for a nuclear norm scaled by some constant γ,

$$\operatorname*{argmin}_Y \frac{1}{2}\|Y - X\|^2 + \gamma\|Y\|_*,$$

is given by applying the soft-threshold operator to the singular values of X with a threshold of γ.

4.2 Background Subtraction

Download and extract *HighwayI.zip*, which contains images of successive frames of a highway video (and 'GT_HIghwayI.png' is the image for the background). Our aim is to eliminate the foreground from all frames in the video. Run *parse_input* to form a matrix D (stored in D.mat), where each columns represents one frame (the groundtruth image is stored in GT.mat). Complete and submit the code in *proxGradRPCA* to implement the block update steps you found above (note that the code already decrease μ as it goes). Report the PSNR achieved with $\lambda = 2^{-5}$ and $\lambda = 2^{-8}$. You can check your output by running *parse_output* with the estimated A as input (see *example_RPCA*). This script displays the resulting successive foreground subtracted frames found by your code (with $\lambda = 2^{-5}$ you can see cars as part of A but with $\lambda = 2^{-8}$ it should mostly remove the cars in the images).

Hint: to make the code run faster, you will want to call *svd* with the 'econ' option.