Intro to Machine Learning (CS771A, Autumn 2018)
Homework 3
Due Date: Oct 31, 2018 (11:59pm)

Instructions:

- Only electronic submissions will be accepted. Your main PDF writeup must be typeset in LaTeX (please also refer to the "Additional Instructions" below).
- Your submission will have two parts: The main PDF writeup (to be submitted via Gradescope `https://www.gradescope.com/`) and the code for the programming part (to be submitted via this Dropbox link: `https://tinyurl.com/cs771-a18-hw3-a`). Both parts must be submitted by the deadline. We will be accepting late submissions upto 72 hours after the deadline (with every 24 hours delay incurring a 10% late penalty). We won't be able to accept submissions after that.
- We have created your Gradescope account (you should have received the notification). Please use your IITK CC ID (not any other email ID) to login. Use the "Forgot Password" option to set your password.

Additional Instructions

- We have provided a LaTeX template file `hw3sol.tex` to help typeset your PDF writeup. There is also a style file `ml.sty` that contain shortcuts to many of the useful LaTeX commends for doing things such as boldfaced/calligraphic fonts for letters, various mathematical/greek symbols, etc., and others. Use of these shortcuts is recommended (but not necessary).
- Your answer to every question should begin on a new page. The provided template is designed to do this automatically. However, if it fails to do so, use the `\clearpage` option in LaTeX before starting the answer to a new question, to *enforce* this.
- While submitting your assignment on the Gradescope website, you will have to specify on which page(s) is question 1 answered, on which page(s) is question 2 answered etc. To do this properly, first ensure that the answer to each question starts on a different page.
- Be careful to flush all your floats (figures, tables) corresponding to question n before starting the answer to question $n+1$ otherwise, while grading, we might miss your important parts of your answers.
- Your solutions must appear in proper order in the PDF file i.e. solution to question n must be complete in the PDF file (including all plots, tables, proofs etc) before you present a solution to question $n+1$.
- For the programming part, all the code and README should be zipped together and submitted as a single file named `yourrollnumber.zip`. Please DO NOT submit the data provided.

Problem 1 (15 marks)

(Learning SVM via Co-ordinate Ascent) Consider the soft-margin linear SVM problem

$$\arg\max_{0\leq\boldsymbol{\alpha}\leq C} f(\boldsymbol{\alpha})$$

where $f(\boldsymbol{\alpha}) = \boldsymbol{\alpha}^\top \mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^\top \mathbf{G}\boldsymbol{\alpha}$, $\mathbf{G}$ is an $N \times N$ matrix such that $G_{nm} = y_n y_m \boldsymbol{x}_n^\top \boldsymbol{x}_m$ and $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_N]$ are the Lagrange multipliers. Given the optimal $\boldsymbol{\alpha}$, the SVM weight vector is $\boldsymbol{w} = \sum_{n=1}^{N} \alpha_n y_n \boldsymbol{x}_n$

Your goal is to derive a **co-ordinate ascent** procedure for the vector $\boldsymbol{\alpha}$, such that each iteration updates a uniformly randomly chosen entry α_n of the vector $\boldsymbol{\alpha}$. However, instead of updating $\boldsymbol{\alpha}$ via standard co-ordinate descent as $\alpha_n = \alpha_n + \eta g_n$ where g_n denotes the n-th entry of the gradient vector $\nabla_{\boldsymbol{\alpha}} f(\boldsymbol{\alpha})$, we will update it as $\alpha_n = \alpha_n + \delta_*$ where $\delta_* = \arg\max_\delta f(\boldsymbol{\alpha} + \delta \mathbf{e}_n)$ and $\mathbf{e}_n$ denotes a vector of all zeros except a 1 at entry n.

Essentially, this will give the new α_n that guarantees the maximum increase in f, with all other α_n's fixed at their current value. Derive the expression for δ_* and give a sketch of the overall co-ordinate ascent algorithm.

Note that your expression for δ_* should be such that the constraint $0 \leq \alpha_n \leq C$ is maintained.

PS: I know I had said that, in this homework, I will give a programming task to implement SVM. :-) Well, even though I am not asking you to implement it, by solving the problem above, you would have pretty much everything you need to implement one of the state-of-the-art algorithms for linear SVM.

Problem 2 (5 marks)

(Within and Across) Suppose we wish to cluster some data by learning a function f such that $f_n = f(\boldsymbol{x}_n)$ is the cluster assignment for point $\boldsymbol{x}_n$. Show that finding f by minimizing $\mathcal{L}_W$, which is defined as the sum of squared distances between all pairs of points that are *within* the same cluster, i.e.,

$$\arg\min_f \mathcal{L}_W = \arg\min_f \sum_{n,m} \mathbb{I}[f_n = f_m] ||\boldsymbol{x}_n - \boldsymbol{x}_m||^2$$

implicitly *also* maximizes the sum of squared distances between all pairs of points that are in *different* clusters. (Note: It is not for credit but you can also show that the above is equivalent to the K-means objective!)

Problem 3 (15 marks)

(Estimating a Gaussian when Data is Missing) Suppose we have collected N observations $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ using a sensor. Let us assume each $\boldsymbol{x} \in \mathbb{R}^D$ as generated from a Gaussian distribution $\mathcal{N}(\mu, \Sigma)$. We would like to estimate the mean and covariance of this Gaussian. However, suppose the sensor was faulty and each $\boldsymbol{x}_n$ could only have part it as observed (think of a blacked out image). Denote $\boldsymbol{x}_n = [\boldsymbol{x}_n^{obs}, \boldsymbol{x}_n^{miss}]$ where $\boldsymbol{x}_n^{obs}$ and $\boldsymbol{x}_n^{miss}$ denote the observed and missing parts, respectively, of $\boldsymbol{x}_n$. We only get to see $\boldsymbol{x}_n^{obs}$. Note that different observations could have different parts as missing (e.g., different images may have different sets of pixels as missing), so the indices of the observed/missing entries of the vector $\boldsymbol{x}_n$ may be different for different n.

Your goal is to develop an EM algorithm that gives maximum likelihood estimates of μ and Σ given this partially observed data. In particular, in the EM setting, you will treat each $\boldsymbol{x}_n^{miss}$ as a latent variable and estimate its conditional distribution $p(\boldsymbol{x}_n^{miss} | \boldsymbol{x}_n^{obs}, \mu, \Sigma)$, given the current estimates μ and Σ of the parameters. In the M step, you will re-estimate μ and Σ, and will alternate between E and M steps until convergence.

Clearly write down the following: (1) The expression for $p(\boldsymbol{x}_n^{miss} | \boldsymbol{x}_n^{obs}, \mu, \Sigma)$; (2) The expected CLL for this model; (3) The update equations for μ and Σ.

Also clearly write down all the steps of the EM algorithm in this case, with appropriate equations.

Note/Hint: For this problem, you may find it useful to use the result that if $\boldsymbol{x} = [\boldsymbol{x}_a, \boldsymbol{x}_b]$ is Gaussian then $p(\boldsymbol{x}_a | \boldsymbol{x}_b)$ is also Gaussian (you may refer to Section 4.3.1 of MLAPP for the result).

Problem 4 (10 marks)

Some features are labeled and some are unlabeled that's why it is sem supervised learning.

(Semi-supervised Classification) Consider learning a generative classification model for K-class classification with Gaussian class-conditionals $\mathcal{N}(\boldsymbol{x}|\mu_k, \Sigma_k)$, $k = 1, \ldots, K$ with class marginals $p(y = k) = \pi_k$. However, unlike traditional generative classification, in this setting we are given N labeled examples $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$ and an additional M *unlabeled* examples $\{\boldsymbol{x}_{N+1}, \ldots, \boldsymbol{x}_{N+M}\}$. Design an EM algorithm to estimate all the unknowns of this model and clearly write down the expressions required in each step of the EM algorithm.

Note: You need not re-do the derivations we have done in the class or other homeworks/practice sets; feel free to re-use those without re-deriving from scratch.

Problem 5 (25 marks)

(Latent Variable Models for Supervised Learning) Consider learning a regression model given training data $\{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$, with $\boldsymbol{x}_n \in \mathbb{R}^D$ and $y_n \in \mathbb{R}$. Let us give a small twist to the standard probabilistic linear model for regression that we have seen in the class. In particular, we will be introducing a latent variable $\boldsymbol{z}_n$ with each training example $(\boldsymbol{x}_n, y_n)$. The generative story would now be as follows

$$\begin{aligned} z_n &\sim \text{multinoulli}(\pi_1, \ldots, \pi_K) \\ y_n &\sim \mathcal{N}(\boldsymbol{w}_{z_n}^\top \boldsymbol{x}_n, \beta^{-1}) \end{aligned}$$

Note that the model for the responses y_n is still discriminative, since the inputs are not being modeled.

The latent variables are $\mathbf{Z} = (z_1, \ldots, z_N)$ and the global parameters are $\Theta = \{(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_K), (\pi_1, \ldots, \pi_K)\}$.

(1) Give a brief explanation (max. 5 sentences) of what the above model is doing and why you might want to use it instead of the standard probabilistic linear model which models each response as $y_n \sim \mathcal{N}(\boldsymbol{w}^\top \boldsymbol{x}_n, \beta^{-1})$.

(2) Derive an ALT-OPT algorithm to estimate $\mathbf{Z}$ and (MLE of) Θ, and clearly write down each step's update equations. For $\mathbf{Z}$, you must give the update equation for each individual latent variable $z_n, n = 1, \ldots, N$. Likewise, for Θ, you must give the update equation for each $\boldsymbol{w}_k, k = 1, \ldots, K$, and $\pi_k, k = 1, \ldots, K$. Also, what will be the update of each z_n if $\pi_k = 1/K, \forall k$. Give a brief intuitive explanation (max 1-2 sentences) as to what this update does.

(3) Derive an expectation-maximization (EM) algorithm to estimate $\mathbf{Z}$ and (MLE of) Θ, and clearly write down each step's update equations. Also show that, as $\beta \to \infty$, the EM algorithm reduces to ALT-OPT.

Note: It is okay to skip some of the standard/obvious steps from your derivations and write down the final expressions directly if the derivations are similar to what we have done in the class or previous homeworks/practice sets, but your final expressions must be clearly and unambiguously written.

Problem 6 (30 marks)

(Programming Problem 1) For this problem, your task will be to implement kernel ridge regression and ridge regression with landmark based features (let's call the latter simply "landmark-ridge"), and train and test these models on the provided dataset (already split into training and test). For both models, you have to use the RBF kernel $k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \exp(-\gamma||\boldsymbol{x}_n - \boldsymbol{x}_m||^2)$ with bandwidth parameter $\gamma = 0.1$. The training and test datasets are given in files `ridgetrain.txt` and `ridgetest.txt`. In each file, each line is an example, with first number being the input (a single feature) and the second number being the output.

You need to do the following

1. For kernel ridge regression, train the model with the regularization hyperparameter $\lambda = 0.1$ and use the learned model to predict the outputs for the test data. Compare the model's predictions with the true test

outputs (given to you) by plotting on a graph. In particular, plot all the test inputs and the corresponding true outputs (x, y) on a 2D plot in blue color, and likewise all the test inputs and corresponding predicted outputs in red color. Repeat this exercise for $\lambda = 1, 10, 100$. What do you observe from the plots? For each case, also report the root-mean-squared-error (RMSE) on the test data, which is defined as square root of the mean of squared differences of true outputs and the predicted outputs.

2. For landmark-ridge, you first need to extract landmark based features using the RBF kernel and you will then use data with these features to train a linear ridge regression model. Again, keep the regularization hyperparameter fixed at $\lambda = 0.1$. Try $L = 2, 5, 20, 50, 100$ uniformly randomly chosen landmark points from the training set, train the model for each case and compute the predictions on the test data. Plot the results for each case just like you did in part 1 (but only for $\lambda = 0.1$). What do you observe from the plots? What's the RMSE in each case? What value of L seems good enough?

(Programming Problem 2) For this problem, your task will be to implement the K-means clustering algorithm and try it on a provided toy (but interesting) dataset (`kmeans_data.txt`) consisting of points in two dimensions. The provided dataset also has 2 clusters (so you would use $K = 2$). However, the data is such that the standard K-means will NOT work well since the clusters are not spherical and not separable linearly (you can check this by plotting the data in 2D using a scatter plot). You will consider two ways to handle this issue.

1. **Using Hand-crafted Features:** Propose a feature transformation to the original data that will transform it such that K-means will be able to learn the correct clusters! Before proposing the transformation, plot the original data to see what transformation(s) might probably work. Apply your K-means implementation on this transformed version of the data to verify if your transformation works. Plot your obtained clustering results by showing points (in the original 2D space) in cluster 1 in red and points in cluster 2 in green.

2. **Using Kernels:** Although you can kernelize K-mean via the kernel K-means algorithm, you will try something else. You will use the landmark based approach to extract good features and your implementation of standard K-means on these features. The kernel that you will use for the landmark based approach is the RBF kernel $k(\boldsymbol{x}_n, \boldsymbol{x}_m) = \exp(-\gamma||\boldsymbol{x}_n - \boldsymbol{x}_m||^2)$ with bandwidth parameter $\gamma = 0.1$. We are going to try something (that may seem) crazy: We will pick $L = 1$ (yes, just ONE) landmark point randomly from the dataset and see whether $L = 1$ (which basically means just a single landmark based feature) is good enough to learn a correct clustering (at least for this data). It turns out that some landmark choices will be work and some won't work. Try 10 runs of the algorithm, each time with a different randomly chosen (single) landmark and check the obtained clustering. For each run, produce a plot as you did in part 1 and, on the same plot, also show the chosen landmark point in blue color. Justify why you get a correct clustering in some cases and a not-so-correct looking clustering in other cases.

Important Note: To avoid randomness in cluster mean initialization, we will choose the first two points in the dataset (the first two lines in the provided dataset) as the initial cluster centers.

Deliverables: Create two folders, one for each of the programming problems. The folder for each problem should contain the code for both of its parts (in separate files) and the respective plots. Both folders should be archived in a single zip file (follow the naming convention as in previous homeworks). In the main writeup, write about your observations for each of the problems.