Intro to Machine Learning (CS771A, Autumn 2020)
Homework 2
Due Date: November 27, 2020 (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).
- The PDF writeup containing your solution has 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-a20-hw2`)
- 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 `hw2sol.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 (20 marks)

(Second-Order Optimization for Logistic Regression) Show that, for the logistic regression model (assuming each label $y_n \in \{0, 1\}$, and no regularization) with loss function $\mathcal{L}(\boldsymbol{w}) = -\sum_{n=1}^{N}(y_n \boldsymbol{w}^\top \boldsymbol{x}_n - \log(1 + \exp(\boldsymbol{w}^\top \boldsymbol{x}_n)))$, iteration t of a *second-order* optimization based update $\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \mathbf{H}^{(t)^{-1}}\boldsymbol{g}^{(t)}$, where $\mathbf{H}$ denotes the Hessian and $\boldsymbol{g}$ denotes the gradient, reduces to solving an *importance-weighted* regression problem of the form $\boldsymbol{w}^{(t+1)} = \arg\min_{\boldsymbol{w}} \sum_{n=1}^{N} \gamma_n^{(t)}(\hat{y}_n^{(t)} - \boldsymbol{w}^\top \boldsymbol{x}_n)^2$, where γ_n denotes the importance of the n^{th} training example and $\hat{y}_n$ denotes a modified real-valued label. Also, clearly write down the expression for both, and provide a brief justification as to why the expression of γ_n makes intuitive sense here.

Problem 2 (20 marks)

(Perceptron with Kernels) We have seen that, due to the form of Perceptron updates $\boldsymbol{w} = \boldsymbol{w} + y_n \boldsymbol{x}_n$ (ignore the bias b), the weight vector learned by Perceptron can be written as $\boldsymbol{w} = \sum_{n=1}^{N} \alpha_n y_n \boldsymbol{x}_n$, where α_n is the number of times Perceptron makes a mistake on example n. Suppose our goal is to make Perceptron learn nonlinear boundaries, using a kernel k with feature map ϕ. Modify the standard Perceptron algorithm to do this. In particular, for this kernelized variant of the Perceptron algorithm (1) Give the initialization, (2) Give the mistake condition, and (3) Give the update equation.

Problem 3 (20 marks)

(SVM with Unequal Class Importance) Sometimes it costs us a lot more to classify negative points as positive than positive points as negative. (for instance, if we are predicting if someone has cancer then we would rather err on the side of caution (predicting "yes" when the answer is "no") than vice versa). One way of expressing this in the support vector machine model is to assign different costs to the two kinds of mis-classification. The primal formulation of this is:

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad \frac{||\boldsymbol{w}||^2}{2} + \sum_{n=1}^{N} C_{y_n} \xi_n$$

subject to $y_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) \geq 1 - \xi_n$ and $\xi_n \geq 0, \forall n$.

The only difference is that instead of one cost parameter C, there are two, C_{+1} and C_{-1}, representing the costs of misclassifying positive examples and misclassifying negative examples, respectively.

Write down the Lagrangian problem of this modified SVM. Take derivatives w.r.t. the primal variables and construct the dual, namely, the maximization problem that depends only on the dual variables $\boldsymbol{\alpha}$, rather than the primal variables. In your final PDF write-up, you need not give each of every step in these derivations (e.g., standard steps of substituting and eliminating some variables) but do write down the key steps. Explain (intuitively) how this differs from the standard SVM dual problem; in particular, how the C variables differ between the two duals.

Problem 4 (20 marks)

(SGD for K-means Objective) Recall the K-means objective function: $\mathcal{L} = \sum_{n=1}^{N}\sum_{k=1}^{K} z_{nk}||x_n - \mu_k||^2$. As we have seen, the K- means algorithm minimizes this objective by taking a greedy iterative approach of assigning each point to its closest center (finding the z_{nk}'s) and updating the cluster means $\{\mu_k\}_{k=1}^{K}$. The standard K-means algorithm is a batch algorithm and uses all the data in every iteration. It however can be made online by taking a random example x_n at a time, and then (1) assigning x_n "greedily" to the "best" cluster, and (2) updating the cluster means using SGD on the objective $\mathcal{L}$. Assuming you have initialized $\{\mu_k\}_{k=1}^{K}$ randomly and are reading one data point x_n at a time,

- How would you solve step 1?
- What will be the SGD-based cluster mean update equations for step 2? Intuitively, why does the update equation make sense?

- Note that the SGD update requires a step size. For your derived SGD update, suggest a good choice of the step size (and mention why you think it is a good choice).

Problem 5 (20 marks)

(Kernel K-means) Assuming a kernel k with an infinite dimensional feature map ϕ (e.g., an RBF kernel), we can neither store the kernel-induced feature map representation of the data points nor can store the cluster means in the kernel-induced feature space. How can we still implement the kernel K-means algorithm in practice? Justify your answer by sketching the algorithm, showing all the steps (initialization, cluster assignment, mean computation), clearly giving the mathematical operations in each. In particular, what is the difference between how the clusters means would need to be stored in kernel K-means versus how they are stored in standard K-means? Finally, assuming each input to be D-dimensional in the original feature space, and N to be the number of inputs, how does kernel K-means compare with standard K-means in terms of the cost of input to cluster mean distance calculation (please answer this using the big $\mathcal{O}$ notation)?

Problem 6 (Programming Problem, 50 marks)

Part 1: You are provided a dataset in the file `binclass.txt`. In this file, the first two numbers on each line denote the two features of the input $\boldsymbol{x}_n$, and the third number is the binary label $y_n \in \{-1, +1\}$.

Implement a generative classification model for this data assuming Gaussian class-conditional distributions of the positive and negative class examples to be $\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_+, \sigma_+^2\mathbf{I}_2)$ and $\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_-, \sigma_-^2\mathbf{I}_2)$, respectively. Note that here $\mathbf{I}_2$ denotes a 2×2 identity matrix. Assume the class-marginal to be $p(y_n = 1) = 0.5$, and use MLE estimates for the unknown parameters. Your implementation need not be specific to two-dimensional inputs and it should be almost equally easy to implement it such that it works for any number of features (but it is okay if your implementation is specific to two-dimensional inputs only).

On a two-dimensional plane, plot the examples from both the classes (use red color for positives and blue color for negatives) and the **learned decision boundary** for this model. Note that we are not providing any separate test data. Your task is only to learn the decision boundary using the provided training data and visualize it.

Next, repeat the same exercise but assuming the Gaussian class-conditional distributions of the positive and negative class examples to be $\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_+, \sigma^2\mathbf{I}_2)$ and $\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_-, \sigma^2\mathbf{I}_2)$, respectively.

Finally, try out an SVM classifier (with **linear kernel**) on this data (we've also provided the data in the format libSVM requires) and show the learn decision boundary. For this part, you do not need to implement SVM. There are many nice implementations of SVM available, such as the one in scikit-learn and the very popular libSVM toolkit. Assume the "C" (or λ) hyperparameter of SVM in these implementations to be 1.

Part 2: Repeat the same experiments as you did for part 1 but now using a different dataset `binclassv2.txt`. Looking at the results of both the parts, which of the two models (generative classification with Gaussian class-conditional and SVM) do you think seems to work better for each of these datasets, and in general?

Deliverables: Include your plots (use a separate, appropriately labeled plot, for each case) and experimental findings in the main writeup PDF. Submit your codes in a separate zip file on the provided Dropbox link. Please comment the code so that it is easy to read and also provide a README that briefly explains how to run the code. For the SVM part, you do not have to submit any code but do include the plots in the PDF (and mention the software used - scikit-learn or libSVM).