Intro to Machine Learning (CS771A, Autumn 2018)
Homework 1
Due Date: August 31, 2018 (11:59pm)

Instructions:

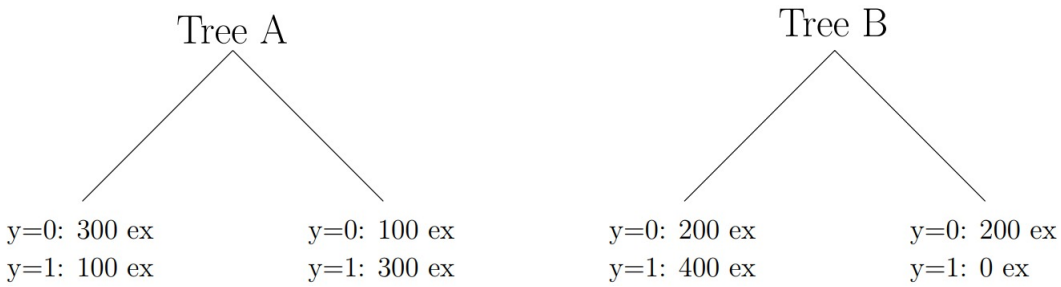
- Only electronic submissions will be accepted. Your main PDF writeup must be typeset in LaTeX (please also refer to the "Additional Instructions" below).
- Your submission will have two parts: The main PDF writeup (to be submitted via Gradescope `https://www.gradescope.com/`) and the code for the programming part (to be submitted via this Dropbox link: `https://tinyurl.com/cs771-a18-hw1-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 `hw1sol.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 (10 marks)

(Misclassification Rate vs Information Gain) Consider a binary classification data set consisting of 400 data points from class 0 and 400 data points from class 1. Suppose that a decision tree model $\mathcal{A}$ splits these into (300, 100) at the first leaf node and (100, 300) at the second leaf node. (Here, (n, m) denotes that n points are assigned to class 0 and m points are assigned to class 1.) Similarly, suppose that a second decision tree model $\mathcal{B}$ splits them into (200, 400) and (200, 0). (See the figure below.)



(1) Compute the training data **misclassification rate** (i.e., what fraction of training examples will be misclassified) for the two trees: are they equal or not? (2) Evaluate the **information gain** for the two trees and use these to compare the trees. (3) Do you get different answers for (1) and (2)? Does this make sense?

Problem 2 (10 marks)

(Consistent or Not?) An important notion for a classifier is that of *consistency*. A classification algorithm is said to be *consistent* if, whenever it has access to **infinite** amounts of training data, its error rate approaches the optimal error rate (a.k.a. *Bayes optimal*). Consider the noise-free setting (i.e., every training input is labeled correctly). Here, the Bayes optimal error rate is zero. Is the one-nearest-neighbor algorithm consistent in this setting? Briefly justify your answer in 100 words or less. no, consider a case of 1d data points, alternating classes

Problem 3 (10 marks)

(Linear Regression meets Nearest Neighbors) Show that, for the unregularized linear regression model, where the solution $\hat{\boldsymbol{w}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \boldsymbol{y}$, the prediction at a test input $\boldsymbol{x}_*$ can be written as a weighted sum of all the training responses, i.e.,

$$f(\boldsymbol{x}_*) = \sum_{n=1}^{N} w_n y_n$$

write (XtT)-1X as F, then w = F1y1 + .. FnYn

Give the expression for the weights w_n's in this case and briefly discuss (<50 words) in what way these weights are different from the weights in a *weighted version* of K nearest neighbors where each w_n typically is the inverse distance of $\boldsymbol{x}_*$ from the training input $\boldsymbol{x}_n$. **Note:** You do not need to give a very detailed expression for w_n (if it makes algebra messy) but you must give a precise meaning as to what w_n depends on and how it is different from the weights in the weighted K nearest neighbors.

Problem 4 (10 marks)

(Feature-Specific ℓ_2 Regularization) The usual ℓ_2 regularized least squares regression objective can be written as $\mathcal{L}(\boldsymbol{w}) = \sum_{n=1}^{N} (y_n - \boldsymbol{w}^\top \boldsymbol{x})^2 + \frac{\lambda}{2} \boldsymbol{w}^\top \boldsymbol{w}$. In this case, the extent of regularization is the same on all the features (and is controlled by λ). Propose an alternative that still uses ℓ_2 regularization on $\boldsymbol{w}$ but the extent of regularization is different for each entry w_d. Write down the objective function and derive the closed form expression for the weight vector $\boldsymbol{w}$.

closed form - instead of Id, diagonal entries will be lambda1, lambda2 and so on.
expression : l1^2w1 + l2*w2 + ... + ln^2wn

Problem 5 (20 marks)

(Multi-output Regression with Reduced Number of Parameters) Consider the multi-output regression in which each output $\boldsymbol{y}_n \in \mathbb{R}^M$ in a real-valued vector, rather than a scalar. Assuming a linear model, we can model the outputs as $\mathbf{Y} = \mathbf{XW}$, where $\mathbf{X}$ is the $N \times D$ feature matrix and $\mathbf{Y}$ is $N \times M$ response *matrix* with row n being $\boldsymbol{y}_n^\top$ (note that each column of $\mathbf{Y}$ denotes one of the M responses), and $\mathbf{W}$ is the $D \times M$ **weight matrix**, with its M columns containing the M weight vectors $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_M$. Let's define a squared error loss function $\sum_{n=1}^{N}\sum_{m=1}^{M}(y_{nm} - \boldsymbol{w}_m^\top \boldsymbol{x}_n)^2$, which is just the usual squared error but summed over all the M outputs. Firstly, verify that this can also be written in a more compact notation as $\text{TRACE}[(\mathbf{Y} - \mathbf{XW})^\top(\mathbf{Y} - \mathbf{XW})]$.

Note that this is the same set-up as one of the problems from Practice Set 1 (if you had a chance to look at it :-)). However, here we will assume that the weight matrix $\mathbf{W}$ can be written as a product of two matrices, i.e., $\mathbf{W} = \mathbf{BS}$ where $\mathbf{B}$ is $D \times K$ and $\mathbf{S}$ is $K \times M$ (assume $K < \min\{D, M\}$). Note that there is a benefit of modeling $\mathbf{W}$ this way, since now we need to learn only $K \times (D + M)$ parameters as opposed to $D \times M$ parameters and, if K is small, this can significantly reduce the number of parameters (in fact, reducing the *effective* number of parameters to be learned is another way of regularizing a machine learning model). Note (you can verify) that in this formulation, each $\boldsymbol{w}_m$ can be written as a linear combination of K columns of $\mathbf{B}$.

With the proposed representation of $\mathbf{W}$, the new objective will be $\text{TRACE}[(\mathbf{Y} - \mathbf{XBS})^\top(\mathbf{Y} - \mathbf{XBS})]$ and you need to learn both $\mathbf{B}$ and $\mathbf{S}$. While it is certainly possible to learn both (one way to do it is using a procedure called "alternating optimization" that we will look at later), for now, let's keep it simple and assume that the matrix $\mathbf{B}$ is known and only $\mathbf{S}$ needs to be estimated. So your problem reduces to

$$\hat{\mathbf{S}} = \arg\min_{\mathbf{S}} \text{TRACE}[(\mathbf{Y} - \mathbf{XBS})^\top(\mathbf{Y} - \mathbf{XBS})]$$

While it is possible (and straighforward) to also include the Frobenious forms of $\mathbf{S}$ in the above objective to regularize $\mathbf{S}$, we will ignore that for brevity/simplicity.

Derive the expression for $\hat{\mathbf{S}}$. Feel free to use results for matrix derivatives (results you will need can be found in Sec. 2.5 of the Matrix Cookbook). Briefly explain how the form of the solution is identifical to the solution of standard multi-output regression (Practice Set 1 problem), but uses a *transformed* version of the inputs $\mathbf{X}$.

Bonus (not for credit, just for practice :-)) Assume $\mathbf{B}$ is also unknown and find its estimate as well. I already mentioned that one way to do so is by using an "alternating" procedure that opimizes the above objective w.r.t. $\mathbf{S}$ with $\mathbf{B}$ fixed, and then w.r.t. $\mathbf{B}$ with $\mathbf{S}$ fixed (and repeating until convergence). Does the optimal $\mathbf{B}$ has closed form solution, when $\mathbf{S}$ is fixed?

Problem 6 (40 marks)

(Programming Problem) Your task is to implement and test prototype based classification using the provided dataset ("Animals with Attributes" version 1 - AwA_v1). You may download the dataset (given in MATLAB and Python format) using this link: `https://tinyurl.com/cs771-a18-hw1data` (caution: the data is in a couple of hundred MBs; there is also a README file that contains the description of the data). In this dataset, each input represents an image of an animal and output is the class (what this animal is). The dataset has a total of 50 classes and the number of features for each input is 4096 (note: these features were extracted by a deep learning model and you don't need to perform any other preprocessing of features for this problem).

However, we are going to give a small twist to the basic prototype based classification problem. The training set provided to you has only examples from 40 of the classes. We will refer to these 40 classes as "seen classes" (have training data for these classes) and the remaining 10 classes as "unseen classes" (do not have training data for these classes). The test inputs will be only from these 10 unseen clases.

Recall that prototype based classification requires computing the mean of each class. While computing the means of the 40 seen classes is easy (since we have training data from these classes), what we actually need is the mean of the remaining 10 classes (since these are our test classes). How do we get these means?

Well, we clearly need some *additional information* about the classes in order to solve this problem (without that there is no hope of solving this problem). To this end, you are provided an 85-dimensional **class attribute vector** $\boldsymbol{a}_c \in \mathbb{R}^{85}$, for each class c (both seen as well as unseen classes). Each class attribute vector contains information about that class and consists of 85 binary-valued attributes representing the class (e.g., whether this animal has stripes). You may think of each $\boldsymbol{a}_c$ as a *class feature* vector which tells us what the class looks like.

Now consider two ways how these class attribute vectors can be used to obtain the means of unseen classes:

- **Method 1:** Model the mean $\mu_c \in \mathbb{R}^{4096}$ of each unseen class c (where $c = 41, \ldots, 40$) as a *convex* combination of the means $\mu_1, \ldots, \mu_{40}$ of the 40 seen classes (again note that $\mu_1, \ldots, \mu_{40}$ can be computed easily since we do have training data from these 40 classes)

$$\boldsymbol{\mu}_c = \sum_{k=1}^{40} s_{c,k} \boldsymbol{\mu}_k, \quad c = 41, \ldots, 50$$

  where $\boldsymbol{s}_c = [s_{c,1}, s_{c,2}, \ldots, s_{c,40}]$ is a vector of similarities of the unseen class c with each of the 40 seen classes. Here each similarity is defined as the inner product of the class attribute vectors of two classes, e.g., $s_{c,k} = \boldsymbol{a}_c^\top \boldsymbol{a}_k$ is the similarity between two clases c and k. **Note:** We will also *normalize* the vector $\boldsymbol{s}_c$ as $s_{c,k} = s_{c,k} / \sum_{\ell=1}^{40} s_{c,\ell}$ so that it sums to 1 (and thus can be used as weights in the convex combination defined above). This procedure will give us the means of 10 unseen classes and then you can apply the prototype based classifer to predict the labels of each test input.

- **Method 2:** Train a linear model that can predict the mean $\mu_c \in \mathbb{R}^{4096}$ of any class using its class attribute vector $\boldsymbol{a}_c \in \mathbb{R}^{85}$. We can train this linear model using $\{(\boldsymbol{a}_c, \mu_c)\}_{c=1}^{40}$ as our training data and then apply it to predict μ_c for each unseen class using its class attribute vector $\boldsymbol{a}_c$. Note that this can be posed as a multi-output regression problem $\mu_c = \mathbf{W}\boldsymbol{a}_c$ where $\boldsymbol{a}_c \in \mathbb{R}^{85}$ is the input, $\mu_c \in \mathbb{R}^{4096}$ is the vector-valued regression output, and $\mathbf{W}$ is 85×4096 matrix of weights that need to be learned. The solution to this multi-output regression problem is $\mathbf{W} = (\mathbf{A}_s^\top \mathbf{A}_s + \lambda \mathbf{I})^{-1} \mathbf{A}_s^\top \mathbf{M}_s$ where $\mathbf{A}_s$ is the 40×85 matrix containing the class attribute vectors of 40 seen classes, and $\mathbf{M}_s$ is the 40×4096 matrix containing the means of the 40 seen classes. Once you have $\mathbf{W}$, you can compute the mean μ_c of any unseen class as $\mathbf{W}\boldsymbol{a}_c$ where $\boldsymbol{a}_c$ is its class attribute vector. This procedure will give us the means of 10 unseen classes and then you can apply the prototype based classifer to predict the labels of each test input.

You task is to implement both methods. In your main write-up, report the test-set classification accuracy for each by comparing the respective model's predictions with the provided ground truth labels (accuracy is the percentage of test inputs with correctly predicted classes). For method 2, try $\lambda = 0.01, 0.1, 1, 10, 20, 50, 100$ and report the test accuracy for each value of λ. Which value of λ gives the best test set accuracy?

Note: In the dataset folder, I've also provided a brief pseudo-code (or rather a summary) of the procedure to follow for the implementation (since this is the very first assignment, I felt it would be helpful for you all).

Important: To predict the class of a test input, you should only compute its distances from the means of 10 unseen classes, not from the means of the seen classes (since the test data only consists of inputs from the unseen classes). The other setting where the test data can consist of both seen and unseen class inputs is also possible but we are not considering it here.

Deliverables: The code for each method, submitted as separate files. For method 1, name the file `convex.py` or `convex.m` (extension depending on whether you used Python or MATLAB); likewise, for method 2, name the file `regress.py` or `regress.m`. Your codes should be easily readable. Please use comments wherever necessary. Also include a README file to briefly describe how to run the code. 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.