Data Mining (CSC 503/SENG 474)

Assignment 1 **Due on Sunday, February 5th, 11:59pm**

Instructions:

- You must complete this assignment on your own; this includes any coding/implementing, running of experiments, generating plots, analyzing results, writing up results, and working out problems. Assignments are for developing skills to make you strong. If you do the assignments well, you'll almost surely do better on the midterm and final.
- On the other hand, you can certainly have high-level discussions with classmates about course material. You also are welcome to come to office hours (or otherwise somehow ask me and the TAs things if you are stuck).
- You must type up your analysis and solutions; I strongly encourage you to use LaTeX to do this. LaTeX is great both for presenting figures and math.
- Please submit your solutions via Brightspace by the due date/time indicated above. This is a hard deadline; the penalty for lateness is getting a zero on the assignment because, as senior undergraduate students or graduate students, I know that you know the importance of deadlines. (In particular, when you submit on Brightspace, be sure to check that you did in fact submit and that you submitted the correct file.) However, I will make an exception if I am notified prior to the deadline with an acceptable excuse and if you further can (soon thereafter) provide a signed note related to this excuse.

This assignment has two parts. The first part involves implementing some of the methods that you have already seen and analyzing their results on a classification problem; this part is for all students (both CSC 503 and SENG 474). The second part is actually only for graduate students (only CSC 503). The first part might seem like a lot of work. However, if you do it well, you'll become strong, and also gain valuable skills for your project.

1 Experiments and Analysis

First, "implement" the following methods:

- **Decision trees (with pruning)**. I suggest using either reduced error pruning (this is the form of pruning we covered in class) or *minimal cost complexity pruning* (it's implemented in scikit-learn; see here). For the split criterion, you can use information gain or the Gini index or some other good criterion. You might even try a few different ones as part of your analysis (explained further below).

- **Random forests (do not use pruning)**. What forest size (number of trees) should you use? Well, you should experiment with different forest sizes and see what happens. For the number of random features d' when selecting the split feature at each decision node, I suggest starting at $\sqrt{d}$ (where d is the number of features) and experimenting by going up and down from there. What should be the size n' of the random sample (sampled *with* replacement) used to learn each tree? When you are doing an experiment that varies other parameters (like d'), please set n' to be equal to the original sample size; this way, each decision tree in the forest will be trained using a bootstrap sample.

- **Neural networks**. Any number of layers is fine, as long at there is at least one hidden layer; I suggest going with 3 layers (i.e. the input layer, 1 hidden layer, and the output layer). To get good results, check out the data preprocessing suggestions in Appendix A. There are a number of hyperparameters that you could play with when experimenting with neural networks. For example, one hyperparameter is the number of nodes in the hidden layer, another is the choice of nonlinearity, yet another is the regularization parameter (which modulates how much you penalize the size of the weights), and yet another is choice of optimization algorithm (common choices are stochastic gradient descent and Adam).

I put "implement" in quotes because I won't actually require you to implement these methods; you can instead use machine learning software that you find online. If you do implement anything, you can use whatever programming language you like. For neural networks in particular, I recommend using an existing implementation here, but eventually (after this assignment) for your own edification it would be great if you implement a neural network yourself.

One thing I actually *do* want you to implement is an out-of-bag (OOB) error estimate for random forests. What is an OOB error estimate? They are well explained in the textbook *The Elements of Statistical Learning*:

> "For each observation $z_i = (x_i, y_i)$, construct its random forest predictor by averaging only those trees corresponding to bootstrap samples in which z_i did not appear."

The purpose of an OOB error estimate is to allow you to estimate the risk (true error) of the random forest as you are growing it. It may seem daunting to compute an OOB error estimate. After all, for each example in the training set, you need to figure out all of the trees in the forest that were not trained on that example. To help out those who are using the random forest implementation from scikit-learn, we have provided some starter code; see the file `oob_error_starter.py`. This code will generate a list with one element for each tree in the forest; the j^{th} element is an

array of the example indices that were *not* used in the training of j^{th} tree in the forest. You will have a bit more work to do in order to figure out, for each example, which trees were not trained on that example.

What to test on

You'll be analyzing the performance of these methods on a binary classification problem. This problem comes from adult dataset from the UCI repository:

https://archive.ics.uci.edu/ml/datasets/adult

In order to help get you started, we have provided a slightly cleaned and modified dataset, `cleaned_adult.data`. In case you're interested, you can find details on how we prepared this dataset in Appendix B. The last attribute/feature (the label) takes values 0 and 1, where 0 indicates income "<=50K" and 1 indicates income ">50K". Keep in mind that the dataset has not been split into a training and test set. You should do this (after first shuffling the examples so that they are in a random order). For any training/test split, a good rule of thumb is 80% for the training set and 20% for the test set.

How to do the analysis

The core of this assignment, meaning what is worth the most marks, is the experiment analysis. Your analysis will go into a file called `report.pdf`. This file should:

- present the performance of the methods in terms of the training and test error on the problem. A good experiment involves keeping all but one hyperparameter fixed[1], and then varying one hyperparameter along the x-axis and plotting *both* training error and test error (ideally, with both curves in the same plot). One experiment I definitely want to see you do is to present plots that show how each of training error and test error vary with training set size. But beyond that, you should also play with the parameters of the methods to see the effect. For instance, what happens when you change the learning rate/regularization parameter (or number of nodes in the hidden layer, or the number of iterations of training) for the neural network? What happens if you change the number of random features used for the random forest? What happens if you change the pruning rule (or use a different split criterion) for the decision tree? As much as possible, try to present your results with plots. For each method, please experiment with at least two hyperparameters (and again, in addition, remember to show training/test curves as the training set size varies).

- for random forests only, you should show a plot with the OOB error estimate as you increase the number of trees in the forest. The OOB error estimate should be computed using your own implementation. For this experiment, it is important that as you increase the number of trees, the previous trees stay the same. That is, if you construct a random forest with 50 trees and another random forest with say 60 trees, then in both cases the first 50 trees are the same. There are two ways to accomplish this if you are using RandomForestClassifier() from scikit-learn:

    1. The first way is less efficient. You simply need to set the `random_state` parameter to the same number each time you call RandomForestClassifier(), i.e., for all values of `n_estimators` that you try.

[1] For the hyperparameters that are kept fixed, try to keep each one set to some sensible value; you don't need to do a full grid-search as that will be the focus of a later assignment, but try to pick something that seems to give OK results.

2. The second way is more efficient. You can set the `warm_start` parameter to True. If you stored the output of RandomForestClassifier in a variable `rf`, then when you want to increase the number of trees, set `rf.n_estimators` to a higher value and then call `rf.fit(...)` on exactly the same data as before.

- contain a detailed analysis of your results, including various design choices you made (like choice of split criterion for decision trees, choice of nonlinearity for neural networks, etc.). Try and explain the results that you got for the various experiments you ran, and use these results to compare the methods. Think about the best parameter settings for the methods (and maybe think about how the best parameter setting might change as the training sample size increases). Ideally, use your analysis to come up with ideas on how you might improve the methods. To help with organization, I suggest first presenting all experiments and analysis for the first problem and then presenting all experiments and analysis for the second problem.

Please make your analysis concise (yet thorough). Don't ramble, and don't make stuff up. Act as if you are a respected scientist presenting some work to your colleagues.

What to submit

In all, for the analysis portion of the assignment, you should submit (as a zip file):

- the file `report.pdf` explained above;
- a file called `README.txt` which contains instructions for running your code (for any code you wrote) or gives proper attribution to any code/datasets you got from other sources (like the Internet, for example). If you mostly used some existing code/software but needed to modify a few files, then give attribution and mention the files you modified.
- a file called `code.zip`, containing your code. Please organize this file well (embrace the idea of directories). In case you mostly used some existing code/software but needed to modify a few files, then just provide those files here, and make sure your `README.txt` mentions those files in relation to the existing code/software.
- any additional files that you need.

2 Problem-solving part - CSC 503 only

In this problem, we consider the gradient descent algorithm for a two-dimensional regression problem. So, we have 2 continuous features x_1 and x_2, and the task is to predict a continuous target y. Suppose that for a given input $\mathbf{x}_i = \begin{pmatrix} x_{1,i} \\ x_{2,i} \end{pmatrix}$, we predict using hypotheses of the following form:

$$f_{\mathbf{w},b}(\mathbf{x}_i) = w_1\, x_{1,i}^2 + w_2\, x_{2,i}^2 + w_3\, x_{1,i}\, x_{2,i} + b.$$

Assume that we have n training examples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$. Suppose that we measure the error according to the squared error with a further penalty on the squared Euclidean norm of $\mathbf{w}$. Then, for a fixed, positive number λ, the training error can be written as

$$\begin{aligned} E(\mathbf{w}) &= \frac{1}{2n}\sum_{i=1}^{n}(y_i - f_{\mathbf{w},b}(\mathbf{x}_i))^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2 \\ &= \frac{1}{2n}\sum_{i=1}^{n}(y_i - f_{\mathbf{w},b}(\mathbf{x}_i))^2 + \frac{\lambda}{2}\sum_{j=1}^{d} w_j^2, \end{aligned}$$

where $d = 3$.

In the below, assume the learning rate (also called the step size) is some positive number η.

(a) Derive the gradient descent update for w_1.

(b) Derive the gradient descent update for w_2.

(c) Derive the gradient descent update for w_3.

(d) Show the full gradient descent update for the vector $\mathbf{w}$.

(e) Derive the gradient descent update for the bias term b. What is your interpretation of the gradient of the training error with respect to b?

3 How you'll be marked

For both SENG 474 and CSC 503, the total number of marks is 100.

For undergrads (SENG 474):

- the analysis (in `report.pdf`) is worth 80 marks;
- you receive 20 marks for your code (a lot of these marks are for your implementation of the OOB error estimate).

For grad students (CSC 503),

- the analysis (in `report.pdf`) is worth 70 marks; you receive 20 marks for your code (a lot of these marks are for your implementation of the OOB error estimate);
- the Problem-solving part is worth 10 marks.

A Suggested preprocessing when training neural networks

A common issue when training neural networks is that good results can be elusive unless the data is suitably preprocessed. Below, I suggest two typical ways of preprocessing data:

- For each feature separately: shift and rescale the values of the feature so that, among the training examples, the feature's minimum value is 0 and its maximum value is 1. Be sure to apply exactly the same scaling to all the data (training and test data); therefore, the range of a given feature in the test set will not necessarily be $[0, 1]$.
- For each feature separately: shift and rescale the values of the feature so that, among the training examples, the feature's sample mean is 0 and its sample variance is 1. Again, be sure to apply the same scaling to all the data; therefore, a given feature's sample mean and sample variance in the test set are not necssarily 0 and 1 respectively.

B Cleaned version of Adult dataset

In order to obtain the provided cleaned version of the data, we started from the original dataset — consisting of `adult.data` and `adult.test` — which can be found in the Data Folder from the link above. You should *not* use this dataset, as the cleaned dataset we provide is much easier to use!

First, we combined the datasets `adult.data` and `adult.test` into a single dataset, hereafter called the "original dataset".

In the original dataset, some of the examples have one feature with the value '?'. This is a missing value. There are various ways to deal with missing values, but an easy way (when not many values are missing) is to simply discard those examples; this is what we did. In total, 3621 examples were discarded. Even so, the dataset is still large.

Some of the features are categorical. We used one-hot encoding to transform each such feature into several binary features.