Data Mining (CSC 503/SENG 474)

Assignment 1 **Due on Friday, January 31st, 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, the understanding and intuition you develop will allow you to do better on the midterm/final and probably better in life.
- 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 components. The first component is for all students (both CSC 503 and SENG 474); it involves implementing some of the methods that you have already seen and analyzing their results on a classification problem. The second component is for graduate students (CSC 503), but undergrads (SENG 474) can also complete it for bonus marks; it involves implementing and experimenting with the out-of-bag error estimate for random forests. The assignment will take serious effort. If you do it well, you will become much stronger and gain valuable skills for your project and beyond.

1 Component 1: Experiments and Analysis

First, "implement" the following three learning methods and cross-validation technique:

- **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.

- **Boosted decision trees**. Use AdaBoost with the weak learning method set to decision trees. For AdaBoost itself (the "outer" algorithm), you should experiment with the number of iterations (number of weak hypotheses). For the weak learner (the "inner" algorithm, which learns decision trees), you should experiment with the maximum depth. Please start with a maximum depth of 1 (which gives decision stumps), and try a few larger values as well to see how results change as the maximum depth increases.

- ***k*-fold cross-validation**. Please implement this yourself so that you actually know how it works. Various machine learning/statistics software include it, **but I still want you to implement it yourself. We will be checking that you coded this up.** *For all tasks involving using k-fold cross-validation, you will use the implementation you coded up.*

I put "implement" in quotes because I won't actually require you to implement decision trees, random forests, and AdaBoost; you can instead use machine learning software that you find online. Again, you *do* need to implement k-fold cross-validation. For any implementations you do, you can use whatever programming language you like. Decision trees, random forests, and AdaBoost are all available in scikit-learn.

For the methods above, in case you are unable to get any good results, try the data preprocessing suggestions in Appendix A (but also see the footnote there).

What to test on

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

`https://archive.ics.uci.edu/dataset/94/spambase`

You can read about the dataset at the link above. However, instead of using the version of the dataset at the UCI repository, you will use a custom version of the dataset created for this course. This version, which is available in the Brightspace page for this assignment, includes many additional features that were derived from the additional features (in case you are interested, this is explained in Appendix B). The last attribute/feature (the label) takes values 1 and 0, where 1 indicates "spam" and 0 indicates "not spam". 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` and consists of the following parts, which should be presented in separate sections, in this order:

Part I - Separate analysis. Present the performance of the methods (decision trees, random forests, and boosted decision trees) 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, for boosted decision trees, what happens when you change the number of iterations of training or change various properties of the weak learner (the decision tree)? For random forests, what happens if you change the number of random features? For decision trees, what happens if you change the pruning rule (or use a different split criterion)? 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 do an experiment showing training/test error curves as the training set size varies). I recommend presenting the results for each method in a separate subsection. That said, if you have a nice way to present them all together, you are welcome to do so.

Part II: Comparative analysis. This part involves using k-fold cross-validation to tune a hyperparameter (the size of the ensemble) for each of random forests and boosted decision stumps, followed by comparing the so-tuned methods' performance on a test set. To this end, fix k to a value between 5 and 10. *Pro-tip: experiments are likely faster if you set* $k = 5$. Now:

(i) For random forests: for all hyperparameters but the ensemble size, use a fixed, sensible setting. For example, you might use $d' = \sqrt{d}$ and, for the decision tree settings, perhaps use Gini as the impurity measure and do not put any constraint on the maximum depth. Select at least 10 values for the size of the ensemble (ideally, choose far apart values, or choose many values to cover a larger range), and use k-fold cross-validation to tune (i.e., to select a value for) the size of the ensemble.

(ii) For boosted decision stumps (AdaBoost with decision trees with maximum depth set to 1): just like with random forests, select at least 10 values for the size of the ensemble (again, ideally choose far apart values or choose many values to cover a larger range), and use k-fold cross-validation to tune the size of the ensemble.

[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 gives OK results.

(iii) Finally, compare the test error of your tuned version of AdaBoost to your tuned version of random forests. Be sure that this test set was never touched in steps (i) and (ii)!

For all sections of your report, you should present a detailed analysis of your results, including an explanation of various design choices you made (like choice of split criterion for decision trees, choice of the decision trees' maximum depth when varying the number of iterations in AdaBoost, etc.). You should not only plot results, but also discuss them. We are interested (and expect) to see your conclusions based on your experimental results, as well as your explanations for why you have those conclusions. For Part I, do your best to use the 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. By design, in Part II it should be much more straightforward to compare the two methods (random forests and boosted decision stumps).

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.

2 Component 2: Out-of-bag error estimate (Required for CSC 503/Bonus for SENG 474)

Implement an out-of-bag (OOB) error estimate for random forests (in this case, yes, you do need to implement it yourself). 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.

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.

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.

Present and discuss your results in a new section of the file `report.pdf`.

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.

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`) for Component 1 is worth 90 marks;
- you receive 10 marks for your code for Component 1;
- you can earn up to 20 bonus marks for Component 2. If you get more than 100 marks on the assignment, your extra marks can be used to compensate marks lost on a future assignment.

For grad students (CSC 503),

- the analysis (in `report.pdf`) for Component 1 is worth 70 marks;
- the code for Component 1 is worth 10 marks;
- the analysis (in `report.pdf`) for Component 2 is worth 10 marks;
- the code for Component 2 (your OOB error estimate implementation) is worth 10 marks.

A Suggested preprocessing

A common issue when training many machine learning methods 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 necessarily 0 and 1 respectively.

For this assignment, I expect that either method will work well, but I have experimented with the first method and know good results should be achievable using it.[2]

B Additional features

The original version of the Adult dataset contains 57 numeric features. Each of the first 48 features if the frequency of some word. The dataset you are working with includes the original 57 features. In addition, it has one new feature for every pair of the 48 words: for any pair of words, the new feature is the sum of those words' frequencies.

[2]For some methods, preprocessing may not be required; think about why preprocessing may not matter for a decision tree. Actually, for both decision trees and random forests, preprocessing might not be necessary!