IE 332 - Assignment/Project #2

Due: March 14th, 11:59pm EST

Read Carefully. Important!

As outlined in the course syllabus, this homework is worth 9% of your final grade. The maximum attainable mark on this homework is **315**. As was also outlined in the syllabus, there is a **zero tolerance** policy for any form of academic misconduct. The assignment must be done in groups.

You must use the provided LaTeX template made available on Brightspace to submit your assignment. The cover page provided must be included in both project and homework submissions. No exceptions. There are resources available on Brightspace for how to easily use LaTeX.

By electronically uploading this assignment to Brightspace you acknowledge these statements and accept any repercussions if found in violation of ANY Purdue Academic Misconduct policies. You must upload your homework on time for it to be graded. No late assignments will be accepted. **Only the last uploaded version of your assignment will be graded**. **NOTE:** Aim to submit no later than 30 minutes before the deadline, as there could be network traffic that could cause your assignment to be late, resulting in a grade of zero.

Assigning Credit to Individual Student Contributions

Given that the homework and project are group work, it is important that appropriate credit be assigned to each member of the group. This will be true for ALL submitted items, and the group will decide within itself the **MEANINGFUL CONTRIBUTION** of each member using a 100pt scale as shown in the example table below. **Every group member must contribute at least 10% effort on each question.** Changes (up or down) to individual grades may be implemented by the instructional staff if unequal contribution is identified. In the event student(s) are not equally contributing to all facets of the class at the expectation level of the group they may be fired from the group and will be assigned a grade of 0, at the discretion of the instructional staff. If issues arise at any point before submitting the work the group should report it to the instructional staff immediately instead of allowing the problem to persist through submission. **IMPORTANT: All students are expected to know all the homework and project material and it will be tested in individual exams and quizzes**. A similar evaluation will be included on the cover page of the project (see project submission instructions).

Ex. For a group of three students A, B, C and 2 questions:

Student	Q1	Q2	Overall	DIFF
A	50	33	83	17
B	50	33	83	17
C	0	33	33	-33
St Dev	29	0	29	29

Overall is the sum of the total effort of each student. DIFF is the difference from equal contribution, and for each member is calculated as their overall score - 100*(# of questions) / (# group members), rounded to the nearest whole number. The last row is the rounded standard deviation of the respective column.

Assignment 2

0.1 Complexity

1. Analyze the complexity of the following algorithms. Determine Θ.
   (a) (10 points) **Require:** n is a positive integer

```
Function f(n)
for i = 1 to n do
    j = 0
    while j ≤ n/i do
        g(j) (Has time complexity of Θ(1))
        j = j + 1
    end while
end for
EndFunction
```

   (b) (10 points) **Require:** n is a positive integer

```
Function f(n)
for i = 1 to n do
    for j = 1 to √n do
        h(j) (Has time complexity of Θ(1))
    end for
end for
EndFunction
```

   (c) (10 points) **Require:** n is a positive integer and $n \geq 3$

```
Function f(n)
for i = 3 to n do
    j = 0
    while j ≤ n/i do
        g(j) (Has time complexity of Θ(1))
        j = j + 1
    end while
end for
EndFunction
```

   (d) **Require:** x, as a number with value > 1.

```
Function f(n)
while X > 1 do
    if x%%3 == 0 and x%%2! = 0 then
        X = 2^X
    else if x%%2 == 0 then
        X = g(X)(returns X/2, runs in log(n) time)
    else

    end if
end while
EndFunction
```

   (e) (20 points) **Require:** x, as a list of numeric elements with length $n > 0$

```
Function f(n)
```

2: $i = 0$
3: **while** $i < length(x)$ **do**
4: **if** $imod2 == 0$ **then**
5: $append(x, x[i]^2)$ (assume this takes 1 operation)
6: $x = g(x)$ (runs in n^2 time)
7: **else**
8: $pop(x)$ (assume this takes 1 operation)
9: $x = g(x)$ (runs in log(n) time)
10: **end if**
11: $i+ = 1$
12: **end while**
13: **EndFunction**

(f) (40 points) Given the following 3 algorithms, explain what they're supposed to do, and analyze their runtime complexity and provide their T function for the worst case. For any steps with multiple operations, write an algorithm to achieve that step, and determine the run-time complexity for that step for your analysis along with the algorithm.

Require: x, as a list of numeric elements with length > 0 Assume this array is indexed at 1 instead of 0.

1: **Function** f(n)
2: n=length(x)
3: **for** $i = 0$ to n-1 **do**
4: $k = findmin(x[1 : n - i])$ (Not one operation, returns a tuple: [value,index])
5: delete(x,k[2]) (Not one operation!)
6: push(x,k[1]) (Not one operation!)
7: **end for**
8: **EndFunction**

Require: x, as a list of numeric elements with length > 0

1: **Function** f(n)
2: n=length(x)
3: **while** $i < length(x)$ **do**
4: j=i
5: **while** $j > 0 \wedge x[j-1] > x[j]$ **do**
6: $t = X[j-1]$
7: $X[j-1] = X[j]$
8: $X[j] = t$
9: j = j-1
10: **end while**
11: i=i+1
12: **end while**
13: **EndFunction**

Require: x, as a list of numeric elements with length > 0

1: **Function** f(n)
2: n=length(x)
3: swapped=true

```
while swapped do
  swapped=false
  for i in 1:n-1 do
    if x[i − 1] > x[i] then
      t=x[i-1]
      x[i-1]=x[i]
      x[i]=t
      swapped=true
    end if
  end for
end while
EndFunction
```

(g) (40 points) Plot T, and determine if/when the T functions intersect. What implications does this have for selecting one of these algorithms to solve a problem?

Loop Invariants

2. Given the following code attempt at a Monte Carlo algorithm to approximate π (pi), perform the following.

(a) (10 points) Find the loop invariant, if it's wrong fix it. If not, state what it is.

Require: x, as a number of attempts

```
Function f(x)
count=0
i=0
while i < x do
  xc=runif(-1,1)
  yc=runif(-1,1)
  if xc^2 + yc^2 < 1 then
    count++
    i++
  end if
end while
println(count/i*4)
EndFunction
```

(b) (30 points) Similar to the lab questions, refactor the imperative code using higher order functions like map, reduce, filter, or equivalent `purrr` options.

(c) (50 points) Now, write the code recursively, do you notice anything regarding the recursive guard expressions and the loop invariant? Explain. (Note, R sucks at tail-recursion, so your recursive function will likely blow the stack. This is one of the reasons invariants and proofs are so important, you may not always be able to test your code effectively for a given language, but this is an implementation detail!)

Decision Boundary and Overfitting

3. **Overfitting Study: in a decision tree**

The students are given a dataset with a feature dimension of m and some of the features are not necessarily independent from each other.

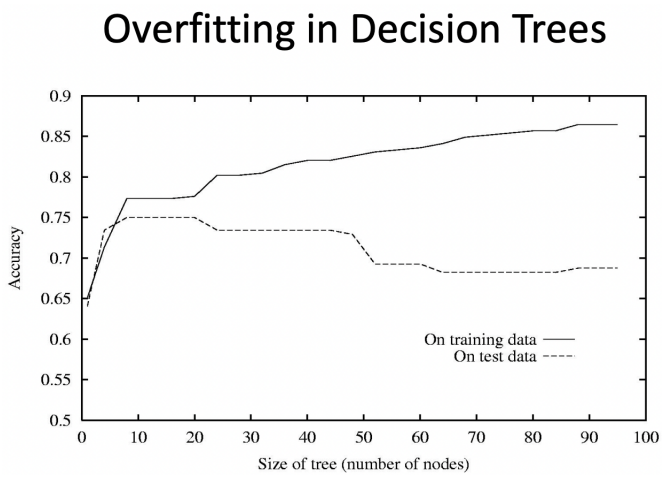

Figure 1: Example of overfitting in a decision tree https://courses.cs.washington.edu/courses/cse446/20wi/Lecture4/04a_Overfitting.pdf

(a) (5 points) Choose two subsets of data from the aps_failure dataset as the training and test datasets. The size of each dataset will not exceed the original feature dimension m. Then train a decision tree and apply it to the test dataset. Finally, calculate the false negative rate for both the train and test datasets to evaluate the overfitting behavior as in Figure 1 by replacing the y-axis accuracy with the **false negative** rate.

Note:

- Use `rpart` function from `rpart` library to train the decision tree model
- the `maxdepth` parameter in the `rpart` function could be adjusted to represent the changes in the size of the tree, which is the x-axis in Figure 1.

Deal with overfitting in a decision tree. The overfitting problem in a decision tree model could be alleviated by pre-pruning(also called early stopping and it stops further splitting the tree when no further significant statistical improvement could be achieved), post-pruning (first growing the tree to its full-size then trimming less significant nodes), or using a random forest (ensembling).

(b) (5 points) Explain the `cp` parameter in the `rpart` function and implement pre-pruning by tuning its value. Redraw the figure between the false negative rate and the size of the decision tree.

(c) (5 points) Implement post-pruning in R and redraw the figure between the false negative rate and the size of the decision tree.

The following links could be helpful for parts b and c.

- https://statinfer.com/203-3-10-pruning-a-decision-tree-in-r/
- https://dzone.com/articles/decision-trees-and-pruning-in-r
- https://www.edureka.co/blog/implementation-of-decision-tree/
- https://www.learnbymarketing.com/tutorials/rpart-decision-trees-in-r/

(d) (5 points) Based on the pruned decision tree model from parts b and c (keep the tree structure from parts b and c unchanged), adjust your train dataset used in part a to help alleviate the overfitting problem. The following link could be helpful. https://datascience.stackexchange.com/questions/14875/data-set-size-versus-data-dimension-is-there-a-rule-of-thumb. Redraw the figure between the false negative rate and the size of the decision tree.

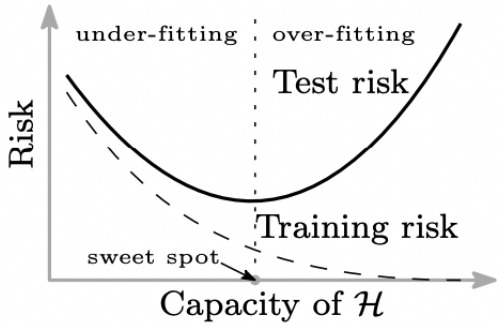

Figure 2: The classical U-shaped risk curve arising from the bias-variance trade-off. https://arxiv.org/pdf/1812.11118.pdf

(e) (5 points) From parts b, c and d, what characteristic of the decision tree model, and of the data itself do you think is impacting the overfitting behavior? Figure 2 may be helpful.

(f) (5 points) Train a random forest model using the `randomForest` library and the dataset from part d. Identify the two most important variables in this dataset. This link may be helpful: https://statinfer.com/203-3-10-pruning-a-decision-tree-in-r/.

(g) (10 points) Plot the decision boundary of the decision tree model from parts b and c based on the two most important variables concluded from part f. This link may be helpful: https://paulvanderlaken.com/2020/03/31/visualizing-decision-tree-partition-and-decision-boundaries/

4. **Underfitting and overfitting Study in a deep learning network**

Now we are applying the same datasets used in the previous problem to a deep learning neural network

(a) (5 points) First, use a neural network that only has one hidden layer and five hidden nodes. Evaluate overfitting/underfitting behavior by comparing the training performance and testing performance over the number of epochs

(b) (5 points) Gradually increases the complexity of the neural network and draw a curve between the complexity of the network and the training performance, and a curve between the complexity of the network and the testing performance

(c) (5 points) Does the results from part b in this question aligns with your answer from part e in question 3?

5. **The use of clustering algorithm** The students will be given a real estate price dataset

(a) (10 points) Apply a clustering algorithm on your dataset

(b) (10 points) What conclusions can you draw from your clustering algorithm implementation? Could you predict which cluster of real estate prices is most likely to go up?

(c) (10 points) Train a regression machine learning algorithm on all data and report the performance

(d) (10 points) Train a separate regression machine learning algorithm for each cluster and report the performance

Project 2

For this project, you're going to train a voting based optimization algorithm to perform adversarial attacks on a binary image classifier. Before further characterizing the problem, you should be aware there is an opportunity for bonus points: the team that makes the "best" algorithm gets 5% bonus to their (project grade), 2nd place gets 4%, all the way until 5th place with 1%. The point of this is to encourage you to test multiple configurations of your algorithm, noting both their accuracy, runtime complexity, and average wall-time (the time it takes the program to run).

Majority Voting Classifier

A Majority Voting Classifier is a compound classifier constructed from a pool of classifiers and the final output of this compound classifier is decided by the majority votes. There are various forms of majority voting classifiers and the simple majority classifier and the weighted majority classifier are two common ones among them.

- Simple majority classifier: every individual classifier votes for a class and the majority (the mode) wins

$$C(x) = mode\{h_1(x), h_2(x), \cdots, h_n(x)\}$$

  where $h_i, i \in 1, \cdots, n$ are a pool of n classifiers, $h_i(x)$ is the prediction result from classifier i, and $C(x)$ is the final prediction decision of the majority voting classifier

- Weighted majority classifier: every individual classifier is assigned an importance weight and the target class with the greatest weighted-sum probabilities wins.

$$C(x) = \arg\max_j \sum_{i=1}^{n} w_i \mathbb{1}(h_i(x) == j)$$

  where $w_1, \cdots, w_n$ are weights that sum up to 1 and $\mathbb{1}(h_i(x) == j)$ is an indicator function decides whether classifier i votes for class label j

The scheme is straightforward: there is a link to a pre-trained image classification model on Brightspace that accepts an image and attempts to classify it as either X or Y. It does this classification based on the preponderance of the evidence, that is 51% probability of an image being an X means the machine will classify it as a X.

Your job is to use 5 different machine learning and/or optimization algorithms to independently try to fool the classifier, but with a twist: these 5 algorithms will be placed within another algorithm that must assign weights to the other 5, along the lines of the weighted majority classifier. The result should fool the provided image classifier with exactly B (the pixel budget) pixels changed in the image. This means, it assigns weights to the different algorithms based on some expected performance given the image; after the algorithms vote on which set of pixels to change, the optimizer must select from this set which pixels to actually use. The pixels' location, amount of color shift, etc. don't impact the budget for this particular project, only the absolute number of pixels used to fool the classifier. The input to your adversarial function should be both an image and a scalar, which tells the algorithm the expected budget in terms of the ratio of the image size (e.g. 0.01 uses 1% of available pixels, 1/P uses only 1 pixel).

We will provide you a set of training data on Brightspace on which the classifier has 100% accuracy, and we will reserve a set of that 100% accurate data to evaluate your models on once you submit your projects.

Note: to qualify for any credit on the project (not just bonus credit!), your model must be able to successfully fool the classifier with a budget of P/100, for at least one image, where P is the number of pixels in the image, and do this in less than 10 seconds per image. In other words, with a 1920x1080 image, you should be able to fool the classifier by changing 20736 pixels, or 1% of them, in less than 10 seconds after your model is fully trained.

Algorithm performance will be scored as such: $\sum_{b=1}^{0.01*P} f * \frac{P}{b}$, where f is the number of successfully fooled images at a given budget level, b. As you can see, the smaller the budget, the more valuable each successfully fooled image is, and this scales with the size of the image. This score is used only for the competition between groups, and does not have a direct impact on your final grade, unless your solution fails to fool the classifier at all.

What will primarily determine your grade is a complete justification of your rationale for selecting each of the 5 machine learning algorithms, how you trained them, and how your optimizer allocates the votes, and why such an approach should work.

Finally, be sure to get started on this early, it will take a while if you are not already familiar with machine learning and optimization, and leveraging the lecture and lab time appropriately is critical to making sure your project gets completed on time.

Project Requirements

All code, notes, documentation, etc. must be tracked using Git and Github for verification of the effort contributed by each team member. For a quick primer, check out the following video. All experimental variants of your algorithm (e.g. with different sub-algorithms, and so forth) should be tracked with separate branches and the best one should then be merged into main. Repeat this iteration process until you're completely satisfied with your final algorithm. Your repositories must be public, but not shared with anyone other than the IE332 instructional staff until the project deadline, and any shared code/text/etc between projects will result in an immediate fail.

All code, barring the classifier model we send to you, must be in R, LaTeX, or some combination of the two (see Report requirements for clarification on this point). You should be using a recent version of R, ideally 4.2.2 and up. Any R packages are allowed, so long as they can be directly installed from R's install.packages function.

Report Requirements

Reports, as in last time, must be written in LaTeX (or RMarkdown, if you want to deal with the hassle of setting it up). All changes to the report must also be tracked in your Github repository. You must justify your selection of algorithms, do correctness proofs on your code, and complexity analysis of your overall algorithm. Keep in mind your algorithm's performance is crucial, so justifying efficiency trade-offs in light of performance is crucial, so make sure you document all iterations of your algorithm and why you took that specific development path.

The report should follow similar length guidelines to the previous report for both the main text, and the appendices; however, in this case the appendices will include Testing/Correctness/Verification (1 section), Runtime Complexity and Walltime (1 section), Performance (according to the aforementioned criterion), and justification for the algorithm selected in your final implementation out of the set of implementations you tested.

You may have noticed the guidelines for this project are seem more open-ended, but that is not due to reduced expectations. Rather, it is designed to grant you freedom to explore the problem and solution space, and report any ideas, interesting insights, and rationale you uncover for tackling the problem. Any useful figures, plots, tables, or analyses are allowed to illustrate your level of depth at which you thought about the problem and your solutions.

All code must be submitted along with your report, and should be well-commented, ideally with type signatures, for an example of good commenting practices, I suggest taking a look at both Julia examples (I think this package has superlative commenting practices, including references, examples, dispatch versions, optional arguments, and more!) and Haskell Type Signatures, which can give your team ideas at a glance for what the inputs/outputs/forms of your functions are. For clarification on this point, feel free to ask in lecture or in the lab sections.