INFO371 Problem Set 5

Your name:

Deadline: Mon, Mar 5th, 10:30am

Introduction

Please submit a) your code (notebooks, rmd, whatever) *and* b) the results in a final output form (html or pdf). You are free to choose either R or python for solving this problem set.

You are welcome to answer some of the questions on paper but please include the result as an image in your final file. Note that both notebooks and .rmd are just markdown documents (besides the code), and hence it's easy to include images.

Also, please stay focused with your solutions. Do not include dozens of pages of computer output (no one is willing to look at that many numbers). Focus on few relevant outputs only.

Working together is fun and useful but you have to submit your own work. Discussing the solutions and problems with your classmates is all right but do not copy-paste their solution! Please list all your collaborators below:

1.
2. ...

Trees and Forests

The main task in this problem set is to implement a decision tree algorithm, and use this to predict Titanic survival. I also ask you to implement *bagging* and *random forest.*

There are several good sources to learn about the methods, I recommend (James, Witten, Hastie, and Tibshirani, 2015, ch 8) (available on canvas).

1 Explore the Data

First, load and explore the data. Feel free to copy-paste this step from your previous work if you already have explored the Titanic data.

1. Load the data. Make sure you know the coding of all variables. In particular, you should be aware if a variable is categorical or numeric. Explain the coding scheme if it's not obvious.

As most of the variables in the data are categorical, we cannot just compute means and correlations. Rather create a table along these lines where you list the averages for those who survived and those who died:

variable	survived	drowned	missings
pclass	1.96	2.5	0
sex = male	0.322	0.843	0
...			

Note: means of the respective variables by group.

The idea is to provide the reader with the information about variables, how these are correlated with survival, and the main categories. The table above suggests that upper classes (lower class number) had higher chances to survive, and men were less likely to survive.

This involves quite a bit of manual work: for instance, if you list males, there is no need to list females. You should *not* list all homes and destinations (there are hundreds of these), but you may list a few more common ones. You should also consider which variables you want to list here (ticket number is probably not interesting, unless it tells something like "how close were you to the boats").

2. Create a summary table where you show means, ranges, and number of missings for each variable. In addition, add correlation between the price and the the variable. You may add more statistics you consider useful to this table.

2 Implement decision tree

Now let's move step-by-step. Let's stay with binary-only variables.

In order to simplify the evaluation, let's split the data into just training and testing parts and avoid cross-validation.

2.1 Create binary variables

As most of the variables are not binary, we have to create a few new binary ones.

Note: you can also choose to split non-binary variables. This is slightly more complex to do (but just slightly), if you go for this approach, you don't have to create the dummies.

1. Create binary variables (dummies) for pclass: pc3 (in class 3), pc1 (in class 1) and pc2 (in class 2).
2. Create the age dummies in a similar fashion. You can pick how to cut the age range, but something like $(0, 10), [11, 18), [18, 45), [45, 60), [60, \infty)$ may be a good choice. You create an indicator for each passenger that shows if the passenger was in this age group.
3. You may create more binaries, for instance *sibsp* or anything else you find potentially informative.

   Note: *do not* include explanatory variables that are the result of survival/death, namely boat number and body number.
4. Split your data into training/testing groups (80/20% or so).

2.2 Implement the decision tree with binary variables

Now we'll get serious. Your task is to implement decision tree by recursive binary splitting by including all the binary variables above and outputting the prediction: survived or drowned.

1. Compute the entropy at the root, based of the total number of survivors and victims. You will use this number to compute entropy gain further below.
2. Repeat the following with each feature you consider:
   (a) Split the data according to 0/1 answer to this feature.
   (b) Compute entropy on both subsets
   (c) Compute the weighted average entropy
   (d) Compute the entropy gain.

3. Now pick the feature that gives the largest entropy gain. This gives you the first split (tree stump). Print the survived/non-survived percentage in both branches.

And now the most important and complex part. Repeat the previous steps over-and-over again until you either a) get a pure leaf; b) run out of data (number of observations in your leaf is too small, say < 5); or c) you run out of variables: there are no more meaningful variables to split along.

But it is not enough to do the splitting and all that, you have also to create a tree, and later be able to use this for predictions. I recommend to use nested lists for this, but you may also use some sort of tree data structures, or graphics if you wish. A potential way to do it in form of lists is (R syntax for more verbosity):

```r
list(variableName, split0, split1)
```

where `split0` and `split1` are the respective subtrees in case the tree branches here, and

```r
list(0.8)
```

where 0.8 is the probability someone survives in this leaf. Now you can distinguish between leafs and nodes by looking at the length of the corresponding subtree.

4. Create such a recursive function we discussed in the class that takes in data and returns a tree.

   Note: before you run out of data or features, you may encounter situation where all features left over from previous splits have identical value. For instance, if we select older passengers, all younger age group indicators will be 0. In such a case there is nothing else to do than to call a leaf–all these passengers are observationally equivalent.

5. Print your tree. Try to make the result as nice as you can (I don't ask anything like fancy graphical results) but ensuring that one decision is on a single line, and adding a little manual indentation will make a large step toward making the tree readable.

   Comment the outcome. Does it make sense? Which variables seem to be more important?

6. Predict (based on training data) using your tree. Compute accuracy, precision and recall.

   Note: you have to create a function that walks through the nested list based on a single observation. As the decisions differ between individual observations, we cannot easily vectorize the operation. A set of loops is necessary (can be done in parallel).

3 Bagging and Random Forests

So far we just estimated a single tree without any pruning. The test results were probably mediocre. Now the task is to improve the performance using bagging and random forests.

3.1 Bagging

First, let's do bagging (bootstrapped aggregating). It means creating a number of trees, using a different bootstrapped dataset each time. The final prediction will be done by majority voting between individual predictions.

To get a bootstrapped dataset you start with your (training) data, and each time you select N observations out of N *with replacement*. This means we'll have some observations in the data repeatedly while some are missing.

1. Create a small number B of bagging trees ($\mathsf{B} = 5$ is a good choice).
2. predict (on test data) the survival according to each individual tree.
3. Show some aggregate statistics: in how many cases all trees agree?
4. Find your final prediction: the majority vote over all trees. Compute accuracy, precision, recall. Did you get better results than in case of the single tree?

True bagging involves many more iterations, potentially thousands. How many you should use to get substantial improvement. Or maybe bagging does not help at all here?

5. Repeat the process above with a large range of B, say, between 1 and 300. (Hint: you don't have to test every single number between 1 and 300. Take a step that makes the process not too slow on your computer.) For each B, compute accuracy, precision, recall.
6. Show on a figure how A, P, R depend on the B.
7. Discuss your findings. What is the optimal bag size B?

3.2 Random Forests

Boosting helps but it does not use much more information than a single tree. Random forests approaches the issue by only considering a random sample of possible features as split candidates, where the sample should be less than all potential candidates (otherwise we are back to bagging). A good choice is to choose sample size $\mathsf{m} \approx \sqrt{\mathsf{p}}$ where p is the total number of features to consider.

1. Implement random tree algorithm by changing your previous `growTree` algorithm. We refer the new version as `growForestTree`.

   Note: depending on how you implemented the data structures and the prediction algorithm, you may have to adjust the prediction function too.
2. Create random forests of different size R, say, between 1 and 1001. For each R, compute accuracy, precision, recall.
3. Show on a figure how A, P, R depend on the B. Include the bagging outcomes on the same figure too.
4. Discuss your findings. What is the optimal forest size? Are random forests superior to bagging?

References

James, G., D. Witten, T. Hastie, and R. Tibshirani (2015): *An Introduction to Statistical Learning with Applications in R*. Springer.