INFX 574 Problem Set 6

Your name:

Deadline: Fri, May 30th, 8:30pm

Introduction

Please submit a) your code (notebooks or whatever you use) *and* b) the results in a final output form (html or pdf).

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 notebooks 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*. Note: this is about *implementing* the algorithm, not just using an existing library.

Before you start I recommend you to become familiar with (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 survivors had lower class numbers, i.e. they were more likely to travel in upper classes, and there were less men amont the survivors.

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 such a summary table. You may add more statistics you consider interesting to this table.

2 Implement decision tree

Now let's move step-by-step. Let's stay with ordinal variables only, i.e. such variables that can be ordered. You can order a) all binary variables (such as sex); b) all continuous variables, such as age; and c) ordered categorical variables, such as *pclass* (1st, 2nd, 3rd, but note that for datasets where *pclass* also includes crew, we cannot order it).

In order to simplify the evaluation, we split data into training and testing samples and avoid cross-validation.

2.1 Prepare data

1. Now select the variables you are going to use. Use those that may have influenced survival (such as class), or those that may be otherwise correlated with survival (like *fare*).

   Note: *do not* include explanatory variables that are result of survival/death, namely boat number and body number.

2. Split your data into training/testing groups (80/20% or so).

2.2 Implement the decision tree

Now we'll get serious. Your task is to implement decision tree by recursive splitting including all the variables you selected above. The tree should predict survival. You'll work only on the training set here.

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.

Next, let's find the optimal split according to age:

2. create an ordered vector of unique age values $(a_1, a_2, \ldots, a_k)$. These will form the potential split points for age. How many different age values do you find in your training data?

   Note: remember that there are missing age values!

3. Split the (training) data into two groups:

   (a) age $\leqslant a_1$ and

   (b) age $> a_1$.

4. Compute the entropy of this split.

5. Now repeat these steps with all possible age splits. Find the best age boundary, the split that gives you the lowest entropy.

This will be your first split in case you decide to start splitting your tree along *age*. But perhaps another variable is better? Let's check it out!

6. Repeat the previous steps over all explanatory variables in your data. For each variable, store (and print) the best split value and the corresponding entropy.

7. Now pick the feature that gives the largest entropy gain. This gives you the first split (tree stump). Show the name of the feature, the resulting entropy, and the split position.

   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 using data in both branches you split as inputs in the next step until your tree is ready.

When is the tree ready? There are a number of options:

i. you get a get a pure leaf (only survivors or victims in the leaf)
ii. you run out of data (number of observations in your leaf is too small, say $n < 5$)
iii. you run out of variables: there are no more features to split along.
iv. you still have features left, but there is no variation. For instance, you extract a small age group and find that there are no men in there.

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 graphs if you wish. A potential way to do it in form of lists is (R syntax for more verbosity):

```
list(feature, splitpoint, split0, split1)
```

for instance

```
list(age, 20, dl, dr)
```

where `dl` and `dr` are the respective subtrees (left and right branch) in case the tree branches here, and list of lenght one is a leaf, for instance

```
list(0.8)
```

means the probability of survival for this branch is 0.8. Now you can distinguish between leafs and nodes by looking at the length of the corresponding subtree.

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

9. Visualize the two top levels of branches of your tree. Try to make the result readable (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?

10. 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* (see James, Witten, Hastie, and Tibshirani, 2015, section 5.2). This means we'll have some observations in the data repeatedly while some are missing.

1. Create a small number B of bagging trees ($B = 5$ is a good choice).

   Note: as your tree algorithm may be slow, use this to measure and boost the speed. Suggestions to improve the speed are a) use fewer features (but keep age, class, sex); b) use a larger minimum number of observations for splits (say, 100); and c) run your tree-growing code in parallel.

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. Choose numbers that make this 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 $m \approx \sqrt{p}$ 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 B. Ideally you want to go up to $B = 1000$, but you may need to keep the number much smaller in order to finish in a reasonable time. For each B, 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.