

Decision Trees, ID3, KNN

This Project involves the implementation of ID3 Decision Tree and the matching KNN Forest to predict whether an individual is healthy/unhealthy based on various parameters provided in the dataset. It includes training and analyzing the results, and adding various improvements (KFOLD, pruning etc).

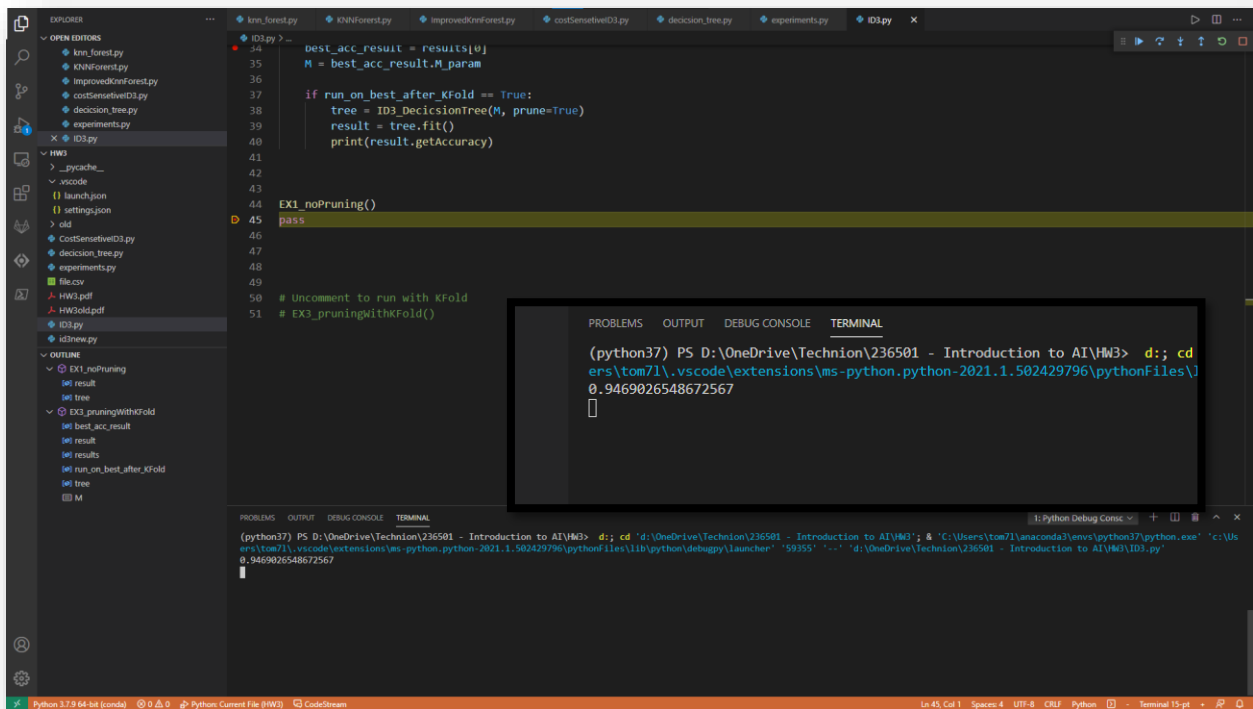
The project doesn't use any external libraries, and contains the entire implementation of ID3 from scratch (only Numpy and Pandas were used to process the data).

SECTION 1 – ID3

Required to implement the ID3 Algorithm with no pre/post pruning or any modifications. The algorithm is using a train.csv and test.csv files as its data sets.

The features in the data sets are numeric, non-normalized, positive values. Therefore, we use a continuous threshold values to split the data and test samples in each node in our Decision Tree. The threshold used for each split is calculated by an entropy, measuring the expected information gain (IG), this is calculated by computing the homogeneity of a each sample.

The accuracy of the ID3 implementation is 0.946 as shown in the screenshot below.



```
EXPLORER
  OPEN EDITORS
    knn_forest.py
    knn_forest.py
    improvedknnforest.py
    costSensitiveID3.py
    decision_tree.py
    experiments.py
    ID3.py
  HW3
    __pycache__
    .vscode
    launch.json
    settings.json
    .id3
    CostSensitiveID3.py
    decision_tree.py
    experiments.py
    file.csv
    HW3.pdf
    HW3old.pdf
    ID3.py
    id3new.py
  OUTLINE
    EX1_noPruning
    result
    tree
    ID3_pruningWithKfold
    best_acc_result
    result
    results
    run_on_best_after_Kfold
    tree
    M

ID3.py
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51

best_acc_result = results[0]
M = best_acc_result.M_param

if run_on_best_after_Kfold == True:
    tree = ID3.DecisionTree(M, prune=True)
    result = tree.fit()
    print(result.getAccuracy())

EX1_noPruning()
pass

# Uncomment to run with Kfold
# EX3_pruningWithKfold()
```

```
TERMINAL
(python37) PS D:\OneDrive\Technion\236501 - Introduction to AI\HW3> d;; cd
ers\tom71\.vscode\extensions\ms-python.python-2021.1.502429796\pythonfiles\
0.9469026548672567

(python37) PS D:\OneDrive\Technion\236501 - Introduction to AI\HW3> d;; cd
ers\tom71\.vscode\extensions\ms-python.python-2021.1.502429796\pythonfiles\
0.9469026548672567
```

SECTION 2 – ID3 WITH PRE-PRUNING

2.1 – The Significance of Pre Pruning

The significance pre-pruning derives from its ability to reduce the size of the tree, and removing redundant nodes that do not benefit the learning process. Pre pruning counters the effect of **overfitting**, which renders the classifier too complex to reliably predict future observations or fit data accurately.

2.3 – ID3 KFold Pre Pruning:

Upon reaching a node, we wish to decide the minimal number of samples for which we want to set a classification (and not building the tree further as opposed to ordinary ID3).

3. The parameters chosen for pruning the tree are marked by M and are as follows:

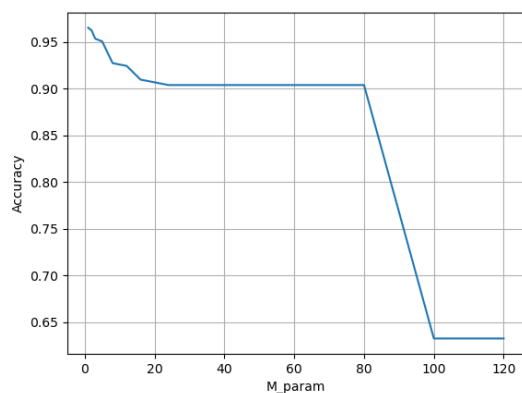
M_params = [1, 2, 3, 5, 8, 12, 16, 24, 32, 48, 64, 80, 100, 120]

4. The graph shows us a relatively optimal results for smaller M values, while for bigger M values we can see a decrease in the classifier's accuracy.

Best result = 0.98~ as seen in the screenshot:

```
C:\Users\user> python3 (extension) (ms-python.python-2021.1.502425750) (python file)
M=1|P=1|R=0 | LOSS: 0.03333333333333333, ACC: 0.927536231884058
M=1|P=1|R=0 | LOSS: 0.014492753623188406, ACC: 0.9855072463768116
M=1|P=1|R=0 | LOSS: 0.01594202898550725, ACC: 0.9710144927536232
```

KFold graph with pruning W.R.T M_PARAMETER



We can conclude from the graph that the higher the M, the less accurate the results. The reason this happens is because we were pruning the tree way too early resulting in inaccurate classifications of leaves.

2.4 – ID3 Best Params:

The best M parameter for pre pruning chosen is 1, although similar results were also received for M =2. For M=1,2,3 pruning did improve the accuracy by around 2%!!

SECTION 3 – COST SENSITIVE LEARNING

3.1 – LOSS VALUE KFOLD:

Presenting a solution for ID3 with pre-pruning utilizing KFold and evaluates Loss Value.

The loss value is calculated as follows:

$$loss(C) = \frac{(0.1 False_{positive} + False_{negative})}{n}$$

Where C is the classified, and n is the number of test samples measured.

3.2 – Loss Minimalization Modification:

I will present a modification to the original ID3 utilizing 3 combined methods to achieve minimal loss value:

- 1. A Modification to the entropy calculation. Marked by P_Parameter**
During the calculation of probabilities in the entropy calculation phase, we will evaluate the weight of each threshold and IG value by how much we are likely to classify the sample as sick.
This will decrease the Information Gain for solutions with large amount of sick samples, allowing us to prioritize solutions that are less likely to be classified as sick.
The logic is that we want to minimize the amount of times we classify a sample as **False Positive**. Therefore we prefer solutions that are less likely to be classified as such to begin with.
The **P parameter** multiplies the probability of a class to be classified as sick, and as such the IG will decrease, and less likely to be chosen. I've noticed that using this parameter, combined with the other techniques gives a loss value of 0, and accuracy of 1.0, I tested it several times but could not find a bug behind it.
- 2. Early Pruning modification by percentage of sick samples in the parent node in relation to the current node's amount of samples.** Marked by R_parameter, which basically means that if the parent node has a big amount of sick labels, for example, more than 30% of the labels are 'M', then we classify the current node as sick aswell. This is another modification that allows us to prefer making a mistake that isn't as costly instead of classifying a sample as **FP**.
- 3. Original early pruning technique with M_parameter.**

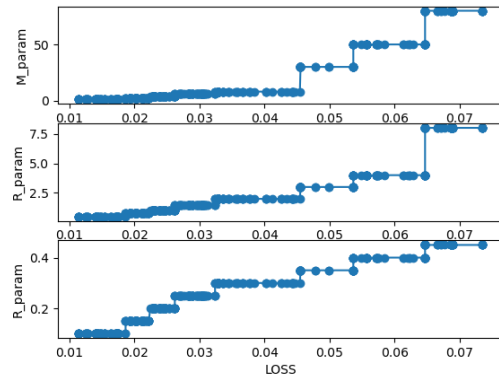
Other techniques tried:

- I've also tried adding randomness for taking a random subgroup of features instead of using all of the features. I sometimes got a very very small improvement (from ~0.0015 to 0.0008) but in some other cases the result was worse (probably due to the randomness factor of the subgroup).
- Post Pruning resulted in a massive decrease of both accuracy and loss.
- Early pruning of the data set, removing identical lines or samples that have common feature value but classified differently also resulted in no significant improvement if any at all.
- Also tried replacing IG entropy with GINI INDEX entropy, the results were way worse than with IG.

3.3 – Graph Analysis for Loss Minimalization:

Using KFold to find the best value. It appears that combining all three methods mentioned above yields a loss value of ~ 0.0017 , while in some cases and for some parameters, the loss value is **0.0** and accuracy is **1.0**. This was achieved using $M = 8$, $P=8$, $M=0.3$, multiplying the weight of probability by 8, and pre-pruning the tree if a node's parent consists of more than 30% sick samples.

Best params used and their results from KFold.



We can see that there is a correlation between the loss increase and the parameters increase.

Ultimately, the best parameters for M were in the range of 8 to 30, for P in the range of 5 to 9 and for R between 0.2 and 0.3. Eventually I chose $M = 8$, $P = 8$, $R = 0.3$.

SECTION 4-5 – KNN FOREST

6.1 – Normal KNN Implementation:

The implementation of my KNN forest uses a data structure that builds a forest out of normal ID3 Decision Trees as implemented in section 1. I've used KFold to experiment on values, I've ran the KFold with various parameters of the amount of N trees to build, K closest trees to evaluate, and different values of parameter P which is anywhere between 0.3 and 0.7.

Eventually I chose the values $N = 60, K = 20, P = 0.45$, the best P was yielded by KFold, I also received multiple results around the specified $N = 60, K = 20$ but eventually I found out that it was more consistent when the numbers are around that size. I concluded that a big enough number (although not too big) yields stable results since there is less room for error and each prediction on a tree with a sample has a smaller impact, minimizing error for the accuracy.

Best accuracy achieved was 0.9911504424778761 although it was not always the case due to randomness of the chosen data that was used for building the trees.

7.1-7.2 Improved KNN Implementation:

As mentioned before, the results were consistent enough around the parameters chosen and the maximum accuracy achieved was very optimal. However I noticed that I could improve the algorithm using the techniques mentioned below :

1. The main difference in this improved implementation is the use of extra accuracy measurements during the construction of each tree.

Given a tree in the KNN Forest and its randomly generated data set that was derived from the main train.csv file, I used the rest of the data wasn't chosen for the tree's train_set, to be used to calculate the tree's weight based on its accuracy. I trained the tree with the chosen set but calculated its accuracy on the sub test data described above. Using the accuracy, I then multiplied the value by the Euclidean distance calculated. This helped me prioritize trees with higher accuracy on their 'sub test set' used previously.

For instance, if X is the main data set, $X' \subset X$ is the train data randomly chosen for the current tree, then we can define subset group as $Y' = X \setminus X'$. Therefore Y' is used as a test_set to calculate the weight of the current tree with the given train set, we can mark that weight as $acc(T)$ for the current tree.

Then, for each sample s , we will use $dist = Euclidean(T.centroid, s) \cdot acc(T)$ and take the K closest trees sorted by that measurement.

2. Using the cost sensitive trees and improvements from previous implementations and the parameters.

Using both combinations yielded better results in average, although the difference was quite small as the accuracy is very high as is, but any other methods either resulted in a decrease of accuracy from the original results or no change at all.

- I've also tried ignoring some of the attributes randomly when building a tree to add randomness, but it significantly decreased the performance.
- Another attempt was to try and position the trees from the sample and sort them by their accuracy used in the first improvement I described. The results improved slightly but it wasn't significant and was probably due to randomness.

Overall, any improvement achieved in 5.1 was extremely minor, I used KFold to measure the results that ran over 8 hours and I very seldom hit 1.00 accuracy, using the improvements I mentioned tempered with how often that result was achieved, there was a difference in that manner but I concluded it cannot improve a lot further since there is always a randomness factor to take in consideration.