Decision Trees for Classifying Diabetes

Project Midterm Report

Team Members: Zach Josten

Abstract

The goal of this project is to create a decision tree classifier that can predict the presence or absence of diabetes in people, based off of other health factors. Diabetes affects many people throughout the world and being able to consistently predict its occurrence based off simple factors can be incredibly helpful. This will be achieved through ensemble learning of decision trees, called forests. So far the prediction power of decision trees shows great promise, however remains difficult to read for humans, therefore further refinement is necessary.

1 Introduction

For this project we will be predicting the occurrence of diabetes in people based off some other health factors. This will be achieved via decision tree classification. We will be doing this predicting whilst also comparing it to another baseline method, logistic regression. The dataset that will be used is from the National Institute of Diabetes and Digestive and Kidney Diseases, `https://www.kaggle.com/datasets/nanditapore/healthcare-diabetes`. This dataset contains 8 features, 1 id and then the binary classifier denoting the presence or lack of diabetes.
The focus will be incrementally improving the decision tree classification by tuning parameters. This will mostly be done through the use of bagging in the case of logistic regression and the equivalent forest for tree classification. The expected outcome of this project is to hopefully create a decision tree classifier that can perform better than our baseline in terms of prediction power, whilst also keeping in check our depth and max leaves.

As I continued through the project this mostly stayed as my goal in mind. However, I believe the way at going about it changed slightly. I believe it was more focused on finding hyperparameters that can find the best forest and then therefore create an accurate decision tree classifier, that is most importantly still readable. This piece is in order to ensure that we can understand and comprehend the exact features which are most important and be able to trust that they are actually the most important. This was accomplished first by creating baselines, creating a forest as a further baseline and then continuing further by using sklearns RandomizedSearchCV and GridSearchCV functions in order to fully explore our forests and their best options.

2 Related Work

-`https://www.researchgate.net/publication/350386944_Classification_Based_on_Decision_Tree_Algorithm_for_Machine_Learning`(1)
This one discusses in some great detail the basics on which decision trees are built, the comparison between different methods of going about decision trees and also comparison in terms of classification performance to other classifiers.
It speaks on entropy and information gain being the main driving forces for determining segmentation throughout as well as providing a good metric. It also speaks on the strengths and weaknesses of the algorithm. Strengths include it's comprehendibility to humans, considering that they can be read and we can quickly gain a good idea of relevant features and accuracy. A drawback mentioned is that the

algorithm can make a poor decision and very quickly become derailed and make more incorrect decisions following the first.
Finally it discusses the performance in terms of classification and how it performs compared to others. As the author states decisions trees are exceptionally good at classification and beats out nearly every other main classification algorithm in terms of performance.

-https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4466856/(2)

This one is a bit more surface level in terms of what it covers, however what it covers is extremely useful information. It discusses applications in which decision trees can be applied, such as finding relevance of variables, handling missing values and variable selection. These can be extremely useful and can even be applied to other algorithms after discovering them via decision tree.
It also discusses the basic concepts that go into the decision tree algorithm(s). The ones discussed include nodes, branches, splitting, stopping and pruning. Many of these are used in my project. I believe the only one not really used at all in my project is pruning. Pruning is extremely useful in cases of decision tree classification, however I wanted to not include it considering there's a bit of overlap with it and what I wanted to examine in this project.

3 Decision Tree

In this project we will be using decision tree classification in order to predict on the presence of diabetes using other health factors. The collection of this dataset was done by the National Institute of Diabetes and Digestive and Kidney Diseases and I retrieved it here, https://www.kaggle.com/datasets/nanditapore/healthcare-diabetes.
Pre-processing from the current standpoint is looking to seem quite small, I don't believe much needs to be done to it, considering it's low dimensionality. The most that needs to be done is some removal of incorrect values and their rows. It appears there are some 0's in places there shouldn't be, most likely a replacement for null values, such as in skin thickness, blood pressure and BMI. There is a chance that some scaling may need to be done to certain features due to units and such, however this will not be done presently.
In terms of features, we will simply be using them all(except id, however this is an id, not a feature). There is few enough of them that there is very little cost from looking at them all. When we get to random feature selection in random forest we will however, be using selection rather than all(or maybe not depending on what the best is). But, this is more related to our hyper-parameter selection than anything. Hyper-parameter selection will be quite important in our tree and forest classification, but we won't change them in our baseline logistic regression. For decision tree algorithm, the most important hyper-parameters are those that change/control the size of each of our trees, this is important for keeping trees both readable and also for preventing them from overfitting.
The main evaluation metric is going to be accuracy, alternatively can be viewed as misclassification rate. This is really our only choice for main evaulation metric, due to it being the only one actually able to showcase how "good" our classifier is. However, if we use misclassification rate in place of accuracy(simply by doing 1 - accuracy), we can also take a look at confusion matrices. On top of this main evaluation metric we also have to keep in mind the other parts of the trees. Depth is an important part of decision tree usefulness. As was talked about in the hyper-parameter section, a relatively small depth is important for both comprehension/usability and avoiding overfitting.

As discussed in the introduction I found that the way of accomplishing my goal had found an alternative and better path. Upon getting more into the project as outlined, I realized that I had many

good options for me in terms of hyperparameter searching. Upon getting a few forests put together and working, I decided to search for ways to go about getting good hyperparameters and overall a best forest classifier put together. This is where I discovered sklearns RandomizedSearchCV and GridSearchCV.
These two functions simplified and streamlined the way I could progress forward my hyperparameter tuning. Basically what they do is they take in a list of hyperparameters for whatever classification technique you would like to use. For each parameter you list you can also give a list of inputs you would like to test. The search algorithms will then iterate through combinations of parameters that were given. You may give just one possibility for a certain parameter, like a max depth of 5, or you can give it a large range of possibilities such as all values between 10 and 100 for max depth, either works. This is again something done for each parameter you give as input, leading to large amounts of possibilities that can be tested.
At first I wanted to test the hyperparameters max_features, max_depth, max_leaf_nodes, n_estimators, min_samples_split and min_samples_leaf. I quickly realized that the first three of these were incorrect/erroneous to include. Max features didn't make sense because the algorithm for obvious reasons simply preferred to keep them all in and also didn't make sense for what I wanted, which was to use them all instead of selecting only a few. Max depth and max leaf nodes were poor to use for a similar reason. These two were basically always expanded as far as they could go in order to have the most information available to the model, which makes sense. Additionally it made them unreadable and therefore I decided to clamp them to the same as the forest and reduced tree, which is depth 5 and max leaf node of 20, which as we will see later makes them much more readable.
As for the remaining hyperparameters, we still have a good selection. Here for this pool we had the values of [30, 50, 100, 200, 400, 500, 1000], [2, 5, 10] and [1, 2, 4] for n estimators, min samples split and min samples leaf respectively. RandomizedSearchCV randomly selects combinations from this pool to then test and eventually find the best parameters from what it selected. I chose 15 folds for this algorithm and also GridSearchCV, I believe more folds means better at avoiding overfitting. I then chose to do 20 iterations for each fold, leading to 300 total fits in randomized search.
Things are a tad bit different for GridSearchCV. Instead of randomly selecting from the possible combinations. It iterates over all possible combinations for every fold. Due to this it is best to base the pool for GridSearchCV off the best results from RandomizedSearchCV. Due to this we then have [30, 200, 500, 1000], [2, 5] and [1, 2, 4] for n estimators, min samples split and min samples leaf respectively. This leads to a total of 24 combinations for 15 folds, meaning a total of 360 fits.

4 Preliminary Results

Method	Training Accuracy	Testing Accuracy	Max Iterations	Max Depth	Max Leaf Nodes
Logistic Regression	0.788	0.785	200	NA	NA
Linear Perceptron	0.679	0.64	200	NA	NA
Decision Tree	1	0.995	NA	None(12)	None(96)
Decision Tree(Reduced)	0.935	0.899	NA	7(7)	50(50)

Above we have tried 4 different methods for our preliminary investigation. For some clarification, the numbers in parantheses is the resulting number of either depth or leaf nodes respectively. Also anything not listed within the columns is just the default values as listed by sklearn documentation. The max depth and max leaf nodes for the reduced decision tree were chosen arbitrarily for just a glance at the changes it can have.
We can see that decision tree, even when clamped, performs quite well at a glance. Obviously the full non-clamped decision is most likely too free and this results in what can be assumed to be an unreadable tree. Both baseline methods fall quite far behind the decision tree.
Obviously further investigation is needed and more tuning will be performed with ensemble methods.

However, for now it seems like decision tree performs quite a bit better than our baseline.

5 Experimental results

Method	Training Accuracy	Testing Accuracy
Logistic Regression	0.793	0.774
Linear Perceptron	0.726	0.72
Decision Tree	1.0	0.971
Decision Tree(Reduced)	0.856	0.79
Ensemble Log Reg	0.793	0.774
Ensemble Percept	0.67	0.676
Forest	0.878	0.829
Forest RandCV	0.91	0.86
Forest GridCV	0.898	0.86

There is quite a lot to unpack from these results past their accuracies.To start off we have our baselines of logistic regression, linear perceptron and two very simple decision trees. The first two have 300 max iterations, this is done so that they can properly converge.

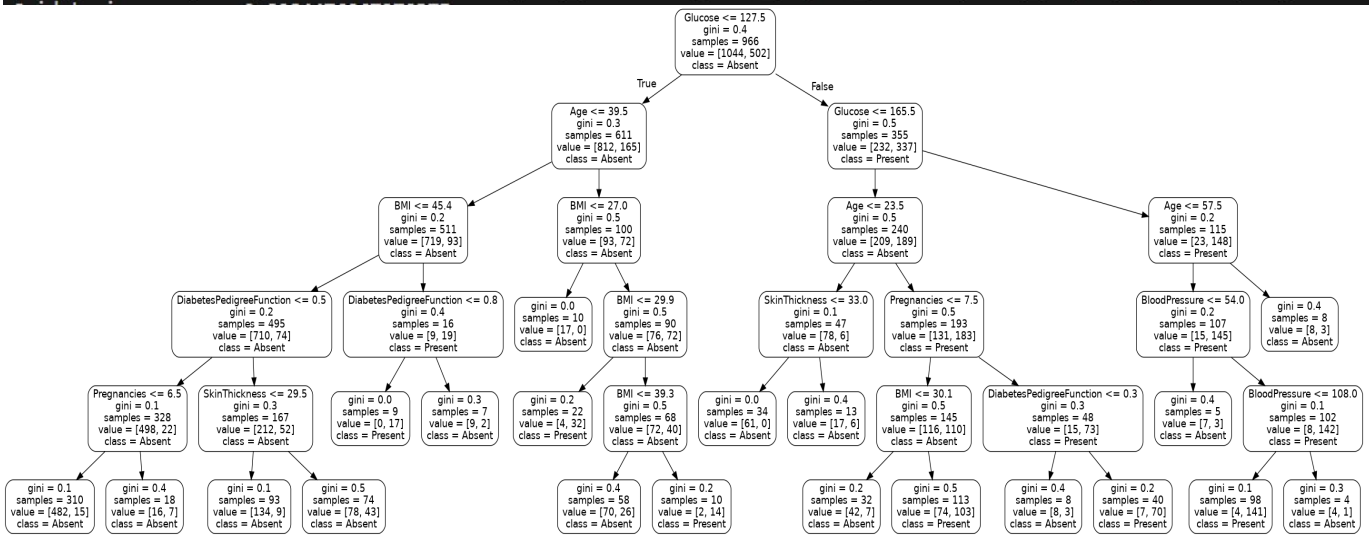
The first decision tree is unclamped, meaning that there is no max depth or max leaf nodes and the tree and continue growing unrestricted until is reaches a point in which it is satisfied. Meanwhile the second one is clamped to be at a maximum of depth 5 and max leaf nodes of 20.

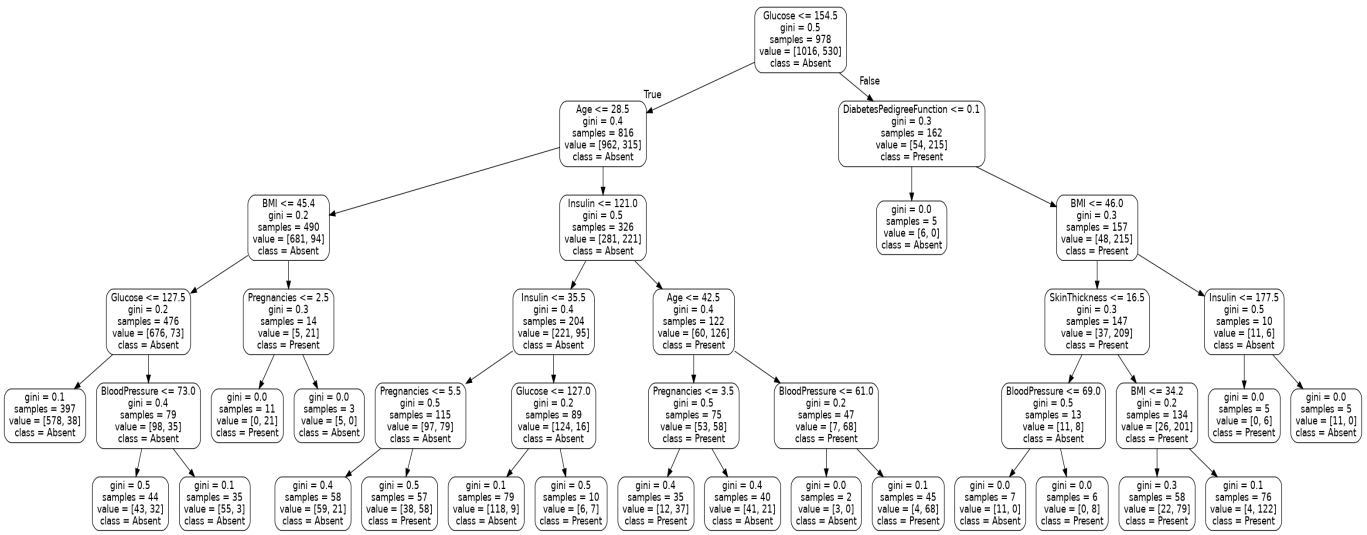
Ensemble logistic regression and perceptron are a bit strange in the results I got from the random state I chose. Logistic regression is the exact same as its non-ensemble version and linear perceptron accuracy dropped from it's non-ensemble. Both of these are a bit strange and can most likely be summed to either error on my part or simply some randomness being involved, especially considering I have seen linear perceptron ensemble actually give expected results.

Forest is quite normal and gives expected results with an increase in accuracy. This is built off of a max depth of 5 and a max leaf nodes of 20 just like the clamped decision tree. Both RandCV and GridCV also share these max depths and max leaf nodes.

RandCV and GridCV both see good improvements over the forest accuracy. They also found the best hyperparameters out of what was given to them, these can be seen below and then we can discuss their decision trees as well.

```
{'n_estimators': 500, 'min_samples_split': 5, 'min_samples_leaf': 4, 'max_leaf_nodes': 20, 'max_features': None, 'max_depth': 5}
{'max_depth': 5, 'max_features': None, 'max_leaf_nodes': 20, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 30}
```





Looking at the hyperparameters chosen, it is incredibly interesting that the two methods chose completely differently where they could, though both still performed well.
Though it is strange that GridSearchCV performed worse considering that it looks through all the possibilities and yet didn't seem to choose the same one RandomizedSearchCV did considering that it performed better. An oddity but I am honestly not sure of its origin.
Theres a lot to be gained from looking at the trees themselves. Looking at both of them there is a common denominator between both root nodes, Glucose. Their split points aren't super far apart either, meaning we can get a pretty good idea of an ideal cutoff and how largely important it is here. We can also see that Age and BMI are also quite important/relevant in these trees. We can also argue that DiabetesPedigreeFunction is also relevant considering it's generally high placement and depth of 2 on GridSearchCVs tree. We then have our 4 most "decisive" features when it comes to determining the outcome of the presence of diabetes.
Something not shown very well here is the necessity of clamping the tree, anything too much larger than this becomes varying levels of unreadable as it continues to grow. Not useless, but certainly time consuming and difficult to work through. As mentioned before the decision tree algorithms really like being able to take as much information as possible as they are incredibly good at making use of the information the can take.

6 Conclusion

Overall I believe this is far from perfect for a multitude of reasons. However I also feel that this is some interesting delving into the entrance of a large topic. I was able to achieve a(two?) largely accurate model that was also able to read and which we could make some conclusions upon.
Again I think it could've been better, mainly in getting some more results overall, especially due to some strangeness in the results. I am not 100% sure on where some of the oddities that I went over here originate from. If more time allowed I would've also liked to perhaps get an average over the data that was obtained to see if randomness was part of the odddity. However, considering a single run of the code could take anywhere between 15-25 minutes, getting an actually good average would take a long time.
But again I am very happy and satisfied with this.

7 References

-Jijo, Bahzad and Mohsin Abdulazeez, Adnan. (2021). Classification Based on Decision Tree Algorithm for Machine Learning. Journal of Applied Science and Technology Trends. https://www.researchgate.net/publication/350386944_Classification_Based_on_Decision_Tree_Algorithm_for_Machine_Learning 2. 20-28.

-Song YY, Lu Y. Decision tree methods: applications for classification and prediction. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4466856/ Shanghai Arch Psychiatry. 2015 Apr 25;27(2):130-5. doi: 10.11919/j.issn.1002-0829.215044. PMID: 26120265; PMCID: PMC4466856.

-Sklearn documentation. https://scikit-learn.org/stable/modules/classes.html