**Data Mining Portfolio Entry for Classification (Frank Liu)**

**Data acquisition**

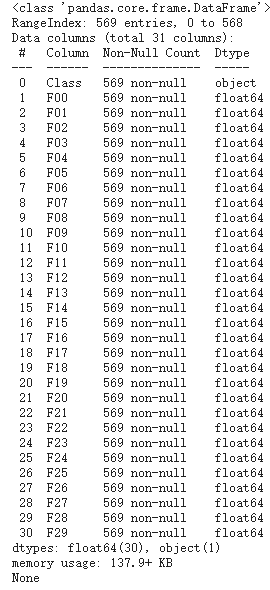
* **Cancer Data**

The data that will be used for this DMP is the cancer data provided in the assignment. I personally believe it is a very interesting dataset, and very clean and diverse, since it contains one categorical variable indicating benign or malicious cancer/tumor, and all other are numerical data that can be analyzed to classify the tumor as B or M. There are 569 samples and 31 features after cleaning (remove the title, space, and parse with ‘,’). The data matrix has a dimension of (569, 31).

Here is a table of the features.

| **Name** | **Type** | **Comments** |
| --- | --- | --- |
| Diagnosis | Categorical | Very important since that is the class label |
| radius \_mean | Numerical | Useful |
| texture\_mean | Numerical | Useful |
| … 29 more | Numerical | Useful |

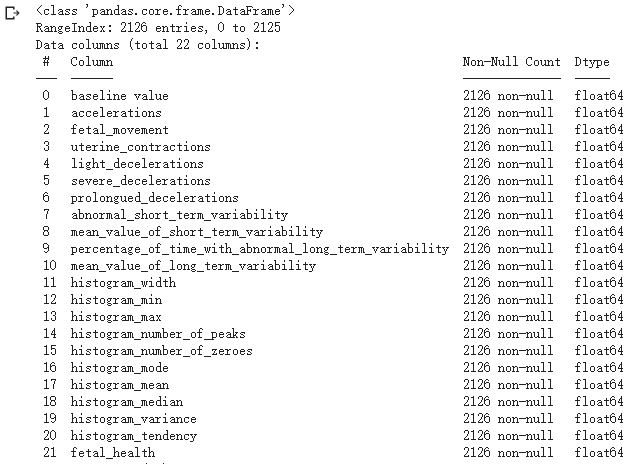
In the second program where we can use packages, I import the data set as a Pandas data frame, and print out the info of the data:



The diagnosis is the final class label we would like to classify. Depending on all other numeric attributes, all of which are some medical or biological measurements, we can build a decision tree classifier to draw association among the specific class and those numerical values.

* **Fetal Health Data**

I also use my own data, which is a similar dataset comparing to cancer data with more training samples, but using 21 numerical values to classify the health of a fetus as normal, suspect, or pathological using CTG data.



This is data frame information from Pandas, and the last feature is the class label, taken value from set{1,2,3}. The shape of the data is (2126, 22), indicating a 2126 number of samples and 22 features. I will use 21 to classify the data.

This entry will use this data set in four tasks, and cancer data will be used in a non-package program, with fetal health data used in a package program. For Bayes Classifier, I will also use the three datasets provided by the instructor.

1. Decision Tree Assignment 1
2. Decision Tree Package Implementation and Hyperparameter Tuning
3. Bayes Classifier Assignment 1
4. K-nearest neighbor Package Implementation

**Program development(without packages)**

I will use Python in Google Colab as my language and IDE. There will be two programs that I will develop in this section

* Decision Tree Assignment 1
* Bayes Classifier Assignment 1

All of these two programs will use cancer data to test the performance. I will use three Bayes data to test the performance of the Bayes classifier as well.

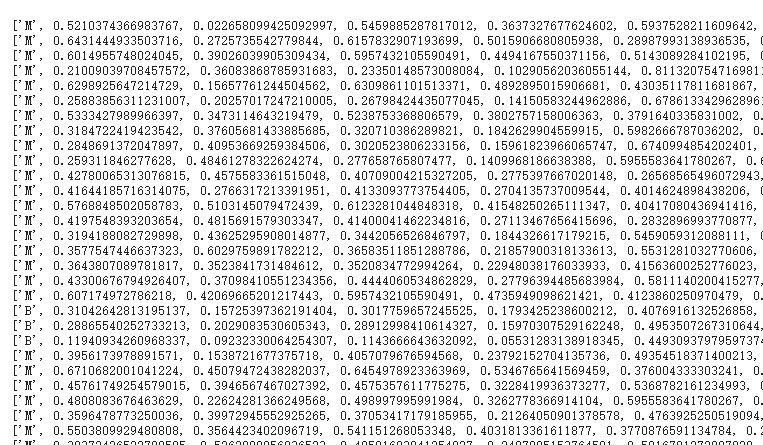
For assignment 1\_Decision\_Tree:

Since this assignment requires not to use any packages, the advantages of slicing a 2D array in NumPy can not be taken advantage of. In order to analyze the column of the data matrix (each feature is a column), I will extract them separately out into a vector using for loop, do operations on them (calculating Gini or entropy or calculate the mean), and then copy them back to the original data matrix.

For the root node, I calculate the number of labels in different classes, Gini index, entropy, and mean for each feature, and then we do the same operations for all child nodes if we split the node based on the mean of each feature. Specific details are in the prompt of the file.

Detailed code can be found here: <https://colab.research.google.com/drive/1IbW6Pk3X_6uCG90L7lLdq3IQeLIuxgFM#scrollTo=clcN-h1ggSn_>

Here is a screenshot of the part of the data matrix from the Colab. I write a normalization function so that each feature falls into the range of [0,1].



There is more data, but this is just a part of it. And the statistics for the root node is:

Class B observations: 357

Class M observations: 212

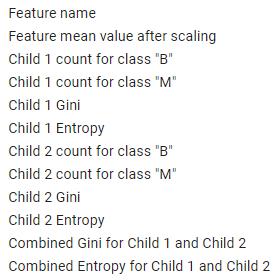
Total observations: 569

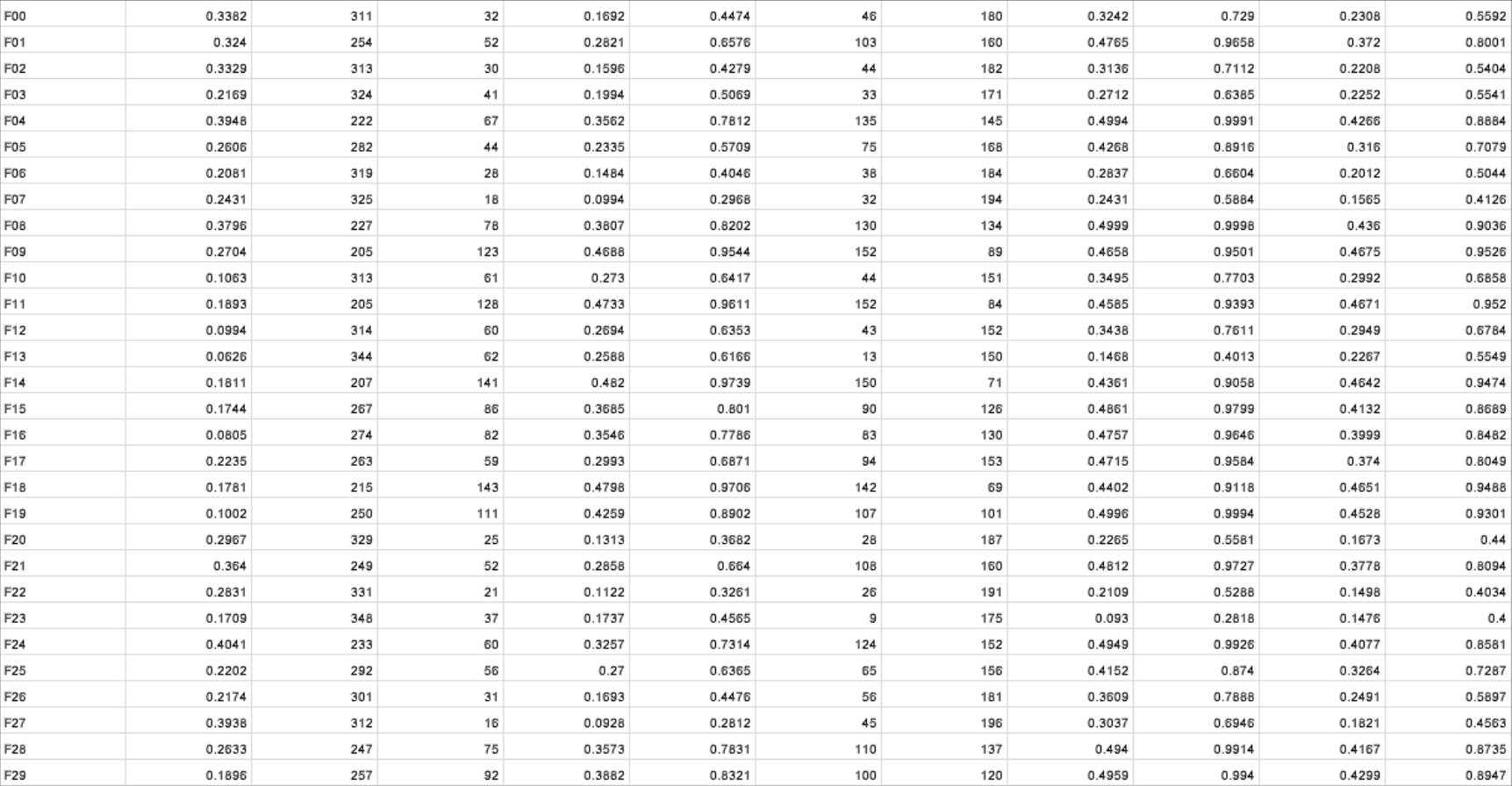
Root gini: 0.4675300607546925

Root entropy: 0.9526351224018599

After splitting the root node based on the mean of each feature and writing the data into the output file, I attach the data here.

The column corresponding to





The output file can be found in this classification folder, and my result is the same as the output given by Professor Santago.

We know that the Gini index is a measure of impurity, and the smaller the value, the purer the class, the better the classification is. Therefore, when looking at the Combined Gini (2nd column from the right), feature 23 has the smallest Gini Index (0.1476), and when you look at the combined entropy, it also has the smallest entropy value (0.4) among all features. Therefore, I can safely conclude that feature 23 can be used to best split the root, and both Gini index and Entropy can prove this.

I will also discover the **optimal split** of this program:

The algorithm I use is that for each feature, I loop through all elements of the feature as a possible splitting point, split the data and count the number of different classes for each child, and then compute the combined gini and combined entropy. The splitting point will be the one that has the smallest combined gini. Here is my output for the first 6 features:

Surprisingly, **my output, using the splitting point I found, is actually having a smaller gini than the output given by Professor Santago**. I checked the way I did it and I think my way of doing it is correct, so maybe the professor’s output is wrong. I would like to have a discussion with the professor, and there are also likely that my output is wrong.

-------------------------------------------------------------------------------------------------------------------------------

for feature F00, combined\_gini is 0.19242486589915644, combined\_entropy is 0.4896488694028093, with 514's feature as a splitting point with a value of 0.3818921860949407

for feature F01, combined\_gini is 0.3690697283638583, combined\_entropy is 0.7961722035638179, with 131's feature as a splitting point with a value of 0.3165370307744335

for feature F02, combined\_gini is 0.1916093835208614, combined\_entropy is 0.4860075931192521, with 205's feature as a splitting point with a value of 0.3694976159214982

for feature F03, combined\_gini is 0.1867183670721237, combined\_entropy is 0.4778349162125473, with 38's feature as a splitting point with a value of 0.23686108165429479

for feature F04, combined\_gini is 0.41023756912945447, combined\_entropy is 0.8555307306582929, with 444's feature as a splitting point with a value of 0.3567753001715266

for feature F05, combined\_gini is 0.30446499363426294, combined\_entropy is 0.6855696883147584, with 23's feature as a splitting point with a value of 0.20425127292804127

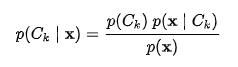
-------------------------------------------------------------------------------------------------------------------------------

For assignment 1\_Bayes Classification:

Bayes Classifier is a probabilistic model that makes the most probable prediction for a new example. The idea is coming from Bayes’ theorem with strong independence assumption between the features.

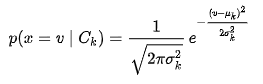
The code is attached here: <https://colab.research.google.com/drive/1sjr6f3Je-6ffApdzZbh0McG0tkOPFjgq#scrollTo=S-x429DBYvYU>

The Bayes Theorem stated that:

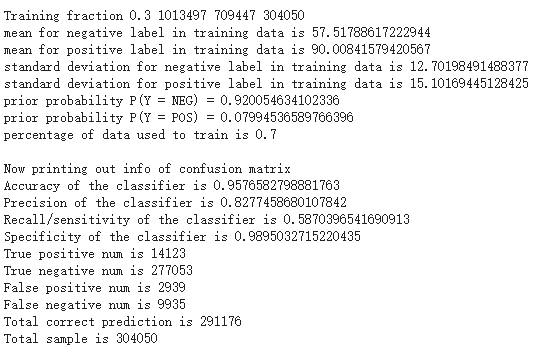


We train the model once we have the mean, standard deviation, and prior probability of different labels. We compute the inequality:

If P(X | C1)\*P(C1) >= P(X| C2)\*P(C2), we classified X into C1, otherwise C2. Since we are dealing with continuous data, we use the Gaussian Naive Bayes to approximate P(X|C) by using the formula:



I implement the classifier using only python math operations, and the performance of the classifier is actually pretty good. I attached the output console including a confusion matrix data:



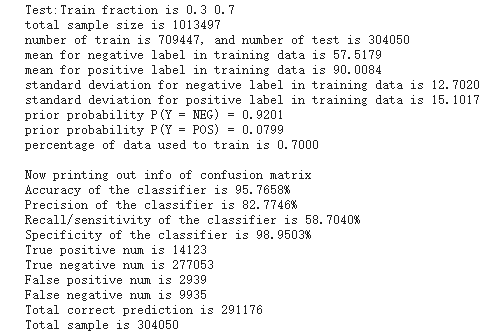
As we can see, the model is able to achieve accuracy greater than 95% on the test set, which is a pretty good performance given such a simple structure of the classifier.

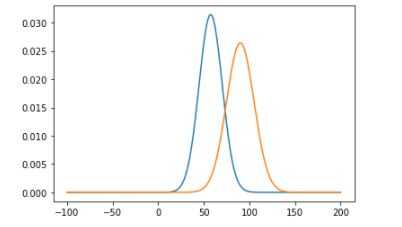
**Analysis of the following three data**

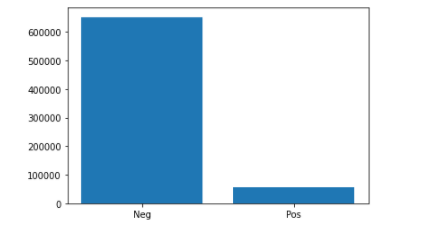
Bayes 03 contains more sample data than the other two. The performance of the classifier on data 1 is the best, and then the 03, the worst performance was on data 02.

I will make the Gaussian plot of the two mean and variance that I calculated and the histogram of two different classes as well.

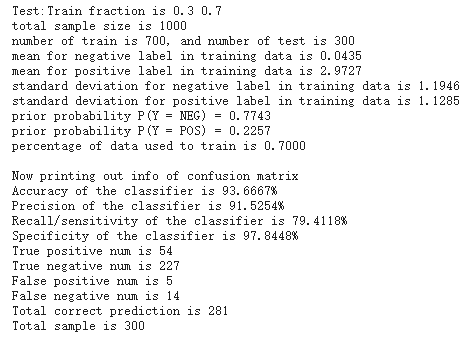
**The performance data for Bayes 03.csv**

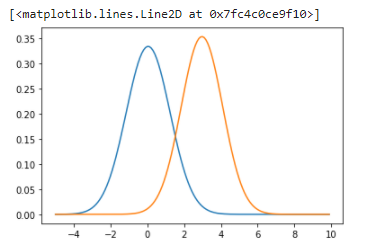


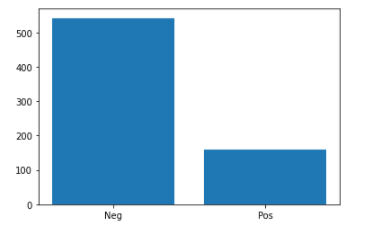




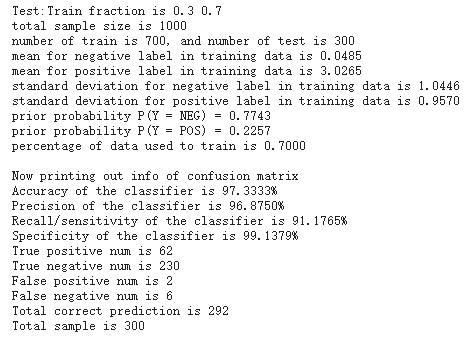
**The performance data for Bayes 02.csv**

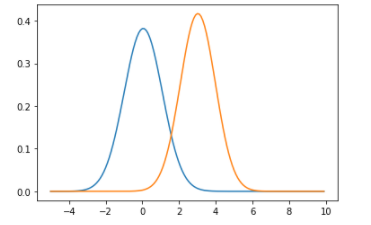


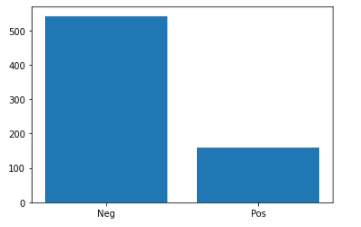




**The performance data for Bayes 01.csv**







**Data analysis and package use**

For Decision Tree Packages:

I will use scikit -learn, Numpy, and Pandas to build and visualize the decision tree. After building the decision tree, I will evaluate the hyperparameters and do some hyperparameter tuning to prevent overfitting, observe how decisions are made, and record some criteria for stop, divide and determine the division techniques.

Detailed code can be found here:

<https://colab.research.google.com/drive/1wjBOKLG3VKkZgr6brpNqVRYYnV4VQrhB>

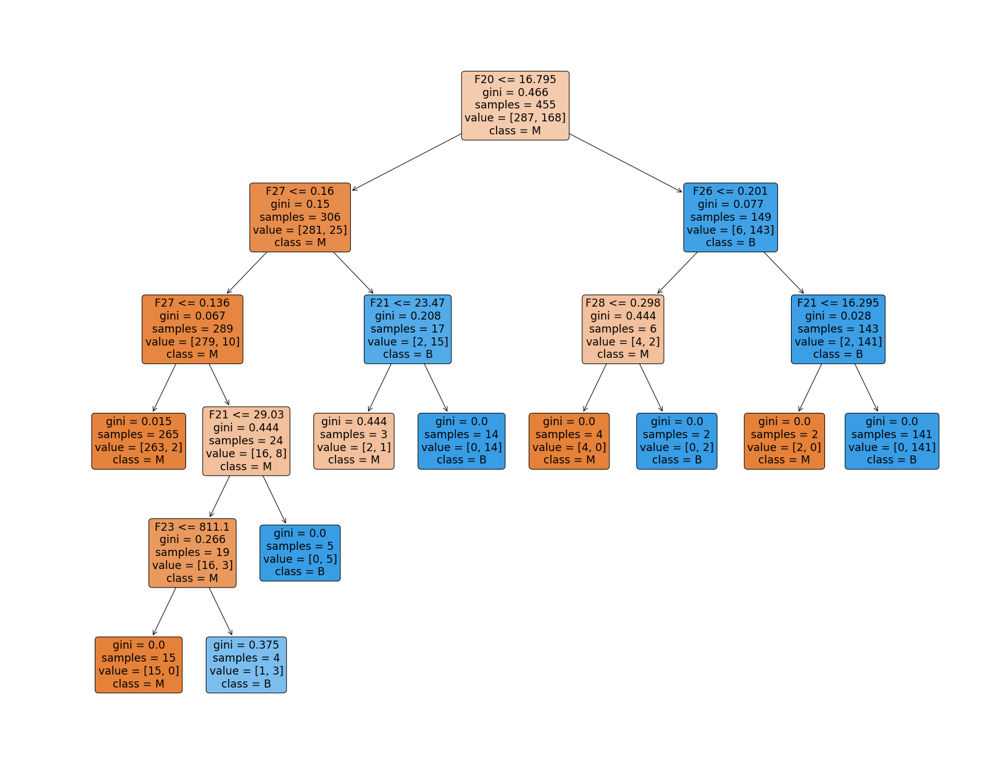
I first use the cancer data to do a hyperparameter tuning and then switch to the fetal health data to do more analysis.

I primarily will tune 5 different hyperparameters and show my result of tuning in the form of a table. (Note, I print out the confusion matrix to evaluate the performance of the tree, but only use accuracy as a criterion to do hyperparameter tuning)

| criterion | max\_depth | min\_split | min\_impurity\_decrease | Test sample ratio | Testing accuracy | Precision | Recall | Specificity |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| gini | 30 | 5 | 0.001 | 0.2 | 92.11% | 91.67% | 89.90% | 93.85% |
| entropy | 30 | 5 | 0.001 | 0.2 | 94.75% | 93.74% | 93.88% | 95.38% |
| gini | 60 | 5 | 0.001 | 0.2 | 92.98% | 91.84% | 91.84% | 93.85% |
| gini | 30 | 2 | 0.001 | 0.2 | 92.11% | 91.67% | 89.80% | 93.85% |
| gini | 30 | 5 | 0 | 0.2 | 90.35% | 91.30% | 85.71% | 93.85% |
| gini | 30 | 5 | 0.001 | 0.1 | 96.49% | **100%** | 90.48% | **100%** |

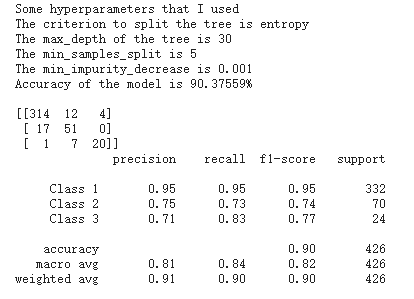
I first used standard hyperparameters and achieved a result of 92%. Then I change the criterion from gini index entropy and see a significant improvement in accuracy, to about 96%, so do all other metrics. For comparison perspective (control variable), I switch back to gini and change the max depth to 60 and the result won’t vary a lot. I think expanding max depth should result in overfitting, but here there is not a clear sign of overfitting. I think the reason is that the model won’t achieve that deep, even we allow a deeper tree. I decrease the min\_split to 2, which is the minimum number of samples required to split an internal node. The accuracy and all other metrics won’t be varied by a lot. If I change the min\_impurity\_decrease to 0, the accuracy drops to about 90%, so do other metrics., and I believe there is a little bit of overfitting here because this parameter is the threshold for early stopping in tree growth. The tree is going to continue to grow, resulting in an overly complicated model. The accuracy is higher (96%), and precision and specificity reach **100%** when I change the test ratio to 1, meaning more samples are used to train the tree so it will have a better performance. The more sample to train the tree, the better ability for the tree to generalize.

To further see the tree, I use a visualization technique to plot the tree out.



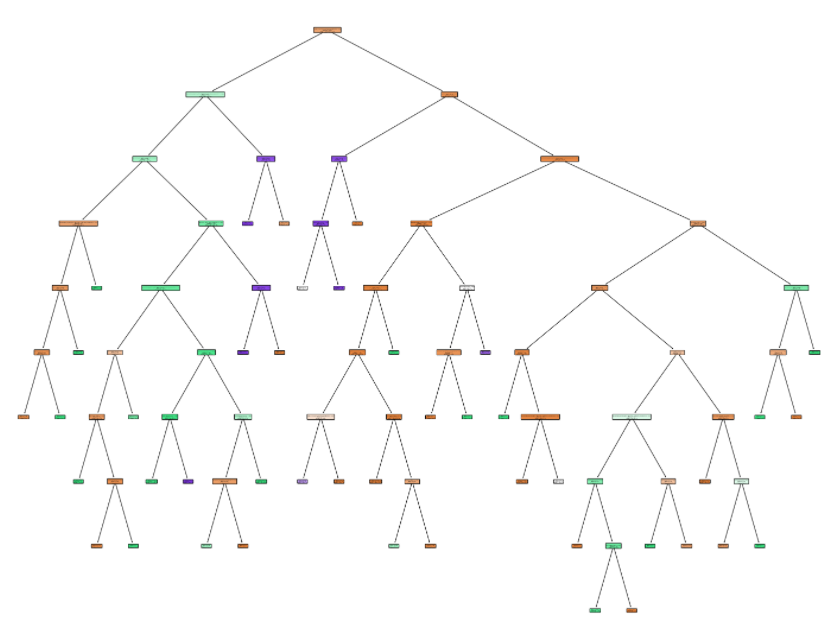
We can see the tree achieved pretty good performance with only a depth of 6. This is a good model that doesn’t overfit the data. How about the performance of the decision tree on fetal data, let’s have a try.

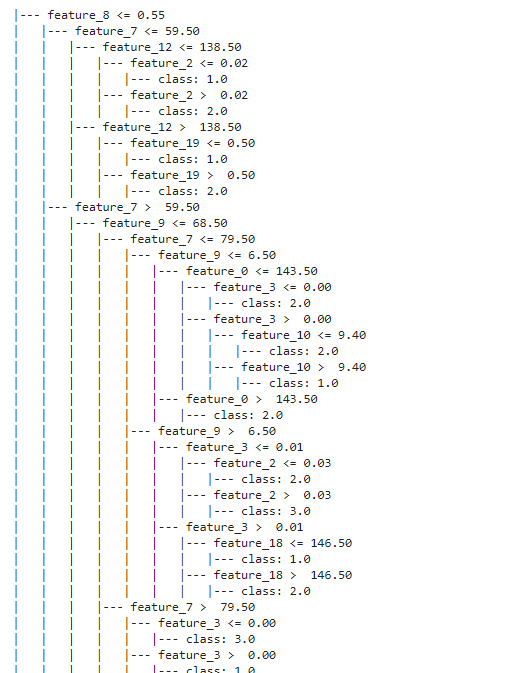
I imported the fetal health data, and since the format of this dataset is a little bit different from the cancer data, I modified the code a little bit.



We also achieved a pretty good classification result.

I attached here a visualization of the tree and a text version of the tree. The most important feature is feature 8 since according to the textual version, it was the feature used to split as root, which means it is an important deciding factor. We can see the precision is lower, so does the recall. This means the true-positive composition is relatively lower. Therefore, to improve it, I can refer to the original table, changing the criterion to entropy, The accuracy increases to 93%, so do other metrics.





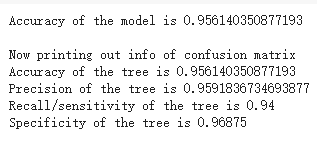
This time, the tree is more complicated since the nature of our data is complicated

For K-Nearest-Neighbors Packages:

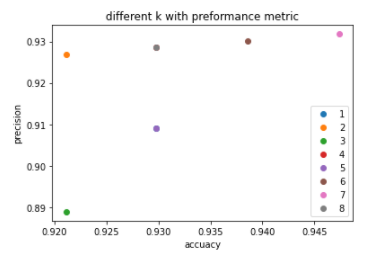
The code using scikit-learn is here

<https://colab.research.google.com/drive/1ZwNdhcy14ceiYOnuONKNxFd2fh21wDrF>

It is using the same idea of using the scikit-learn package to build KNN models, and after doing so for the decision tree, this is much easier because it generally follows the same pattern. For the cancer data, the output of the console is



I also discover how **various k** would influence the accuracy and precision of the model



We can see performance is the greatest when k = 7. So hyperparameter tuning is important in making a good model.

During class, I think the idea of KNN is so simple and straightforward that it couldn’t even be called a classifier. But it turns out that it has a very good performance in classifying datasets. I further discover the power of the classifier by having it classify the fetal health data, which is a little bit more complicated.

The accuracy of the model is 0.8849765258215962, which is not as good as the model above, primarily due to the complexity of the dataset, but it is still a very good classifier from a general perspective.

**Theory (Some ideas below, including text exercises. Not all, would have to be done. In future chapters, there are some good theory-type exercises at the end of the chapter.)**

[Liu\_Classification\_Dataset](https://docs.google.com/document/d/14xXY_BXsEVuQJ_3yID24DDn-VA3SQxDnq9EGOJWqe8s/edit)

**Chapter/section summary or major learning points**

The attached link is the reading notes I take that record some important information of the concept in chapter 3 and chapter 4.

* [Chapter 3 Reading and Lecture Notes](https://docs.google.com/document/d/1QA72Sma_OHoJvy_KuSf4gMJ4B-2PcCvGVEZrkmpIXhU/edit)
* [Chapter 4 Chapter Reading and Lecture Notes](https://docs.google.com/document/d/1Onvw57RWaFJFnawCRGa89swJca1bVAseDUmfX0pV5ds/edit)

. This is still a bit wordy, so I decide to include some major learning points that I have

* General Ideas of what is a classification and type and tasks of classifiers
* How to build a classification model
* Decision Tree and Algorithms
* Characteristics of Decision Tree
* How to split a node and some indexes (gini, entropy)
* Overfitting and model evaluation
* K-nearest neighbor classifier and algorithms
* Naive Bayes classifier and algorithms
* How to use packages to implement classifiers.

**Student learning summary and self-assessment**

After finishing this entry, I have a pretty good idea of what a classifier should look like. I practice my skills through implementing from scratch and also through learning how to use packages. In the past, I only knew how neural networks worked, and I used it in my Reinforcement Learning research. I was amazed by the idea of neural networks, and think this should be the best classifier. However, after these two chapters, I know more about other classifiers. The ideas behind those (decision tree, KNN, Bayes) are pretty simple, but it turns out that the power of those classifiers is huge. It can complete very complicated tasks that I cannot imagine such a simple structure is capable of. I feel like after doing all the programs, I have a strong sense of achievement to see the accuracy is really high.

So far I have no questions regarding this chapter because I think the textbook is pretty self-explanatory, also the PPT. I want to explore a lower-level implementation of Scikit-learn packages in the future rather than simply calling a constructor and magically building a classifier for you. As a computer science student, it is important to understand how things are built.

After reading chapters and lecture notes, finishing the DMP, and doing lots of programming exercises, I think I have an A comfort level in classification.