CSC 503 Assignment 1

Haijia Zhu, V00835420
hjzhu@uvic.ca

1. Classification on heart disease dataset

In this study, three different classifiers are used to identify heart disease. All three classifiers perform acceptable results using scikit-learn [1].

1.1. Pre-processing data

All the heart disease data is loaded using pandas and is split into two groups (ratio of 20/80) to perform cross-validation.

1.2. Decision Tree

Scikit-learn provides several pre-pruning and post-pruning method. First, a benchmark is performed. Gini index is used to evaluate the quality of split, as it is theoretically not as aggressive as entropy (information gain). No pruning is performed on this benchmark; thus, a deep and broad decision tree is expected as shown in Figure 1 and achieves an accuracy of 80% in the test.

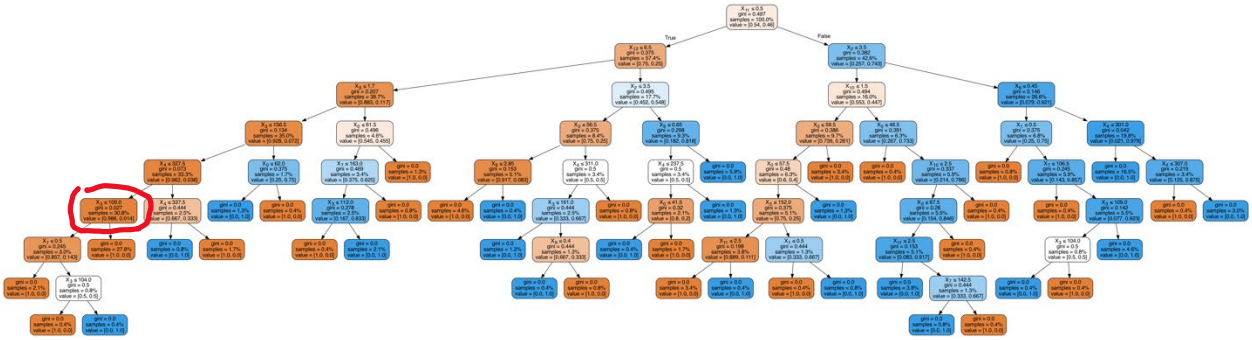

Figure 1 Decision tree benchmark

Next, we perform pruning on the decision tree. By check the decision tree in FIGURE 1, we find that most of the samples have been successfully classified. For example, the node in the red circle in Figure 1 contains 30.8% of the sample and among those, only 0.4% (which is one sample) turns out to be negative. One intuitive way to perform pruning is to reduce the depth of the tree. Figure 2 shows a pre-pruning with a maximum depth of 3. This method is straightforward and effective; however, it is not ideal when the training sample is large and something hard to find the optimal depth. Other stop criteria can be taken into account such as minimum number of samples required to split, minimum number of samples required to be at a leaf, impurity reduction threshold for each split, etc. And by considering the multi-argument, we add more degree of free into the decision tree design and significantly increase the complexity of the problem.

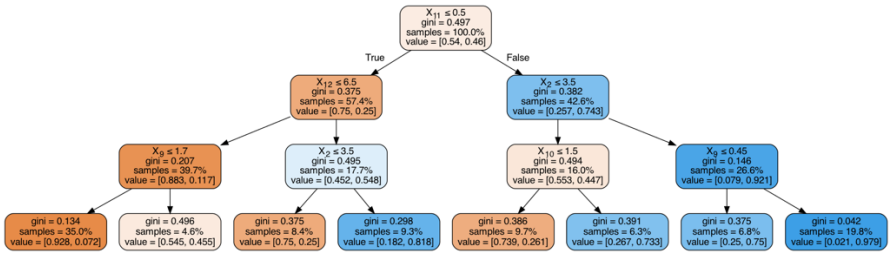

Figure 2 Decision Tree with a depth of 3

Another common post pruning method is called minimal cost-complexity pruning. By increasing the alpha, the number of nodes and the depth of the tree reduced significantly.

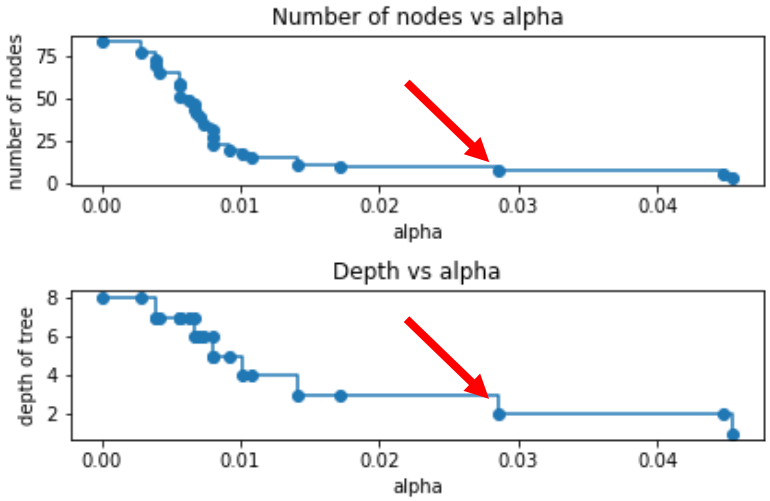

Figure 3 Tree node and depth VS alpha

To find an "optimal" decision tree, we perform an exhaustive search on the possible pruning path during minimal cost-complexity pruning and the result is presented in Figure 3. We can see that as we increase alpha, the accuracy in the train set reduced as expected and the accuracy in the test set increase as pruning can curb overfit. However, a large alpha can lead to worse results as too many nodes are pruned, and the classifier doesn't have enough depth (and node) to identify the belongs of the input data and as shown in the red arrow in Figure 2 and Figure 3. Eventually, a balance needs to be made according to the testing and training results. One possible "optimal" alpha can be found and is pointed by a black arrow in Figure 4. This number provides relatively high accuracy in both training set and test set and achieves similar accuracy. The drop on both training and test set for the alpha indicates that the tree is over-simplified and starts to reduce accuracy (overfit is no longer the major issue under this circumstance). The post-pruning decision is presented in Figure 5.

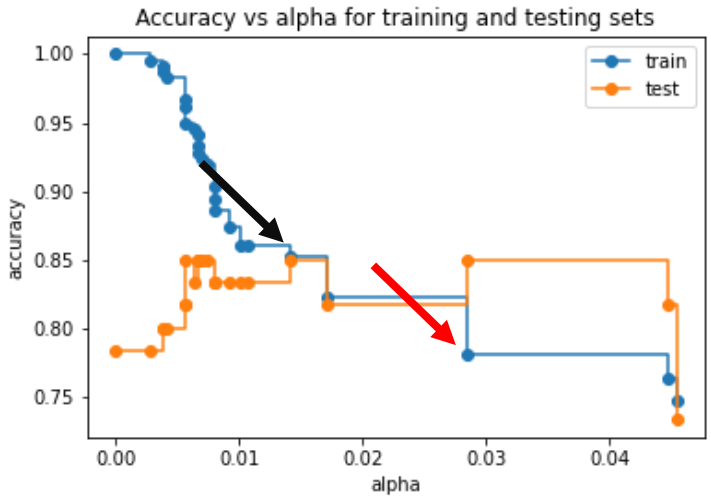

Figure 4 Alpha VS Accuracy

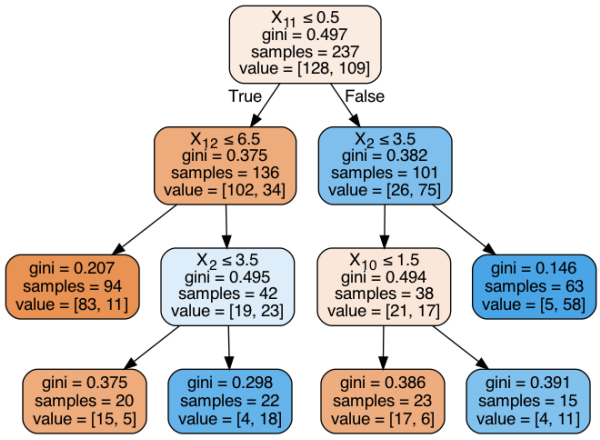

Figure 5 Post-pruning decision tree

An interesting thing that confuses me is that even though no randomization is involved during the training (train the model using the same train set and same fixed parameter, no random split, etc.), the trained model achieves different accuracy, which might be the fact that the datasets are too small and need further study.

1.3. Random Forest

For random forest, the assignment specifically asks us not to perform any pruning. Thus, only a few parameters can be adjusted. They are the number of trees in the forest, the split criterion, and the number of features to consider when performing any split.

According to the original random forest website [2], random forests do not overfit. And we can run as many trees as you want. Thus, the number of trees is set to 1000.

The entropy is chosen as the split criterion after applying several rounds of trial and error as it is more robust for this dataset. The number of features *n'* in the subtree will affect the accuracy of the prediction, and it is recommended to use the square root of the features. And after performing an exhaustion search, we noticed that the best performance can be achieved when the *max_feature* is set as 3 or 4, which is the square root of the features in this study.

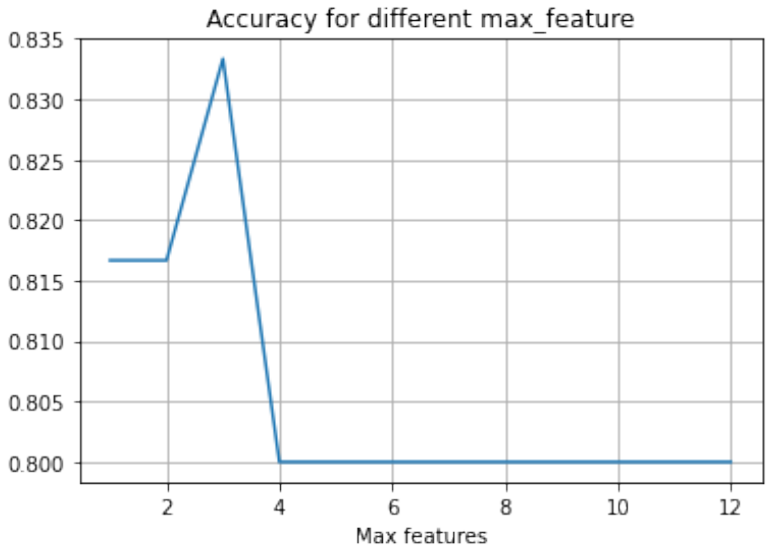

Figure 6 Accuracy when applying different maximum features for the random forest.

Finally, this algorithm achieves 83% accuracy in the test datasets. Although a signal tree in the forest, as shown in Figure 7, still suffers from overfitting, the forest successfully curbs the overfitting and performs better than the decision tree (with post/pre pruning).

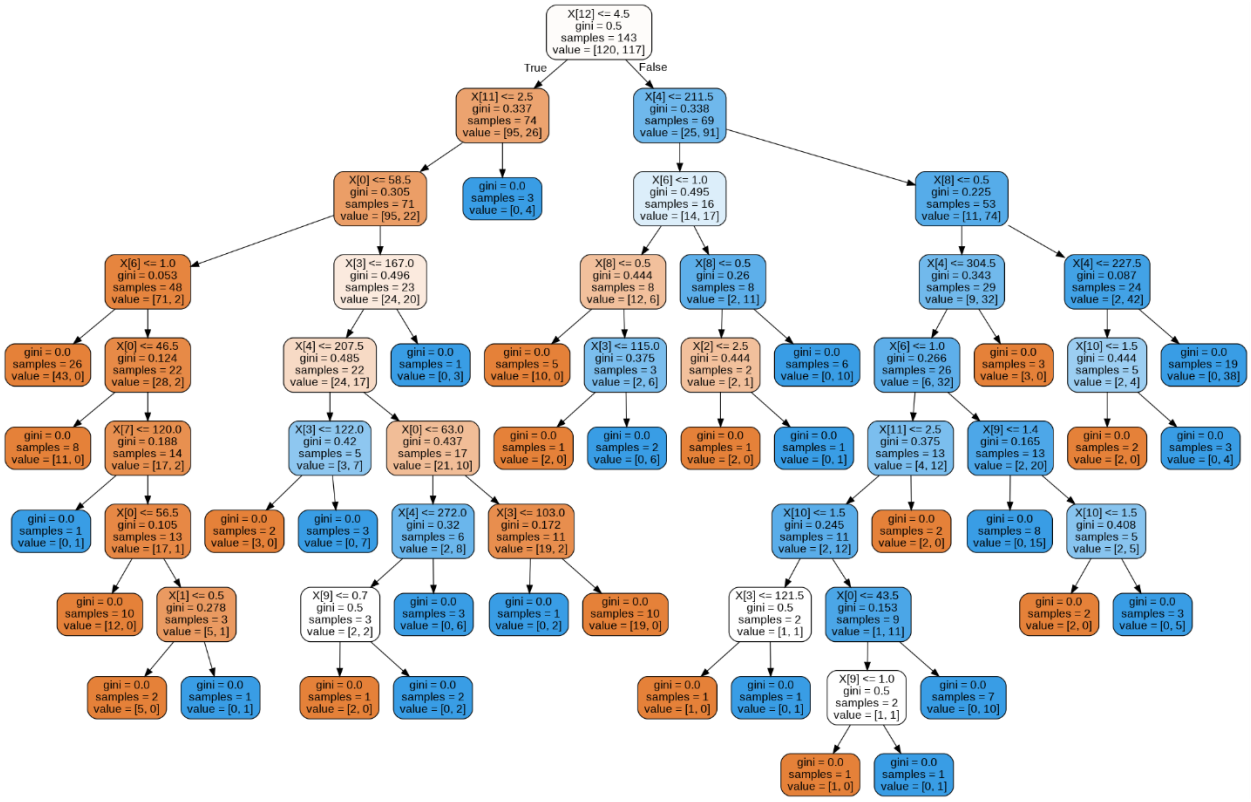

Figure 7 A tree in the forest

1.4. Multilayer perceptron

MLP is sensitive to feature scaling. First, the original processed data need to be standardized. This will give us the data as shown in

Scikit-learn provides 3 different solvers, and LBFGS is selected as it tends to perform better and fast than SDG in small datasets.

Add more layers doesn't significantly improve the performance of the ANN. In this experiment, a 4 hidden layer with a node number of 50 at each layer MLP is tested and only achieve an accuracy of 53.3% while a 3-layer MLP can easily achieve an 80% accuracy. And it has been approved that an MLP with one hidden layer is capable of approximate any function that we require [3].Thus, this study will mainly focus on 3-layer MLP.

The number of nodes in the hidden layer can affect the performance of MLP, however, there is no mathematical way (find a close form result) of finding the optimal node number, thus the number of nodes needs to be studied.

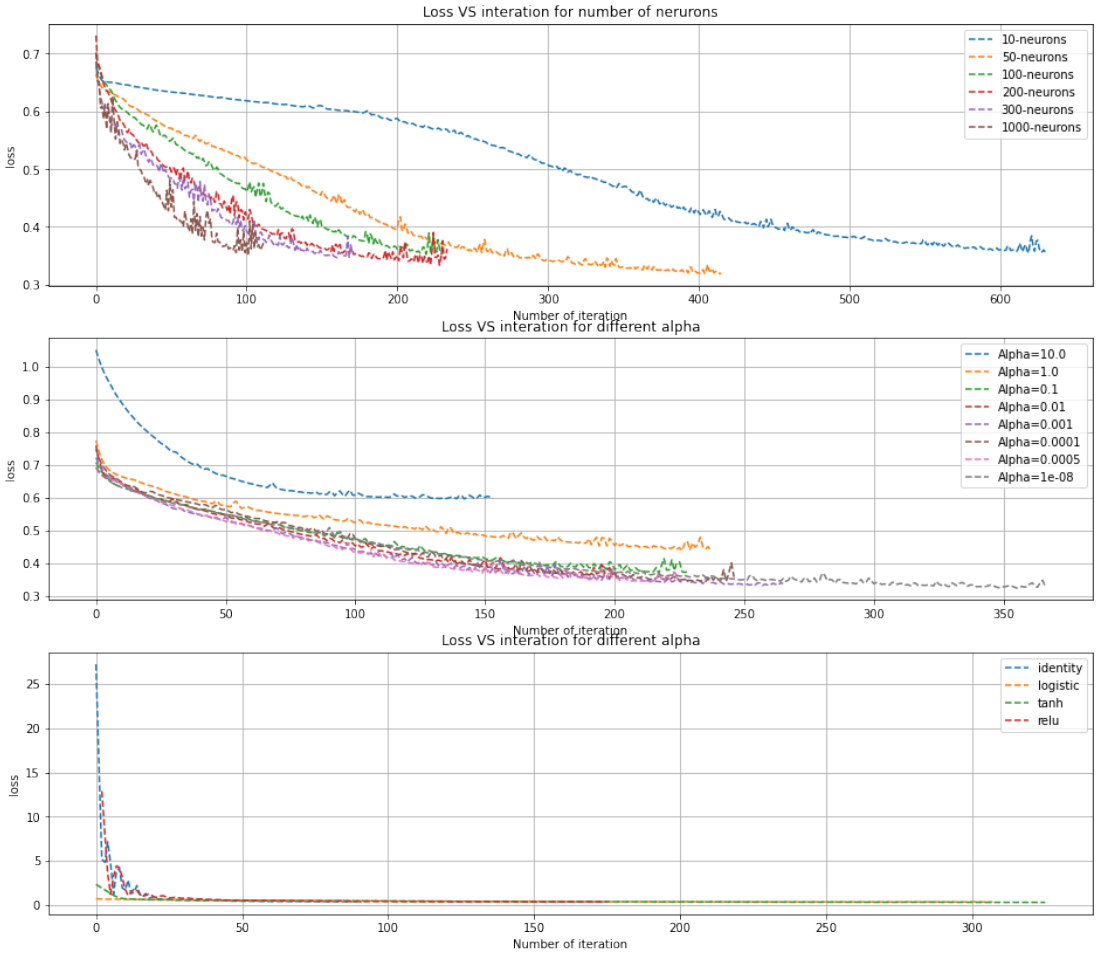

Figure 8 Loss VS number of iterations in different configurations

Other main factors that affect the performance of the MLP are the number of hidden layers, number of nodes in the hidden layer, L2 penalty, and the type of the activation function.

Figure 1 shows the number of iterations during the training. The figure at the top shows the loss curve during the training. And the smaller number of nodes leads to a slower training process and it needs more iteration to convergence. However, a larger neural network also is more computationally expensive. Another problem that comes with the larger network is that overfitting, as shown in Table 4. The largest MLP performance the worst when validating the test samples with an accuracy of 78.33%.

Table 1 Accuracy for different number of nodes

Number of nodes	Accuracy
10	0.8333
50	0.8
100	0.8
200	0.8166
300	0.8166
1000	0.7833

MLPClassfier provides a build-in L2 regulation function (sum of the squared values of the weights), this parameter mainly aims at penalizing larger weights during the train, curbing overfitting.
The figure in the middle shows the number of iteration and their loss when applying different L2 penalty parameters. We can see that a propel alpha can significantly reduce the train time and provide a better loss.

Table 2 Accuracy under different alpha

Alpha	Accuracy
10	0.6666
1	0.8333
0.1	0.7666
0.01	0.8166
0.001	0.8
0.0005	0.8166
0.0001	0.8
0.00000001	0.7833

Table 3 Test data set accuracy using the different activation functions.

Activation function	Accuracy
identity	0.75
logistic	0.8
tanh	0.8
relu	0.8

It should be notice that all the other parameters that are not presented in Table 2, Table 3, and Table 4 are set to the same (100 nodes, alpha = 1e-4, logistic, learning_rate=adaptive, LBFGS). One common way to find the L2 regulation parameter and node number in an ANN is to use an automated search that tests different network configurations. Scikit-learn has a build-in grid search algorithm and is used to find an optimal node in this study. The parameter that needs to be searched is presented in Table 4. And the accuracy of the trained neural network is 0.8333 with an alpha of 0.0001, and a hidden layer node of 10.

Table 4 Grid Search Parameters

Parameter	Hidden layer size	L2 penalty
Value	[5, 10, 15, 20….10]	[1, 0.1, 0.01, 0.001]

1.5. Summary

All three methods can successfully identify the input data with acceptable accuracy. The training time is not compared on this dataset as the dimension of the dataset is small and does not significantly affect the train time. The training speed will be compared in the customs classification problem.

2. Custom classification problem

In this custom classification problem, I am going to classify an ovarian cancer dataset [4]. The reason for selecting this dataset is that each sample consists of 4000 features and it should be a good chance to test the capability of decision, random forest, and MLP.

2.1. Pre-processing data

The original data contains 4000 columns of features and one column of labels, a total of 216 samples were collected. The labels are marked “normal” for healthy patients, and ‘cancer’ for illness.

First, assign all string label, ‘cancer’, to 1 and normal to 0 in Microsoft Excel as classifiers can’t process string labels directly. Next, split the sample to train and test datasets with a ratio of 80/20.

2.2. Decision Tree

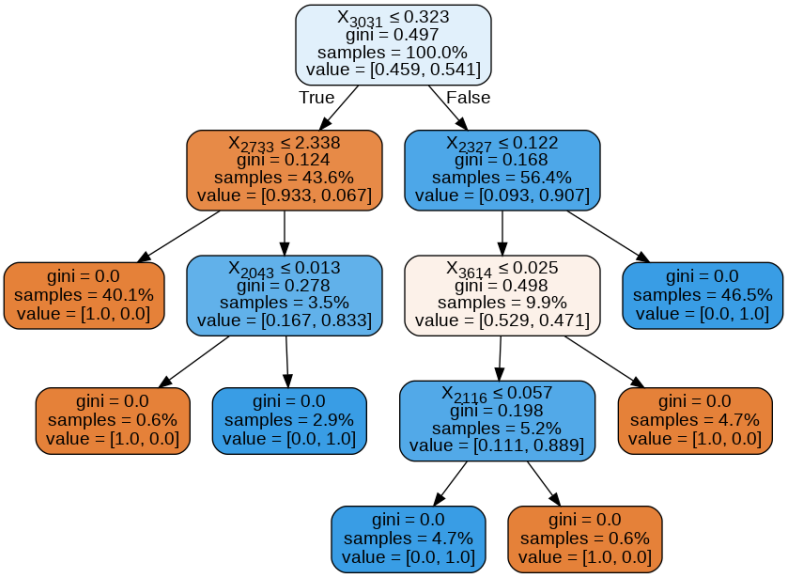

Figure 9 Decision tree

First, the decision tree without any pruning is presented in Figure 9. The decision tree can achieve an acceptable accuracy in this study and is robust. However, the dataset contains plenty of redundant data, and feature ***obs 3031, obs 2327, obs2733*** are the most important feature amount all. This also can be found in Figure 12. It also indicates that the original data has limited room for pruning, as the original tree is not as broad and depth as the tree presented in previous datasets, 0. This decision tree achieves an accuracy of 84% in the test dataset.

After applying a post-pruning, the accuracy increases, as shown in Figure 10.

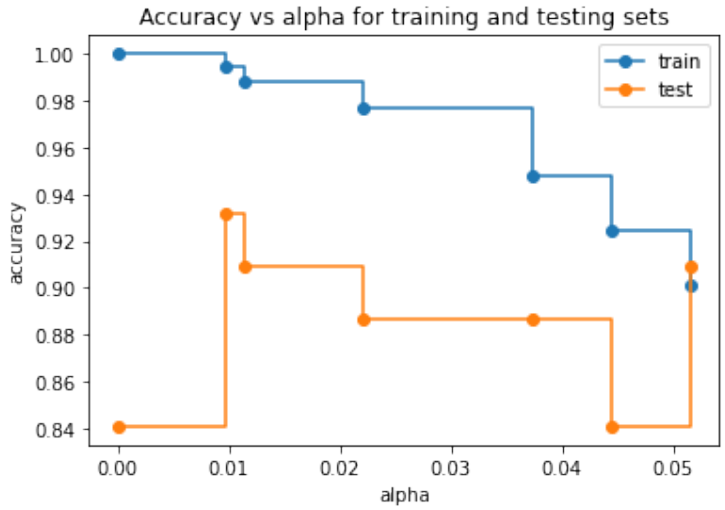

Figure 10 Accuracy VS alpha

2.3. Random Forest

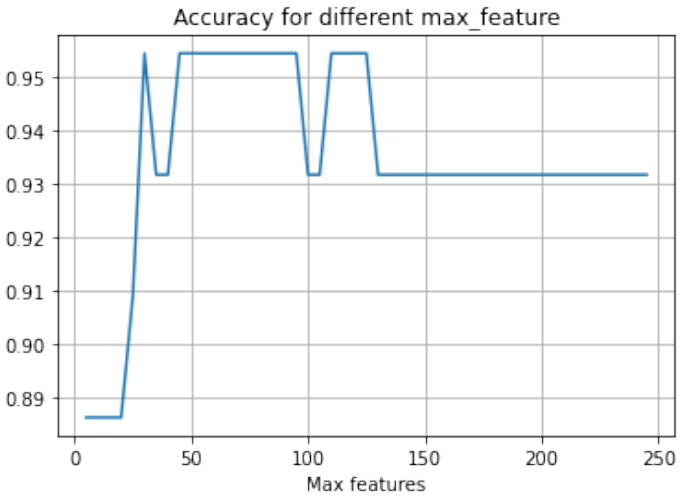

Figure 11 Max features VS accuracy

Random forest performs well on the test datasets. The train time depends on the number of trees in the forest (a 10-member forest only takes 1 second while a 1000-member forest takes around 5 second, which is slower than MLP which has 1000 node, adam, and alpha = 0.001). And can achieves an accuracy of 95.45% in the test datasets. The number of max features does affect the accuracy, and 60 features are chosen as it performs well, as shown in Figure 10, and is close to the square root of the total features in the datasets.

2.4. Multilayer Perceptron

As the number of feature increase significantly, quasi-newton method solver is not only is inefficient in updating the weight value but also failed to converge even though the iteration number is sufficiently large. SGD also trains the MLP slower than ADAM. Thus, ADAM is chosen as the solver.

To have a systematic understanding of the neural network, we evaluate the accuracy of the trained MLP while varies some of the parameters, including alpha, node number and solver. The MLP performs extremely well on the test datasets. However, it is also the slowest among all, especially with the increase of nodes in the hidden layer, as shown in Table 5. Increasing the number of nodes does not improve the accuracy.

Table 5 Number of layers affect performance.

Number of nodes	Accuracy	Train time (s)
2	0.8409	0.3563
10	0.9727	0.3912
50	0.9727	0.5214
100	0.9545	0.5914
1000	0.9545	4.6788
2000	0.9545	4.9325
4000	0.9318	17.5522

10000	0.9090	23.0391

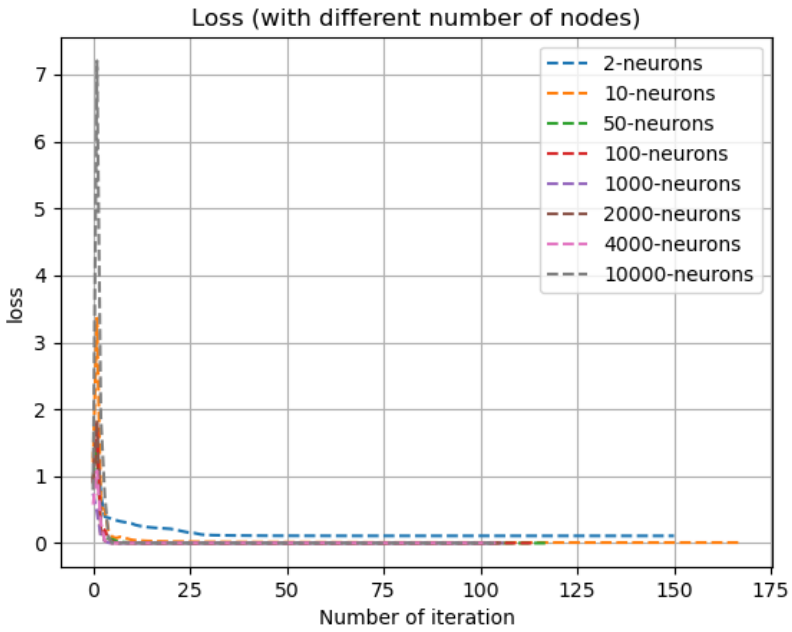

Figure 12 Loss (for different number of nodes) during the training

The L2 regulation improves the performance of MLP, while does not slow down the training process.

Table 6 Different L2 regulation parameter affect train accuracy and speed.

L2 regulation	Accuracy	Train time (s)
1	0.8181	2.5711
0.1	0.9545	2.5681
0.01	0.9318	2.7244
0.001	0.8863	2.6734
0.0001	0.9090	2.7540
0.0005	0.8863	2.6339

Activation function also affects the performance, both on accuracy and train speed. The result is presented in Table 7. And the result shows that identity and logistic activation function performs poorly in this dataset.

Table 7 Different activation functions affect performance.

Activation function	Accuracy	Train time (s)
identity	Failed	NA
logistic	0.7954	0.0406
tanh	0.9090	0.0273
relu	0.9772	0.0378

Finally, Gridserach algorithm is used to find the optimal alpha (L2 penalty), and eventually, an accuracy of 97.7% is achieved.
It should be noticed that MLP sometimes struggles to train the model and especially when the node is greater than 4000.

2.5. Summary

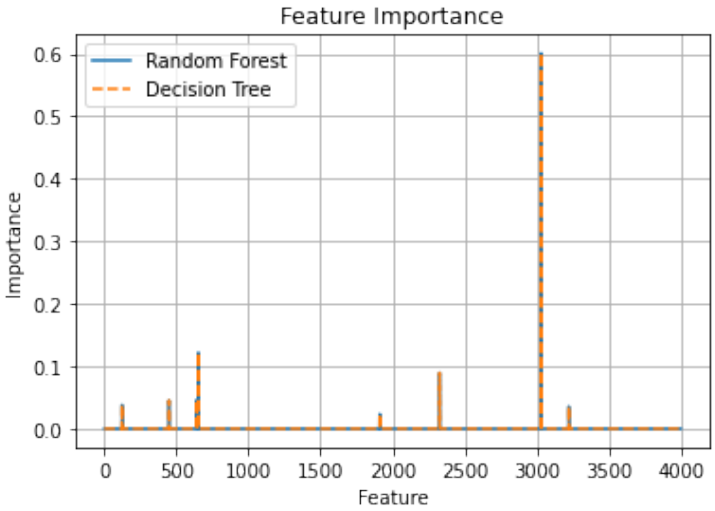

Figure 13 Importance among all features

The trees generated by the decision tree and the random forest are relatively small, with indicates that many of the features do not contribute to distinguishing the label. It also can be found by plotting the importance of the decision as shown in Figure 12. Both algorithms consider only a few key features and the weight on those features are almost identical. This indicates that most of the feature is redundant, and are not as important as the feature *obs3031*, which contribute over 0.6 importance during the classification.
On the other hand, MLP achieves higher accuracy by feeding the same data, which indicate some of the features that are identified as "not important (importance = 0 in Figure 12)" in the decision tree algorithm may contribute to cancer as well.

Table 8 Performance of different algorithms

Algorithm	Accuracy	Training time (s)	Note
Decision tree	0.84	0.2189	No pruning
Decision tree	0.92	0.2197	Post-pruning
Random Forest	0.95	2.6594	500 trees, 60 max features
MLP	0.97	4.6788	1 hidden layer, 1000 node

3. Problem-solving part - CSC 503 only

3.1. Derive the gradient descent update for *b*.

First, gradient needs to be calculated:

$$\frac{\partial E(w)}{\partial b} = \frac{1}{n}\sum_{i=1}^{3}\left(y_i - f_w(x_i)\right) \times \left(-\frac{\partial f_w(x_i)}{\partial b}\right) = -\frac{1}{n}\sum_{i=1}^{3}\left(y_i - f_w(x_i)\right) \times 1$$

Next, the gradient descent update for *b* can be derived as:

$$b^{(new)} = b - \eta\nabla E = b + \frac{\eta}{n}\sum_{i=1}^{3}\left(y_i - f_w(x_i)\right)$$

3.2. Derive the gradient descent update for w_1:

Gradient:

$$\frac{\partial E(w)}{\partial w_1} = \frac{1}{n}\sum_{i=1}^{3}\left(y_i - f_w(x_i)\right) \times \left(-\frac{\partial f_w(x_i)}{\partial w_1}\right) = -\frac{1}{n}\sum_{i=1}^{3}\left(y_i - f_w(x_i)\right) \times x_{1,i} + \lambda \cdot w_1$$

and the new w_1 is:

$$w_1^{(new)} = w_1 - \eta\nabla E = w_1 + \frac{\eta}{n}\sum_{i=1}^{3}\left(\left(y_i - f_w(x_i)\right)x_{1,i}\right) + \lambda \cdot w_1$$

3.3. Derive the gradient descent update for w_3

$$\frac{\partial E(w)}{\partial w_3} = \frac{1}{n}\sum_{i=1}^{3}\left(y_i - f_w(x_i)\right) \times \left(-\frac{\partial f_w(x_i)}{\partial w_3}\right) = -\frac{1}{n}\sum_{i=1}^{3}\left(y_i - f_w(x_i)\right) \cdot x_{1,i} \cdot x_{2,i} + \lambda \cdot w_3$$

And the new w_3 is:

$$w_3^{(new)} = w_3 - \eta\nabla E = w_3 + \frac{\eta}{n}\sum_{i=1}^{3}\left(\left(y_i - f_w(x_i)\right) \cdot x_{1,i} \cdot x_{2,i}\right) + \lambda \cdot w_3$$

3.4. Show the full gradient descent update for the vector ***w***.

From a), b) and c), we have obtained the w_1, *b* and w_3.

Now let drive the gradient decent update for w_2:

$$\frac{\partial E(w)}{\partial w_2} = \frac{1}{n}\sum_{i=1}^{3}\left(y_i - f_w(x_i)\right) \times \left(-\frac{\partial f_w(x_i)}{\partial w_2}\right) = -\frac{1}{n}\sum_{i=1}^{3}\left(y_i - f_w(x_i)\right) \times x_{2,i} + \lambda \cdot w_2$$

And the full gradient descent update for the vector ***w*:**

$$\boldsymbol{w}^{(new)} = \begin{bmatrix} \boldsymbol{w_1} \\ \boldsymbol{w_2} \\ \boldsymbol{w_3} \\ \boldsymbol{b} \end{bmatrix}^{(new)} = \boldsymbol{w} - \eta\boldsymbol{\nabla E} = \begin{bmatrix} \boldsymbol{w_1} \\ \boldsymbol{w_2} \\ \boldsymbol{w_3} \\ \boldsymbol{b} \end{bmatrix} + \eta \begin{bmatrix} \frac{1}{n}\sum_{i=1}^{3}\left(y_i - f_w(x_i)\right) \times x_{1,i} + \lambda \cdot w_1 \\ \frac{1}{n}\sum_{i=1}^{3}\left(y_i - f_w(x_i)\right) \times x_{2,i} + \lambda \cdot w_2 \\ \frac{1}{n}\sum_{i=1}^{3}\left(y_i - f_w(x_i)\right) \cdot x_{1,i} \cdot x_{2,i} + \lambda \cdot w_3 \\ \frac{1}{n}\sum_{i=1}^{3}\left(y_i - f_w(x_i)\right) \end{bmatrix}$$

References

[1] "scikit-learn: machine learning in Python — scikit-learn 0.24.1 documentation," 05 02 2021. [Online]. Available: https://scikit-learn.org/stable/.

[2] "Random forests - classification description," 02 02 2021. [Online]. Available: https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm#remarks.

[3] Ian Goodfellow, Yoshua Bengio and Aaron Courville, Deep Learning, MIT Press, 2016.

[4] Conrads, Thomas P., Vincent A. Fusaro, Sally Ross, Don Johann, Vinodh Rajapakse, Ben A. Hitt, Seth M. Steinberg, "High-Resolution Serum Proteomic Features for Ovarian Cancer Detection," *Endocrine-Related Cancer,* p. 163–78, 2004.