I. Introduction

1. Description and motivation

From features like Clump Thickness, Uniformity of Cell Size/Shape, Marginal Adhesion, etc I want to find the best model that helps classify whether the cancer is benign (noncancerous) or malignant (cancerous). The prediction is important because determining where the cancer is dangerous or not is critical for the patient. False prediction can lead to serious consequence (for example, the patient may suffer from huge anxiety if we output the result that the patient has malignant while she does not have it). With task requiring huge accuracy like this, finding the right classifier is very important.

2. Factor to determine the best classifier

Accuracy: the model needs to perform well on training data, and it also need to perform as good on the data it has not seen before – validation data (not underfit and overfit).

Computational time: the faster the model runs to train the data / make prediction, the better the model is

II. Methods

1. Data preprocessing

I used pandas to read the data, and I discarded the meaningless id column. Then, I impute the missing values (the columns that contains '?') with the value of the nearest neighbor. The final data contains 699 entries, with 9 features for the input and 1 class (of value 0 for benign and 1 for malignant). I map the output from 2 and 4 to 0 and 1 since it is more intuitive for machine learning/deep learning models (0 signals noncancerous and 1 signals cancerous).

Also, for deep learning models, I hot encoded the data with pandas to use softmax activation function at output layer of the neural network.

Extension 1 done: I impute the missing data with NN – select the neighbor that is closest to the missing value to impute the data.

2. Code packages/library used

I used python with the following packages:

- Pandas and NumPy to preprocess the data
- Scikit-learn to build and train the models that are not Neural Network (k-nearest neighbors, Decision tree, etc)
- TensorFlow to build and train the Neural Network models
- Matplotlib to plot the result of cross-validation

III. Result

For each model, I perform 10-fold cross-validation and graph the average score (accuracy) of each model with specific hyperparameter on the validation set.

See my READme.md for instructions on how to install the appropriate dependencies and replicate my result.

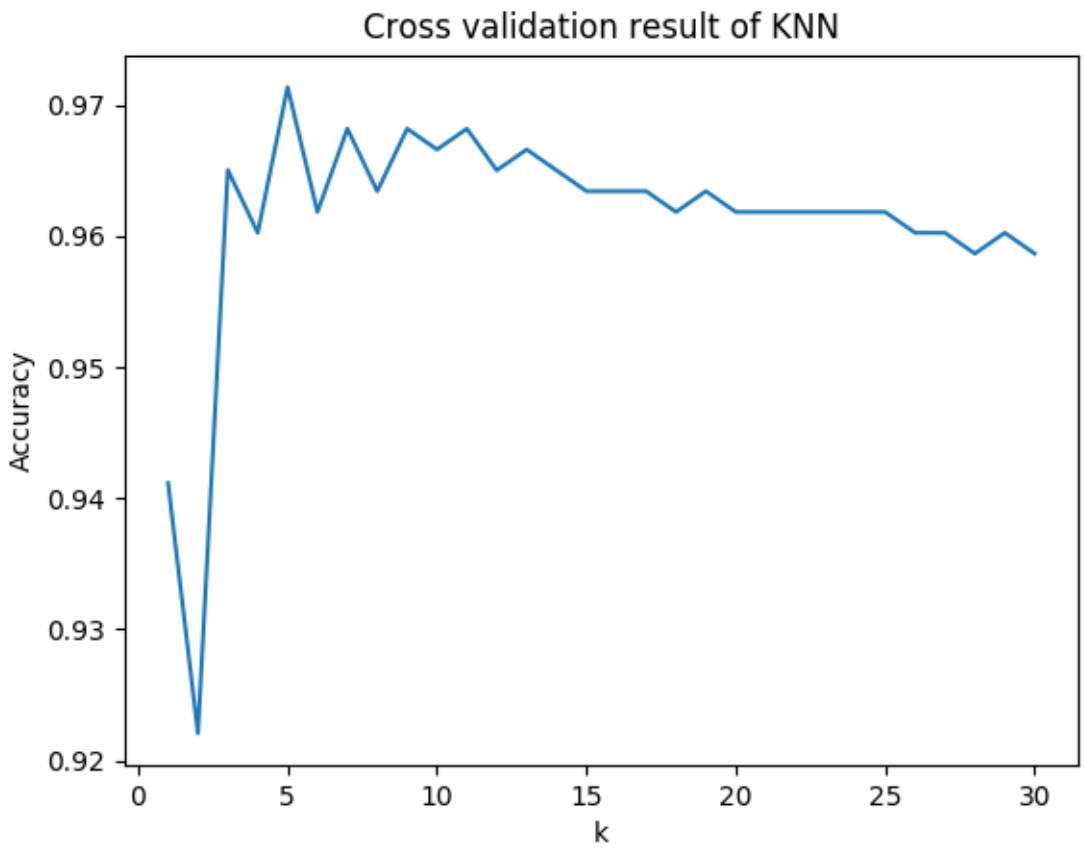
1. k-nearest neighbors

k-nearest neighbors is a machine learning algorithm that based on the similarity defined by distance between data points to make prediction. When encounter an unseen observation, kNN will perform majority vote (or get the average for regression problem) on k seen observations that is closest to the unseen one to determine the class value of it.

Advantage: training is very fast, and how the result gets predicted is very intuitive.

Disadvantage: finding k nearest points is very time consuming

I performed cross-validation on k values ranging from 1 to 30, and this is the result:



Best performance achieved at k = 5 (with mean accuracy on validation folds is 0.9714)

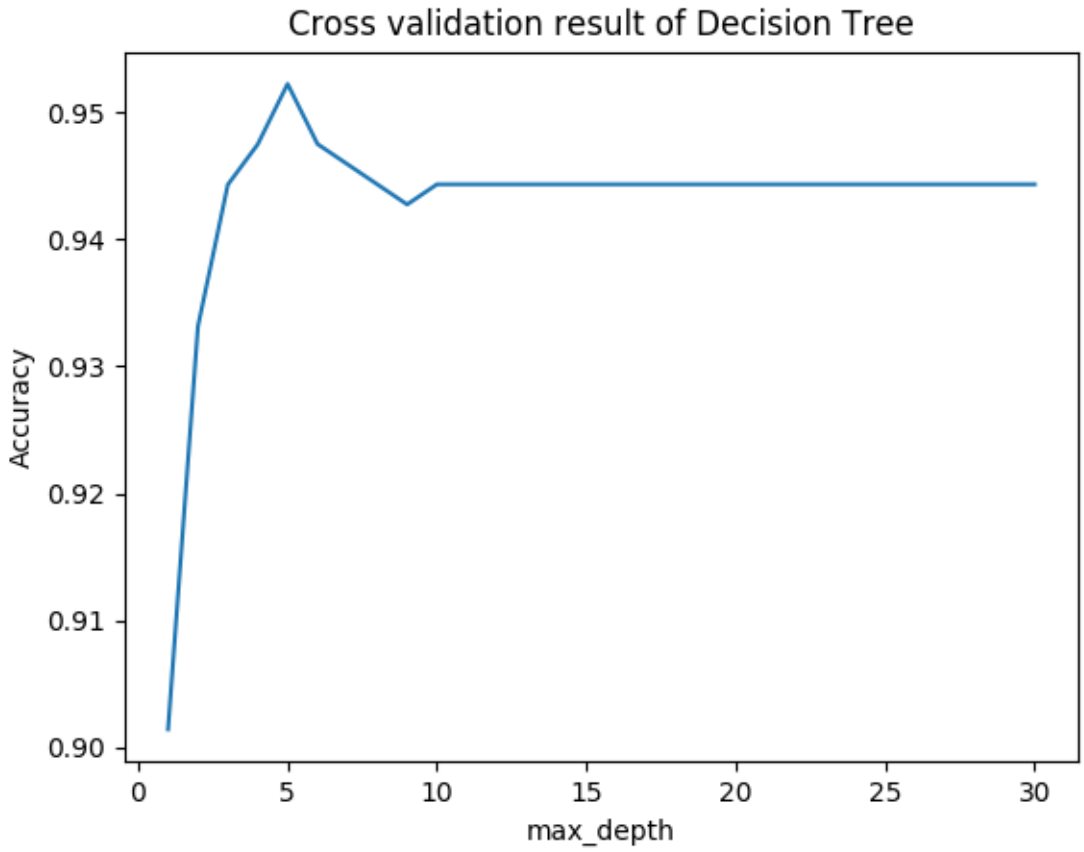
2. Decision tree

Decision tree is a machine learning algorithm that has a root node followed by a series of decision nodes that try to split the data and maximize the homogeneity of each node (each node will be as pure as possible). The leaf node will contain the prediction.

Advantage: work for both numerical and categorical data, easy to explain and produce more intuitive result

Disadvantage: Easy to get overfitted as it is affected by noise in the data.

I performed cross-validation on max depth values of the tree ranging from 1 to 30, and this is the result:



Best max depth of the decision tree is at 5 (with mean accuracy on validation folds is 0.9523)

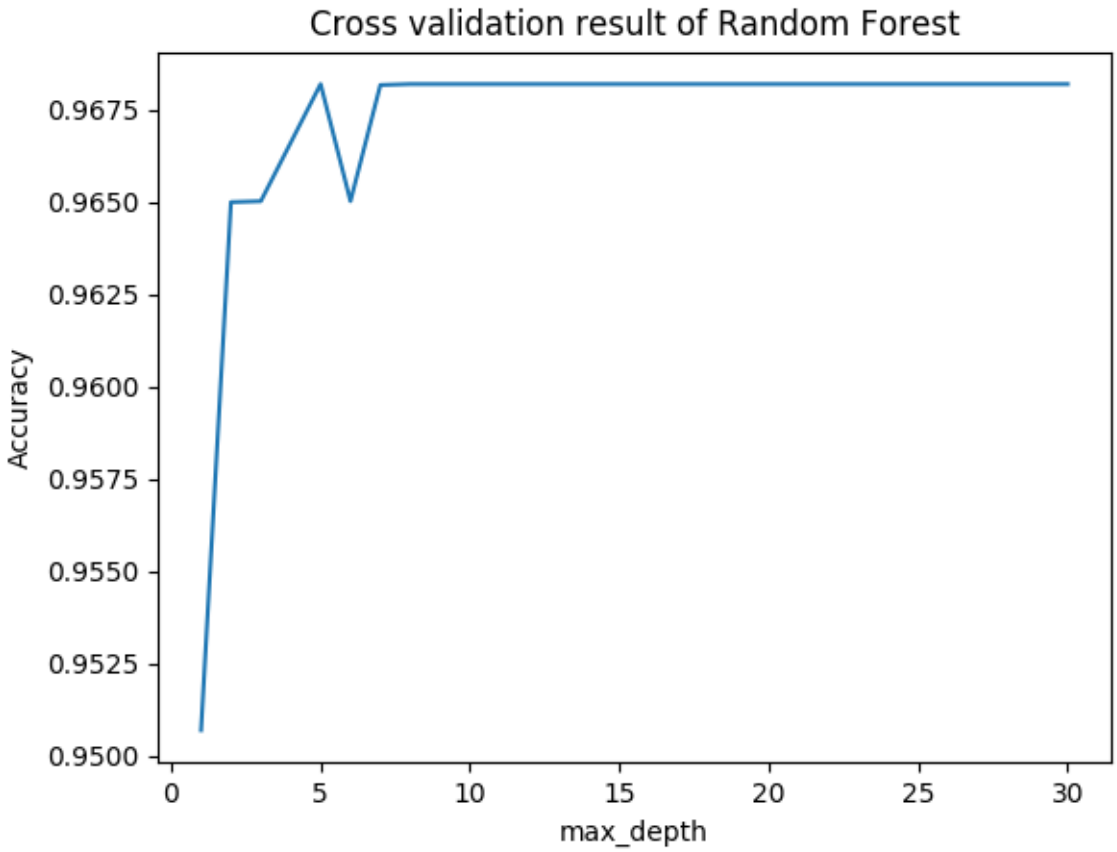
3. Random Forest

Random Forest is a bagging machine learning algorithm that contains large number of individual decision trees, where each tree will predict a specific class label and then major voting will decide the vote for final prediction. The idea is for all the individual tree with low performance to work together to output a strong result.

Advantage: avoid overfitting (the problem of single decision tree), being able to combine result of different decision trees to have a stronger model. It also can solve both classification and regression problem.

Disadvantage: The model created is too complicated that there is no intuitive reasoning behind. The training time is also huge since there is a lot of trees (in my case it is 100 trees).

I performed cross-validation on max depth values of each tree ranging from 1 to 30, and this is the result:



Best max depth of the trees in the random forest is at 5 (with mean accuracy on validation folds is 0.9682)

4. SVM using the polynomial kernel

Support Vector Machine is a machine learning model that works on both classification and regression problems. Intuitively, the model creates a plane that separate the data into classes by maximizing the margin of the data points toward the plane.

The model would only work on cases when the data is not linearly separable. We can perform data preprocessing and transform the data into higher dimension – for example (x) will become (x, x^2).

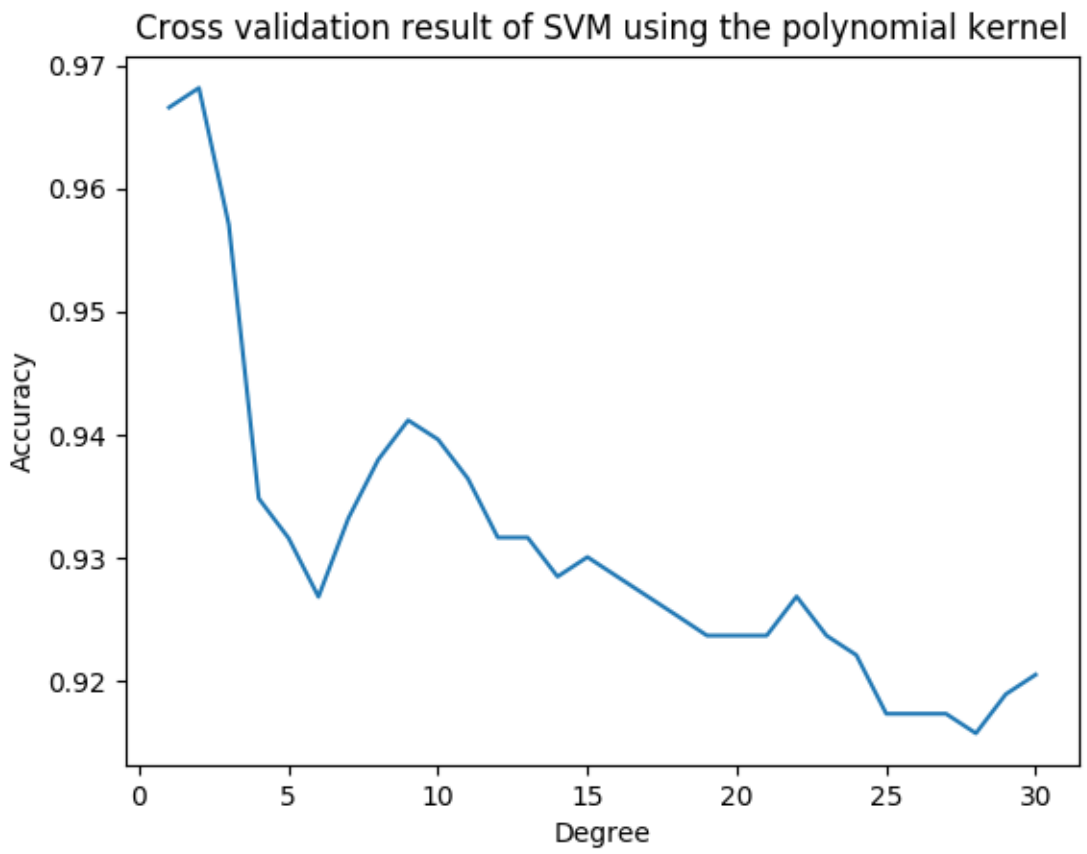
The kernel function will alleviate the transformation process – to make the computation and storage more efficient.

In case of SVM with polynomial kernel, the kernel will be $(a.b + r)^d$ with d is the dimension of the kernel.

Advantage: work very well if the right degree is chosen and the data is clearly separated – or the violation is minimal (where we can perform soft margin classification). And it also work well when the dimension is high.

Disadvantage: SVM does not work so well with large data set, and if the degree is too high there is no intuitive explanation for the result.

I performed cross-validation on the degree from 1 to 30, and this is the result:



Best degree value of the algorithm is at 2 (with mean accuracy on validation folds is 0.9681).

5. SVM using the RBF kernel

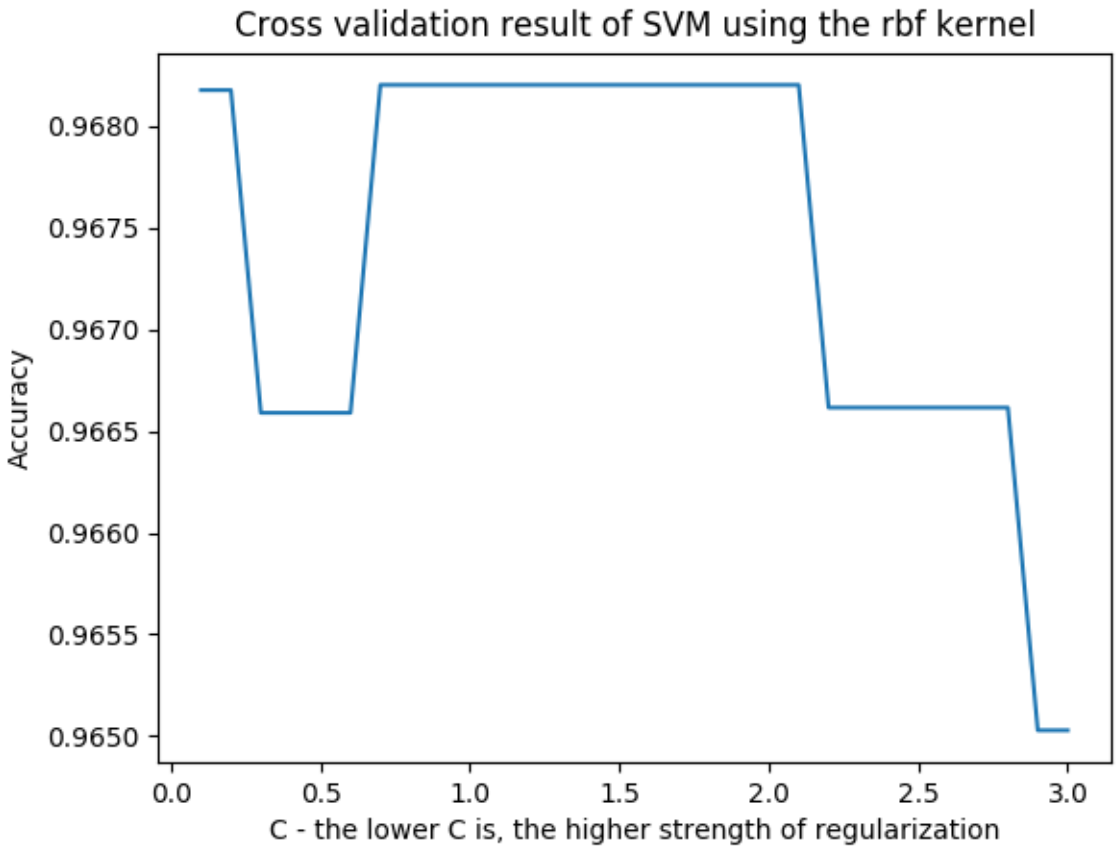
Similar to polynomial kernel, the kernel function will be $e^{-\gamma*(a-b)\wedge 2}$, which project the data onto infinite dimension.

Advantage: It project data into infinite dimension without having to transform and store the huge amount of data.

Disadvantage: the method may not work very well with high dimension problems, since the input space is already good enough, it is not necessary to map it to infinite dimensions.

I performed cross-validation C, a l2-regularization parameters that measure how 'soft' the SVM would be: the bigger the C value, the less strong the regularization will be, and the softer our SVM will be.

1 is a moderate choice and indeed the default one for this model. This is the result of my cross-validation, with C ranging from 0.1 to 3 (with step 0.1):



Best C value of the algorithm is at 0.7 (with mean accuracy on validation folds is 0.9682).

6. Deep neural network with sigmoid activation

Neural network simulates the functionality of human brain, which contains an input layer, followed by a series of hidden layers (usually >= 3 for deep neural network), and an output layer. Each layer will contain nodes, and usually we need big training data to be able to use neural network. We use gradient descent and backpropagation to train the network. The loss function, in this case when we want to do classification, will be cross-entropy loss.

Formula of Sigmoid function:

$$f(x) = \frac{1}{1+e^{-x}}$$

The function will transform the output to be between 0 and 1.

Advantage of neural network: Deep neural network can theoretically learn for any data given that the data is big enough, flexible with both classification and regression problems, good for data with high dimension or large number of inputs.

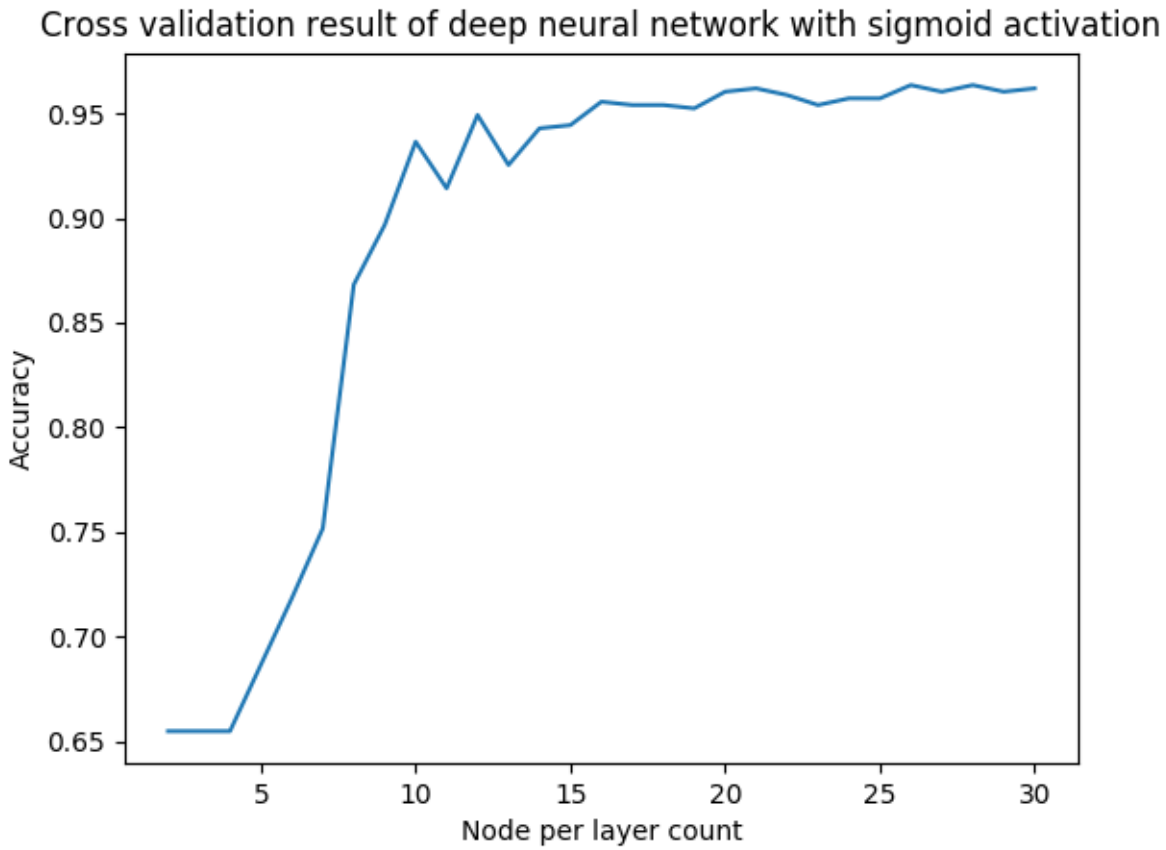
Disadvantage of neural network: no intuitive reasoning on the network, computationally expensive, and require a lot of training data.

Advantage of Sigmoid: differentiable, be able to constraint the output

Disadvantage of Sigmoid: can make gradient vanish, less computationally efficient (compared to ReLU).

My network structure: the input layer will just be the features of the data. I am having 3 hidden layers (with sigmoid activation), each layer has n nodes (n is the hyperparameter I will tune later on). And the output has 2 nodes (with softmax activation), which sums up to 1 and helps indicating whether the patient has benign or malignant (for example benign = 0.9 and malignant = 0.1 → the patient is classified as benign)

I performed cross-validation on the node per layer value (n) from 2 to 30, and this is the result:



Best n value for the network is 26 with mean accuracy 0.9634

7. Deep neural network with ReLU activation

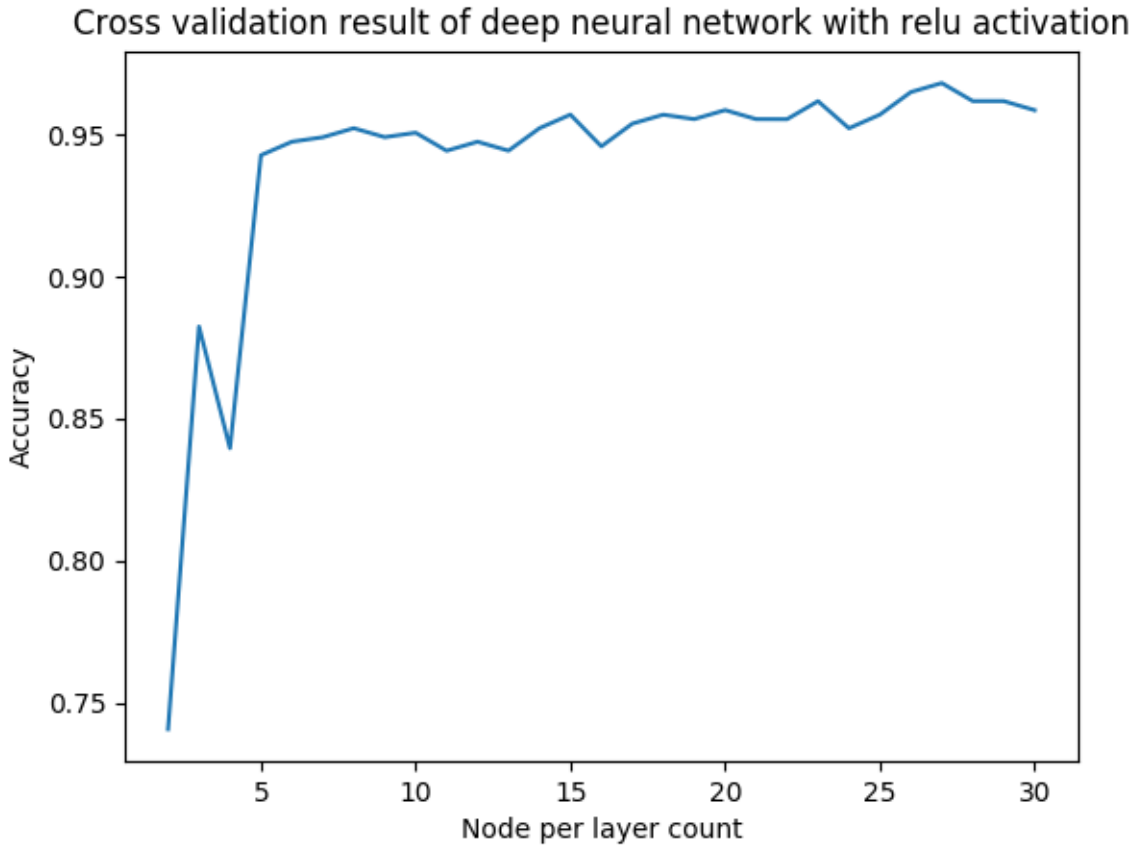
The idea and network structure is exactly the same as section 6, except that we use ReLU instead.

Formula of ReLU: f(x) = max(0, x)

Advantage of ReLU: computationally cheap, avoid gradient vanishing, have better performance in practice than Sigmoid.

Disadvantage of ReLU: if many nodes have value less than 0, ReLU will make everything 0. With no constrain on positive values, ReLU can also blow up the node values.

Same range for number of node per layer (2 → 30), and same network structure, I performed cross-validation and receive the following result:



Best n value for the network is 27 with mean accuracy 0.9682

IV. Discussion

1. Performance & runtime comparison

name	duration_train_ms	duration_val_ms	training_accuracy	validation_accuracy	num_train_samples	num_val_samples
KNN	0.827074051	2.959728241	0.971383148	0.971428571	629	70
Decision Tree	0.947713852	0.109910965	0.982511924	0.928571429	629	70
Random Forest	102.9725075	7.789611816	0.980922099	0.957142857	629	70
Polynomial SVM	2.346515656	0.219345093	0.974562798	0.942857143	629	70
RBF SVM	2.303361893	0.351667404	0.972972973	0.957142857	629	70
NN with Sigmoid	1111.966848	129.4369698	0.968203485	0.985714257	629	70
NN with ReLU	999.5474815	111.8695736	0.979332268	0.95714283	629	70

The table above shows the performance comparation between 7 models:

- On validation accuracy, Neural Network with sigmoid perform best with accuracy up to 98.57%.
- The training time of KNN, Decision Tree, SVM are all relatively low with only about 1-2 ms. Random Forest takes about 100 ms and the neural network take about 1000 ms.
- Predicting on validation set of KNN, Decision Tree, Random Forest, and SVM are all pretty fast with only 0-10 ms. The neural network's validation takes about 100 ms.

2. Conclusion/lesson learned

- All models achieve a very high accuracy on final validation set – from 92% to 98%
- Decision Tree overfit a lot: 98% accuracy on train but only 92% accuracy on test
- The training and predicting time on validation set is generally low for models that are not neural network
- It takes lots of time for a neural network to train
- Decision Tree take long time to train and validate than others because it is an ensemble model with about 100 trees inside
- For this problem, I will choose kNN – the training and testing time is minimal, and the accuracy is although not the highest but is relatively high with 97.1%.
- Sigmoid neural network fit the data pretty good (98.57%), however, I do not think it is the best fit because we are having only 699 data points so I don't think a neural network is appropriate.
- If given the same task with new dataset, I will spend more time exploring the data, maybe graph the relationship between features and class and get an intuition of the data before starting with classification task.
- The project does not have much data, so every model can fit on it easily. Even decision tree, which personally I think is not the best model when using alone, can fit on the data with 92.8% accuracy. Moreover, neural network should not be used with such little amount of data.