Machine Learning Project Report

Worked on the following extra credit tasks -

- (Extra Points #1: 10 Points) You may earn up to 10 extra points if you analyze all four datasets using an additional algorithm. That is, in this case you would be evaluating more than just two algorithms on each dataset.
- (Extra Points #2: 10 Points) You may earn up to 10 extra points if you select a new challenging dataset and evaluate the performance of different algorithms on it. A challenging dataset should necessarily include both categorical and numerical attributes and have more than two classes; or it should be a dataset where you have to classify images.

Discuss which algorithms you decided to test on each dataset and why.

All the algorithms used k- fold stratified cross validation for data preparation. K is set to 10 folds. For each iteration, one fold is used as the testing set, while the other 9 folds are used as training sets. The splitting of training and testing sets are done in a stratified way.

The algorithms used to test on each dataset -

1. **Neural Networks**

   Neural Networks are a deep learning model that has a number of neurons and a number of layers which are used to process a given set of inputs. Each input is treated as a neuron and activation functions are applied to propagate it forward in the network. The number of hidden layers and the number of neurons can be adjusted based on the input size, the output size and the classification tasks based on the dataset. It is a powerful model with high computational power that can keep learning to generalize better.

   Reasons for using Neural Networks -

   - Mini-batch gradient descent algorithm rapidly converges the cost function to the local minimum and we have used this technique here.
   - There are multiple parameters that can be adjusted to create the best performing neural network for a particular dataset. While this may get confusing, it results in a powerful training model that can be used for a large and complex dataset.
   - They can generalize and predict well on the testing data
   - Since they use a back propagation algorithm, neural networks keep learning to make more accurate predictions by adjusting the weights until the stopping criteria is reached.

2. **Random Forests**

   Random Forests are an ensemble of decision trees. The hyperparameter used here - ‘ntree’ adjusts the number of decision trees in each random forest. They can handle both numerical and categorical attributes by calculating the information gain. The information gain of the numerical attributes is calculated by comparing the information gains
   When it comes to numerical attributes, the columns are first sorted in ascending order. Then, the information gain is calculated for two respective values. The maximum information gain is then selected as threshold for this particular feature.
   The information gains of all the features are compared, and the information gain with the highest value is taken as the feature to split for that node.

   Reasons for using Random Forest -
   - Random Forests are less prone to overfitting because they use majority voting by varying the ntree parameter to set the number of decision trees used for classifying the model. A decision tree memorizes the training data and hence has high-variance. It can cause overfitting of test data, which can be avoided by random forests and majority voting.
   - It has good performance and is computationally less expensive.

3. **K-Nearest Neighbours**

   KNN is a supervised classification algorithm where the hyperparameter, k decides how many nearest neighbors are considered to classify the input instance. The distance metric used here is euclidean distance. The euclidean distance is calculated between the input value and all the instances in the training set. The k closest instances are used for making the decision boundary, that is used to classify the input instance.
   An odd value of k is selected so that there are no ties when predicting the class label.

   Reasons for using K-NN -
   - Even though the implementation is simple, the algorithm can learn complex decision boundaries by setting the optimal value of k.
   - Computationally less expensive.

Digits dataset:

1. Neural Networks

(alpha = 1)

Architecture 1: [64, 8, 10]

Lambda	Accuracy	Precision	Recall	F1 Score	Converged J
0	0.92037976	0.92818502	0.92035948	0.92424212	0.244538
0.1	0.91144121	0.91943102	0.91122876	0.91530311	0.41782
0.2	0.92209449	0.92950778	0.92196078	0.92570639	0.54955

Architecture 2: [64, 16, 10]

Lambda	Accuracy	Precision	Recall	F1 Score	Converged J
0	0.93947105	0.94451737	0.93928105	0.94187924	0.1555543
0.1	0.94444086	0.94928599	0.94441176	0.94683978	0.3202470
0.2	0.92995447	0.93814478	0.92970588	0.93389563	0.4398464

Architecture 3: [64, 16, 8, 10]:

Lambda	Accuracy	Precision	Recall	F1 Score	Converged J
0	0.9241935	0.93053588	0.9240719	0.9272814	0.115656492
0.1	0.94409952	0.9511089	0.94506536	0.94506945	0.3256190
0.2	0.92321181	0.9323361	0.92313725	0.92770038	0.48637557

Architecture 4: [64, 16, 16, 10]

Lambda	Accuracy	Precision	Recall	F1 Score	Converged J
0	0.93043563	0.93655785	0.93042484	0.93347104	0.105471279
0.1	0.9382956	0.94483471	0.93833333	0.94156728	0.274793769

0.2	0.9260051	0.9349063	0.92575163	0.93027081	0.418539360

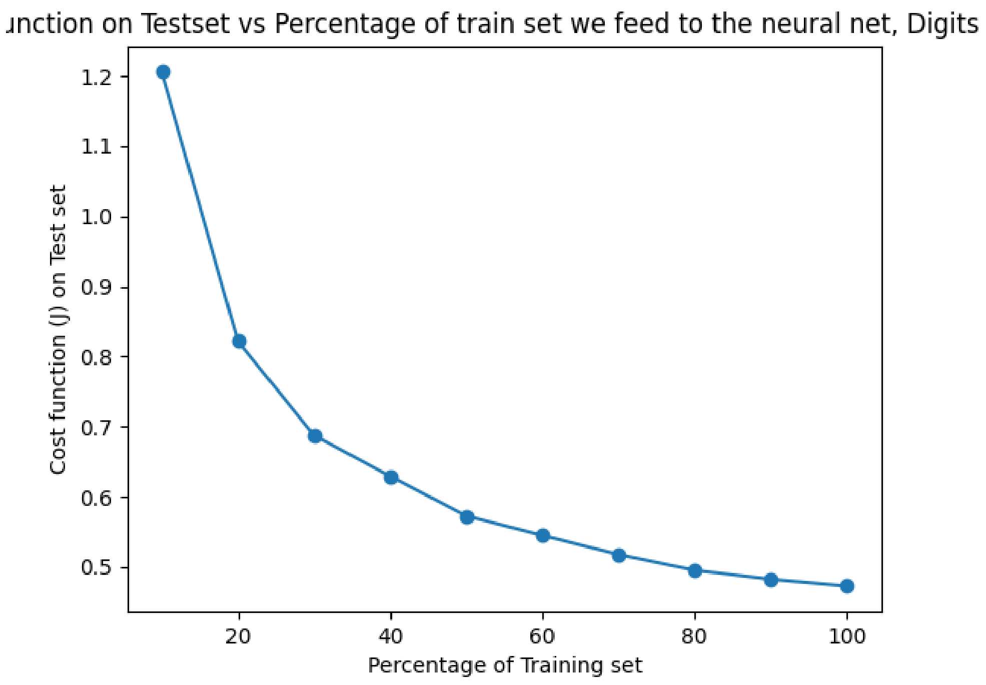
Architecture 5: [64, 16, 8, 4, 10]

Lambda	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**Converged J**
0	0.77734362	0.78296919	0.77622876	0.77813839	0.77085486
0.1	0.79317661	0.79273942	0.79144444	0.79181067	1.037065554
0.2	0.73383925	0.73874452	0.73281291	0.73491347	1.3429483

Architecture 6: [64, 16, 16, 8, 10]

Lambda	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**Converged J**
0	0.92596635	0.93538062	0.92573856	0.93051169	0.09233829
0.1	0.91207737	0.91563006	0.91153595	0.91351904	0.425739994
0.2	0.85636323	0.8710604	0.85584967	0.86323224	0.81705618

Best Architecture - Architecture 2: [64, 16, 10]
Lambda : 0.1
We chose this architecture because of its high accuracy and F1 score.



Graph interpretation:

The cost function on the test set is decreasing as we increase the data we feed to the neural network. This is expected because, the more we feed the neural network, the better it performs on the test set, thus reducing the cost function.

Theory and Observations:

We used alpha = 1 for the dataset. When we trained with alpha = 10, the cost function was overshooting, and when we trained with alpha = 0.1, the cost function was decreasing by a very slight value and taking more iterations to train. Hence, we chose alpha = 1 in which the learning was reasonably fast and did not cause any overshooting.

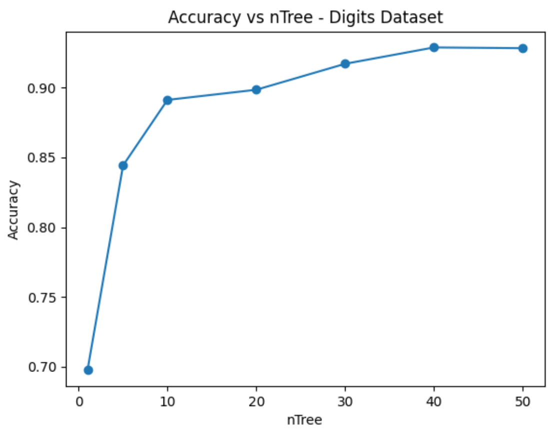
We chose 6 architectures for all the datasets by increasing the number of neurons and number of layers gradually.

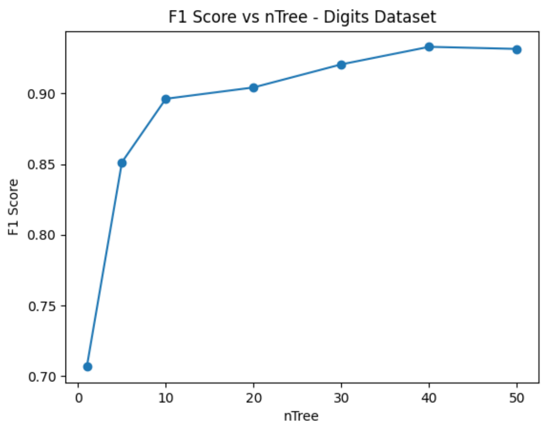
Regularization parameters are used to reduce the overfitting of complex models. With reasonable values of lambda and reasonable increase in those values, we should ideally see an increase in the performance for complex models.

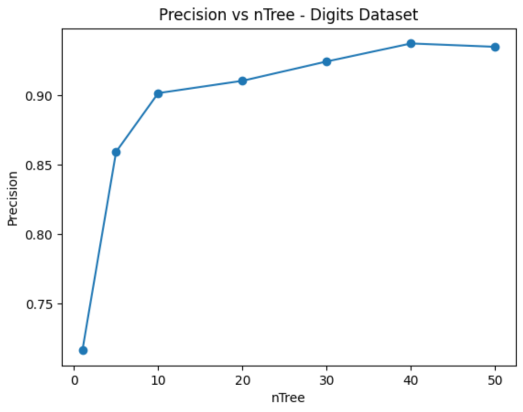
Neural networks with more layers and a greater number of neurons in each layer are expected to perform well because they take complex patterns from the dataset and give us better results. If we make our neural network more complex, it will overfit, and the accuracy drops. So, we must choose a reasonable number of layers and neurons to train our model.

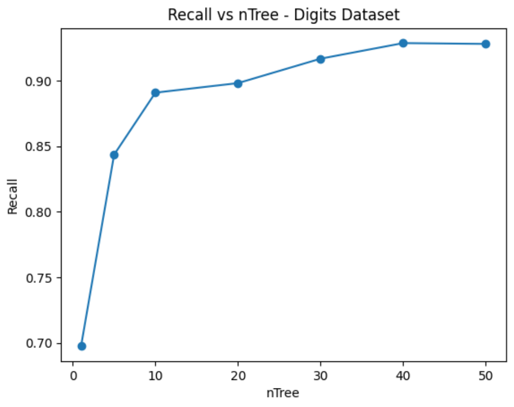
We notice that the neural network performs better for simpler architectures as compared to more complex architectures. This could be because of overfitting. Usually, we should see an increase in performance for complex networks when we increase the value of lambda. But here, the performance is increasing from lamba = 0 to 0.1 and dropping after 0.1. The performance is decreasing when we increase regularization constant post 0.1. This means all our models are mostly simple and we are penalizing the model by adding more regularization constant.

2. Random Forest









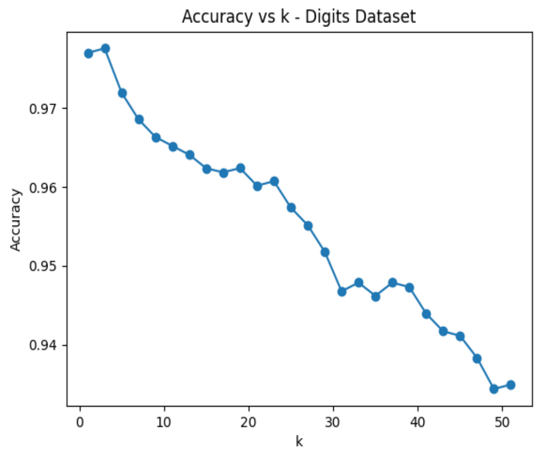
nTree	Accuracy	Precision	Recall	F1 Score
1	0.69762328	0.71650163	0.6974183	0.70667795
5	0.84422773	0.85937423	0.84366667	0.85141336
10	0.89117448	0.90162055	0.89104575	0.89629201
20	0.89845642	0.91047677	0.89833333	0.90433007
30	0.91704718	0.92438382	0.91699346	0.92066423
40	0.92875965	0.93733228	0.92888889	0.93307197
50	0.92822036	0.93492295	0.92826797	0.93157904

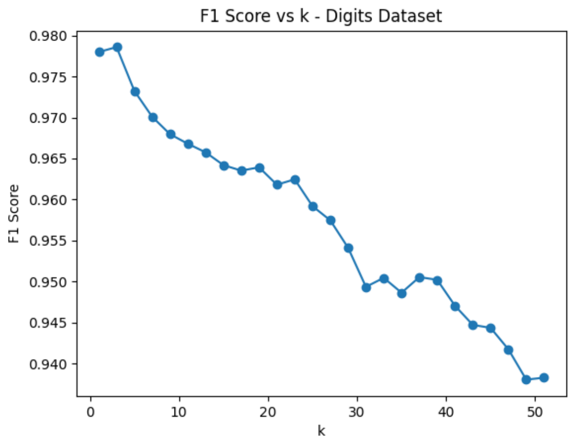
Best hyperparameter - 40
Accuracy - 0.92875965, F1 Score - 0.93307197 (Best F1 Score and Accuracy)

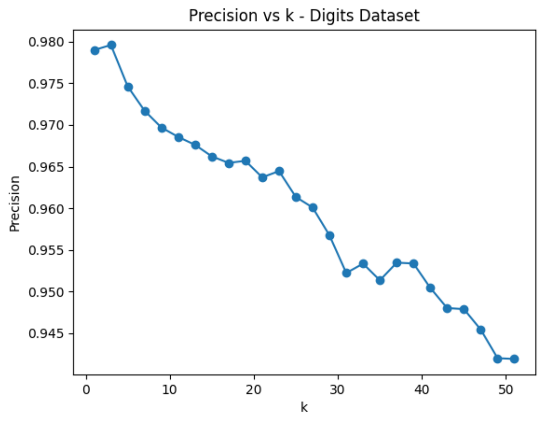
Graph Interpretation

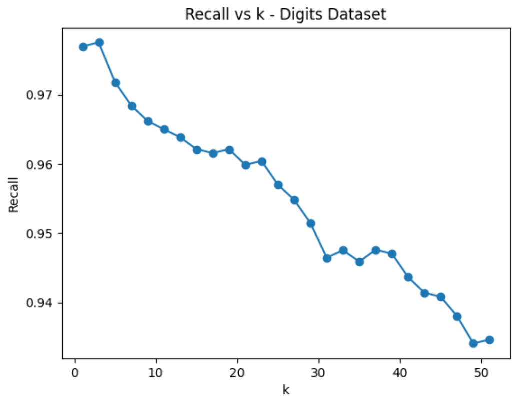
We can observe that the accuracy and the F1 score keeps increasing till ntree = 40. It then remains stable from ntree = 40 to 50. The learning increases rapidly from ntree= 1 to 20 trees. The learning slows down and increases in small amounts from 20 to 40 trees. Also, the model performs the best at ntree = 40 and hence is chosen as the best hyperparameter.

3. K Nearest Neighbours









k	Accuracy	Precision	Recall	F1 Score
3	0.97757613	0.97959267	0.9775817	0.97858365
25	0.9573869	0.96135938	0.95702614	0.95918417
51	0.93494365	0.94191779	0.93464052	0.93825631

Best k value - 3 (Best F1 Score and Accuracy)
(Evaluated k from k=1 to k=51, for all odd values of k)

Graph Interpretation

From the graphs, we can observe that the algorithm performs best when the value of k is set to 3. The accuracy and the F1 score is maximum at this value. After that, we notice that the accuracies and the F1 scores are decreasing gradually till k = 51. As k keeps increasing, we add many wrong class elements as our nearest neighbours, and make a prediction based on them. Thus we are seeing a dip in the accuracy as k increases.

Titanic dataset:

1. Neural Networks

(alpha = 1)

Architecture 1: [9, 4, 2]

Lambda	Accuracy	Precision	Recall	F1 Score	Converged J
0	0.80454545	0.80974952	0.77538126	0.79201062	0.82530311
0.1	0.8	0.80035269	0.77331155	0.78648113	0.8653999
0.2	0.79886364	0.80458343	0.76802832	0.78571988	0.8875317

Architecture 2: [9, 8, 2]

Lambda	Accuracy	Precision	Recall	F1 Score	Converged J
0	0.81386364	0.82893938	0.7998366	0.80389805	0.814236858
0.1	0.79431818	0.79064579	0.76922658	0.77972432	0.86612575
0.2	0.80113636	0.80044602	0.77696078	0.78838227	0.888830

Architecture 3: [9, 8, 4, 2]

Lambda	Accuracy	Precision	Recall	F1 Score	Converged J
0	0.81363636	0.816884	0.78714597	0.80155777	0.7935089
0.1	0.80113636	0.81784707	0.76824619	0.79161244	0.881992627
0.2	0.78977273	0.80769789	0.75680828	0.78040985	0.90294330

Architecture 4: [9, 8, 8, 2]

Lambda	Accuracy	Precision	Recall	F1 Score	Converged J
0	0.80909091	0.81044816	0.7828976	0.79629547	0.790466726
0.1	0.79318182	0.80381135	0.76339869	0.78264185	0.87489238
0.2	0.79318182	0.79425696	0.76775599	0.78051189	0.894643814

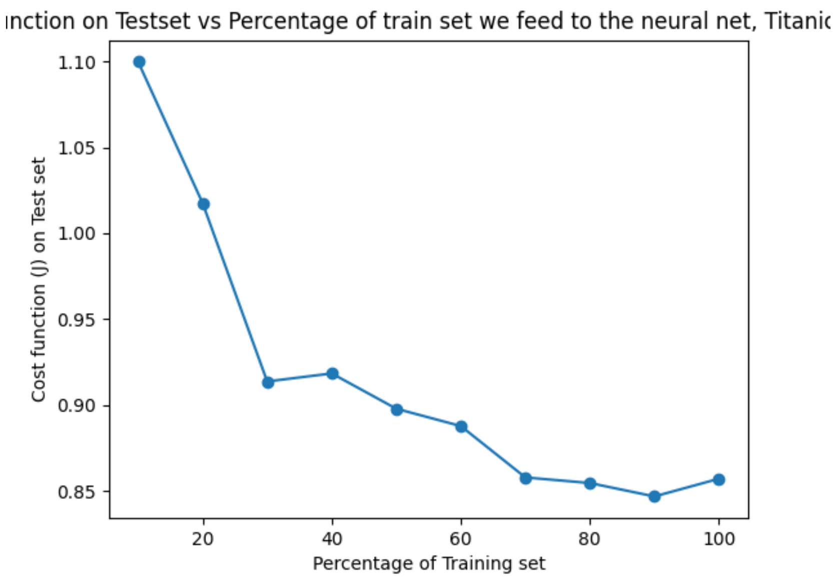
Architecture 5: [9, 8, 6, 4, 2]

Lambda	Accuracy	Precision	Recall	F1 Score	Converged J
0	0.82386364	0.82869252	0.7998366	0.81377661	0.80629491
0.1	0.80568182	0.81255425	0.77521786	0.79330554	0.8726370
0.2	0.79772727	0.81642421	0.76383442	0.78848681	0.9002107

Architecture 6: [9, 8, 8, 8, 2]

Lambda	Accuracy	Precision	Recall	F1 Score	Converged J
0	0.81818182	0.82625045	0.78921569	0.807066	0.79627461
0.1	0.80113636	0.82505925	0.76171024	0.79167025	0.896366319
0.2	0.77045455	0.7919828	0.74215686	0.76471962	0.9206767

Best Architecture - Architecture 5: [9, 8, 6, 4, 2]
Lambda : 0
This is the best architecture because it has high F1 score and Accuracy



Graph Interpretation

The cost function on the test set is decreasing as we increase the data we feed to the neural network. This is expected because, the more we feed the neural network, the better it performs on the test set, thus reducing the cost function.

Theory and Observations:

We incorporated alpha = 1 for this dataset as well. This is because when trained with alpha = 10, the cost function is overshooting and when trained with alpha = 0.1, the cost function is decreasing by a marginal value, that is, it is taking more iterations. Hence, we chose alpha = 1 in which the learning was reasonably fast and did not cause any overshooting.

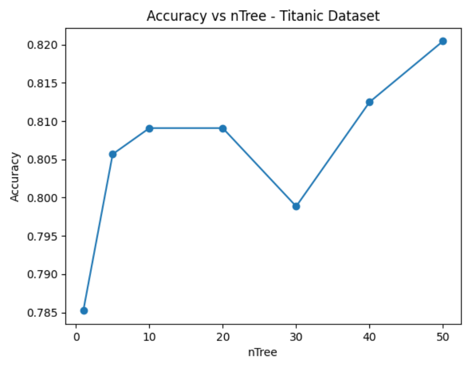
We chose 6 architectures for all the datasets by increasing the number of neurons and number of layers gradually.

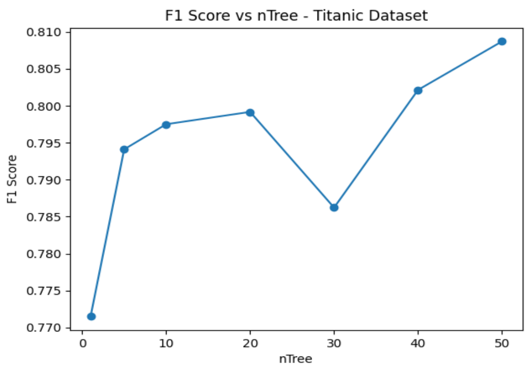
Regularization parameters are used to reduce the overfitting of complex models. With reasonable values of lambda and reasonable increase in those values, we should ideally see an increase in the performance for complex models.

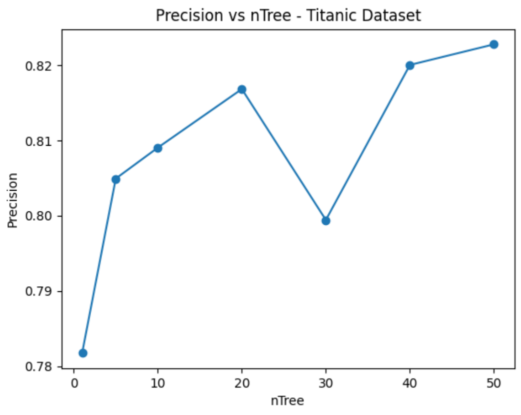
Neural networks with more layers and a greater number of neurons in each layer are expected to perform well because they take complex patterns from the dataset and give us better results. If we make our neural network more complex, it will overfit, and the accuracy drops. So, we must choose a reasonable number of layers and neurons to train our model.

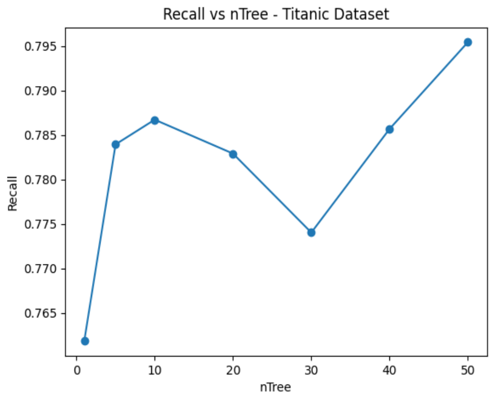
Usually, we should see an increase in performance for complex networks when we increase the value of lambda. But here, we are seeing the reverse. The performance is decreasing when we increase regularization constant. This means all our models are mostly simple and we are penalizing the model by adding more regularization constant.

2. Random Forest









nTree	**Accuracy**	**Precision**	**Recall**	**F1 Score**
1	0.78522727	0.78173514	0.76181917	0.77149481
5	0.80568182	0.80493838	0.78393246	0.79411126
10	0.80909091	0.80904885	0.78671024	0.79751243
20	0.80909091	0.81684431	0.7828976	0.79918
30	0.79886364	0.79943715	0.77401961	0.78623058
40	0.8125	0.82005367	0.78567538	0.80215673
50	0.82045455	0.8227937	0.79542484	0.8087192

Best hyperparameter - nTree = 50 (High F1 Score and Accuracy)
Accuracy - 0.82045455, F1 Score - 0.8087192

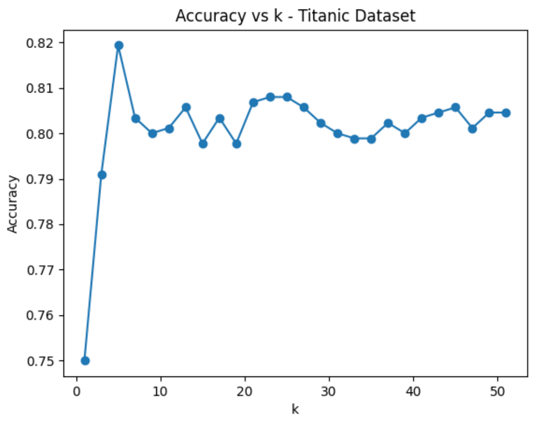
Graph Interpretation

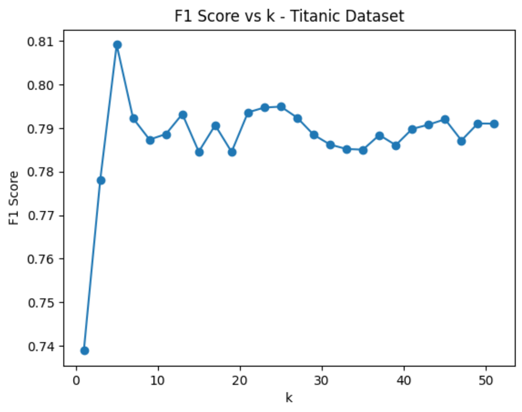
The performance of the random forests is best at ntree = 50 as we can see that the accuracy and F1 Score are maximum at this value of ntree. From the graphs, we can observe that performance increases from ntree = 1 to ntree = 20 rapidly, and then decreases from ntree = 20 to ntree = 30. Once again, it increases rapidly from ntree = 30 to ntree = 50. This could be because the model becomes more efficient when the number of trees increases. Initially, we notice a sudden drop in performance at ntree = 30.

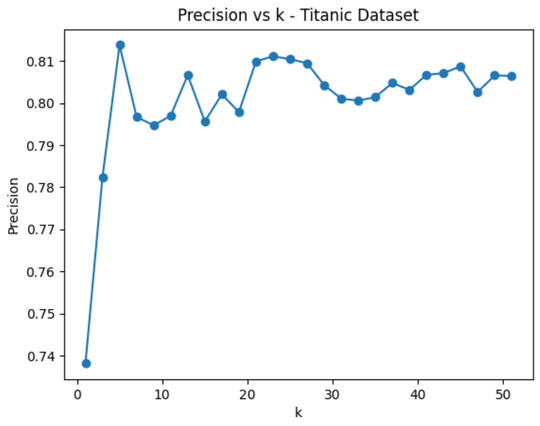
Probable reasons for dips in the curve:

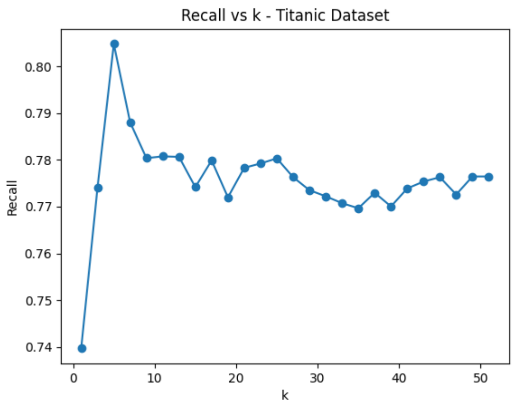
1. We are not training 50 trees for each fold and taking predictions from nTree trees for each nTree value. I am training a new set of nTree trees every time with new bootstrap datasets. Hence the graphs would be little random and we can see low accuracies even for higher nTrees values. Example we see a dip at nTree = 30
2. Also since we are randomly creating bootstrap datasets we can assume that few trees might be fed with really easy data which results in less accuracy.
3. We are randomly choosing features at each node

3. K Nearest Neighbours









k	Accuracy	Precision	Recall	F1 Score
5	0.81931818	0.81377036	0.80484749	0.8091802
25	0.80795455	0.81039801	0.78033769	0.79495418
45	0.80568182	0.8086982	0.77630719	0.79204865

Best value of k = 5 (High F1 score and Accuracy)
(Tested for k values from k =1 to k =51, for all odd values of k)

Graph Interpretation

We observe that the performance increases till k = 5, and decreases rapidly from k =5 to k = 10. After that it keeps decreasing slowly and remains almost stable till k = 51. The performance is measured here based on accuracy and F1 score. This might be because the decision boundaries have become simple and the training data is not generalized well.

<u>Loan Dataset:</u>

1. Neural Networks

(alpha = 1)

Architecture 1: [21, 4, 2]

Lambda	Accuracy	Precision	Recall	F1 Score	Converged J
0	0.77626812	0.76564689	0.69195804	0.72644249	0.78229514
0.1	0.80769928	0.84426156	0.70428904	0.7673371	0.9213882
0.2	0.80353261	0.83523316	0.69944056	0.76083138	0.943498

Architecture 2: [21, 8, 2]

Lambda	Accuracy	Precision	Recall	F1 Score	Converged J
0	0.75751812	0.73755056	0.67468531	0.70364966	0.643808552
0.1	0.78686594	0.81845807	0.68913753	0.74719781	0.93375305
0.2	0.80769928	0.84426156	0.70428904	0.7673371	0.9530871

Architecture 3: [21, 8, 4, 2]

Lambda	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**Converged J**
0	0.75724638	0.72889044	0.66923077	0.69715238	0.65654071
0.1	0.80144928	0.83226915	0.69974359	0.75941964	0.92743440
0.2	0.80561594	0.83199512	0.70822844	0.76429552	0.967183858

Architecture 4: [21, 8, 8, 2]

Lambda	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**Converged J**
0	0.75307971	0.72830795	0.68307692	0.70373206	0.70602989
0.1	0.79728261	0.81753134	0.70216783	0.75427117	0.94128083
0.2	0.80144928	0.83649619	0.69610723	0.75912642	0.962547238

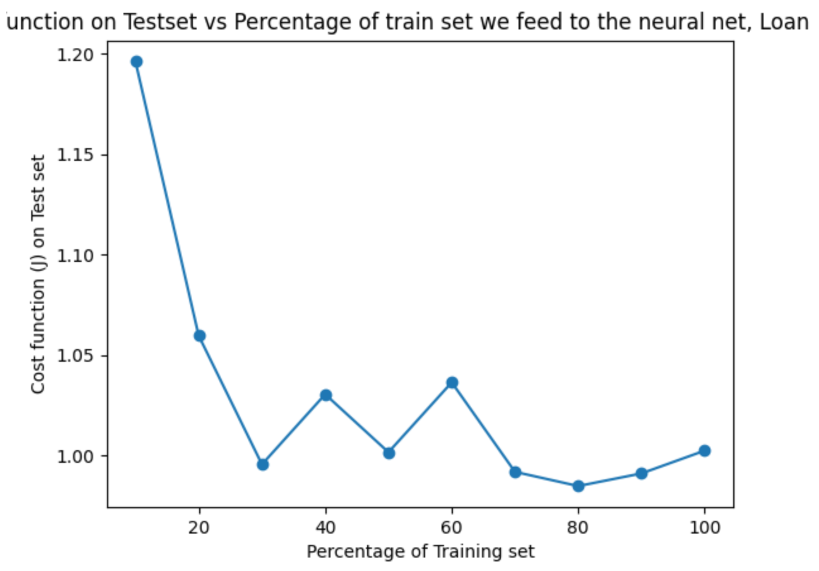
Architecture 5: [21, 8, 6, 4, 2]

Lambda	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**Converged J**
0	0.76168478	0.73616998	0.68834499	0.71067943	0.797438547
0.1	0.79519928	0.79488656	0.68247086	0.73075133	0.978535243
0.2	0.74673913	0.59626547	0.59545455	0.58482707	1.10860084

Architecture 6: [21, 8, 8, 8, 2]

Lambda	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**Converged J**
0	0.75960145	0.75067061	0.67671329	0.71055701	0.7824560
0.1	0.79728261	0.81662756	0.69853147	0.75212198	0.97096690
0.2	0.78894928	0.7432216	0.67247086	0.70048691	1.01932192

Best Architecture - Architecture 1: [21, 4, 2]
Lambda : 0.1
Best architecture because of high F1 Score and Accuracy



Graph Interpretation

The cost function on the test set is decreasing as we increase the data we feed to the neural network. This is expected because, the more we feed the neural network, the better it performs on the test set, thus reducing the cost function.

Theory and Observations:

We used alpha = 1 for the dataset. When we trained with alpha = 10, the cost function was overshooting, and when we trained with alpha = 0.1, the cost function was decreasing by a very slight value and taking more iterations to train. Hence, we chose alpha = 1 in which the learning was reasonably fast and did not cause any overshooting.

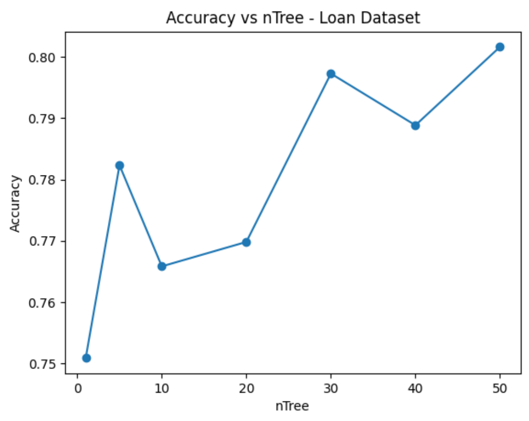
We chose 6 architectures for all the datasets by increasing the number of neurons and number of layers gradually.

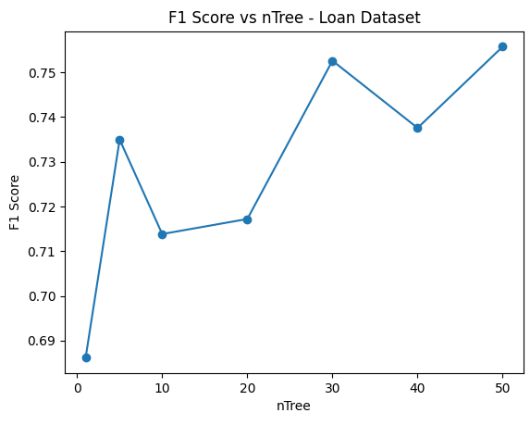
Regularization parameters are used to reduce the overfitting of complex models. With reasonable values of lambda and reasonable increase in those values, we should ideally see an increase in the performance for complex models.

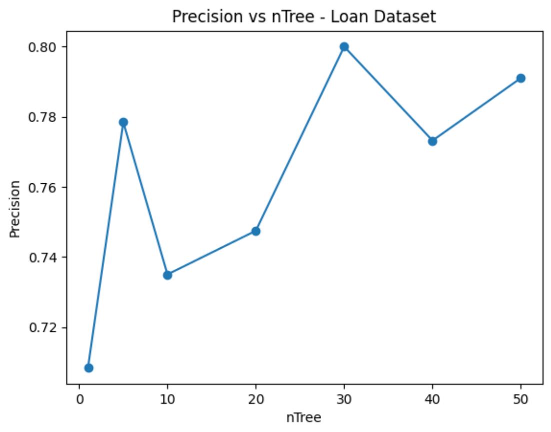
Neural networks with more layers and a greater number of neurons in each layer are expected to perform well because they take complex patterns from the dataset and give us better results. If we make our neural network more complex, it will overfit, and the accuracy drops. So, we must choose a reasonable number of layers and neurons to train our model.

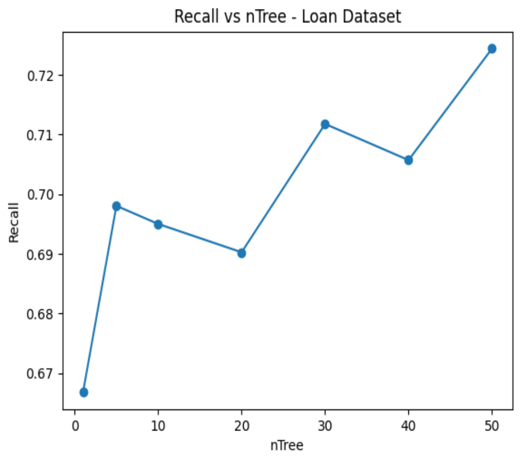
We notice that the neural network performs better for simpler architectures as compared to more complex architectures. This could be because of overfitting. We should see an increase in performance for complex networks when we increase the value of lambda. The performance is good for a reasonable regularization constant and decreasing after 0.1. This means all our models are mostly simple and we are penalizing the model by adding more regularization constant post lambda = 0.1.

2. Random Forest









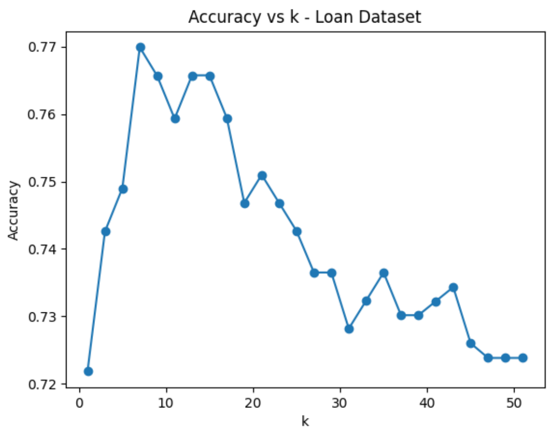
nTree	Accuracy	Precision	Recall	F1 Score
1	0.7509058	0.70837131	0.66678322	0.68615331
5	0.78233696	0.77841766	0.69804196	0.73496594
10	0.76585145	0.73503087	0.69501166	0.71384342
20	0.76983696	0.74747967	0.69025641	0.71720301
30	0.79728261	0.79996808	0.71177156	0.75261336
40	0.7888587	0.77318908	0.70571096	0.7375738
50	0.80163043	0.79101093	0.72440559	0.7557609

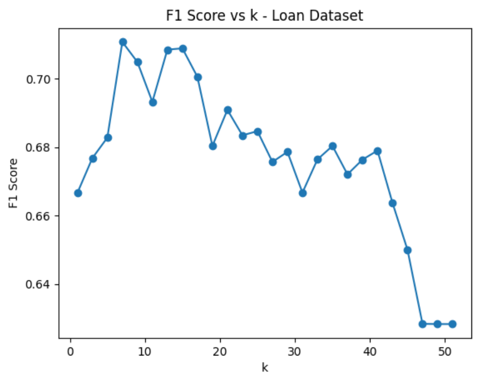
Best hyperparameter - 50 (High F1 Score and Accuracy)
Accuracy - 0.80163043, F1 Score - 0.7557609

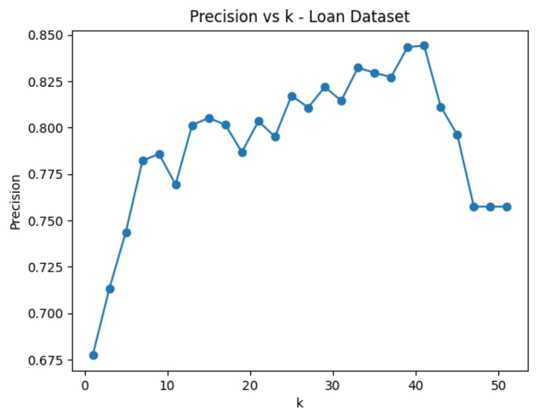
Graph Interpretation

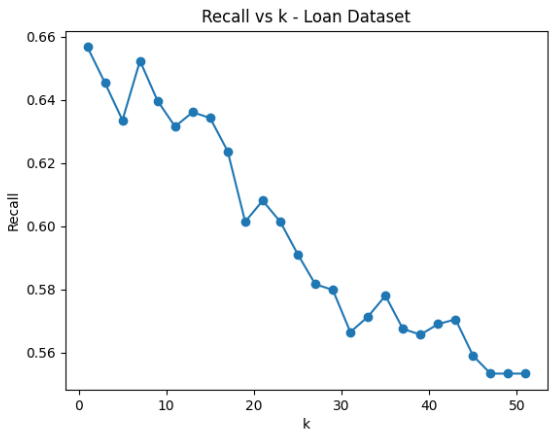
In general, we see that the accuracies and F1 scores are increasing as the ntree value reaches 50. We notice sudden dips at ntree = 20, and ntree = 40. This could be because:
1. We are not training 50 trees for each fold and taking predictions from nTree trees for each nTree value. I am training a new set of nTree trees every time with new bootstrap datasets. Hence the graphs would be little random and we can see low accuracies even for higher nTrees values. Example we see a dip at nTree = 40
2. Also since we are randomly creating bootstrap datasets we can assume that few trees might be fed with really easy data which results in less accuracy.
3. We are randomly choosing features at each node

3. K Nearest Neighbours









k	Accuracy	Precision	Recall	F1 Score
7	0.76992754	0.78220725	0.65235431	0.71081875
15	0.76576087	0.80516977	0.63426573	0.70894827
21	0.75099638	0.80342275	0.60808858	0.69094575

Best Values of k = 7 (High F1 Score and Accuracy)
(Tested for k values ranging from k =1 to k = 51, for all odd values of k)

Graph Interpretation

Here, we observe that the performance is best at k = 7. After that the performance keeps decreasing as the value of k increases. This is because we are including a lot of elements outside the boundary.

Parkinson Dataset:

1. Neural Networks

(alpha = 1)

Architecture 1: [22, 8, 2]

Lambda	Accuracy	Precision	Recall	F1 Score	Converged J
0	0.87333333	0.8210726	0.8225	0.81906612	0.065749517
0.1	0.79833333	0.61018849	0.66916667	0.63623957	0.69289285
0.2	0.825	0.67270192	0.68666667	0.67302016	0.778015028

Architecture 2: [22, 16, 2]

Lambda	Accuracy	Precision	Recall	F1 Score	Converged J
0	0.88833333	0.84533932	0.84666667	0.84439695	0.03808897
0.1	0.84666667	0.77886724	0.74	0.75248071	0.68029265
0.2	0.81	0.66500937	0.67	0.66112541	0.780941123

Architecture 3: [22, 16, 8, 2]

Lambda	Accuracy	Precision	Recall	F1 Score	Converged J
0	0.87333333	0.84098039	0.8025	0.82017804	0.11920925
0.1	0.795	0.59342262	0.66666667	0.62563445	0.725469359
0.2	0.825	0.66182598	0.67416667	0.66220783	0.847005

Architecture 4: [22, 16, 16, 2]

Lambda	Accuracy	Precision	Recall	F1 Score	Converged J
0	0.88666667	0.85463936	0.8325	0.8423547	0.018452896
0.1	0.75666667	0.60849838	0.6475	0.62458996	0.73178677
0.2	0.805	0.68104167	0.64	0.65126009	0.81153658

Architecture 5: [22, 16, 8, 4, 2]

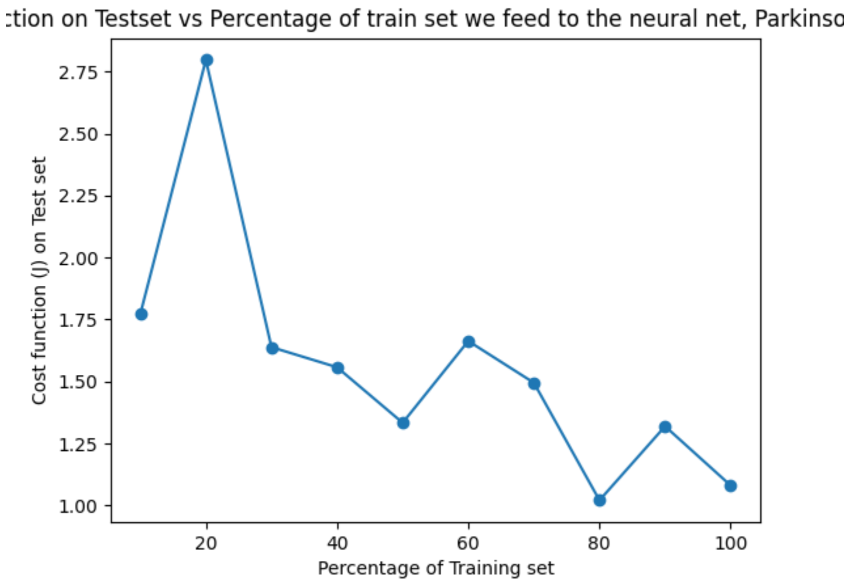
Lambda	Accuracy	Precision	Recall	F1 Score	Converged J
0	0.84166667	0.7992583	0.76916667	0.7801816	0.27900
0.1	0.765	0.50353157	0.55333333	0.52049391	0.842452379

0.2	0.755	0.3775	0.5	0.43015873	1.11973015

Architecture 6: [22, 16, 8, 8, 2]

Lambda	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**Converged J**
0	0.86166667	0.76224161	0.775	0.76534221	0.263650368
0.1	0.745	0.51577614	0.57333333	0.53974575	0.82172829
0.2	0.755	0.3775	0.5	0.43015873	1.0663889

Best Architecture - Architecture 2: [22, 16, 2]
Lambda : 0
Best Architecture because of high F1 Score and Accuracy



Graph Interpretation

The cost function on the test set is decreasing as we increase the data we feed to the neural network. This is expected because, the more we feed the neural network, the better it performs on the test set, thus reducing the cost function.

Theory and Observations:

We used alpha = 1 for the dataset. When we trained with alpha = 10, the cost function was overshooting, and when we trained with alpha = 0.1, the cost function was decreasing by a very slight value and taking more iterations to train. Hence, we chose alpha = 1 in which the learning was reasonably fast and did not cause any overshooting.

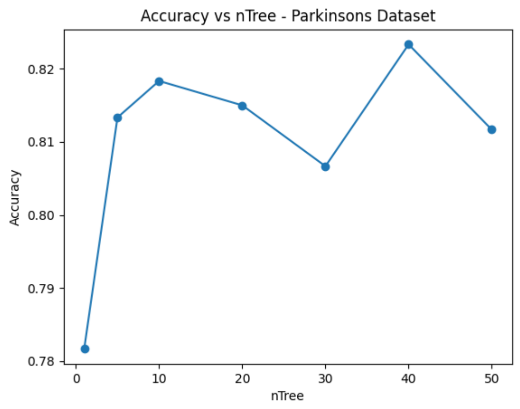
We chose 6 architectures for all the datasets by increasing the number of neurons and number of layers gradually.

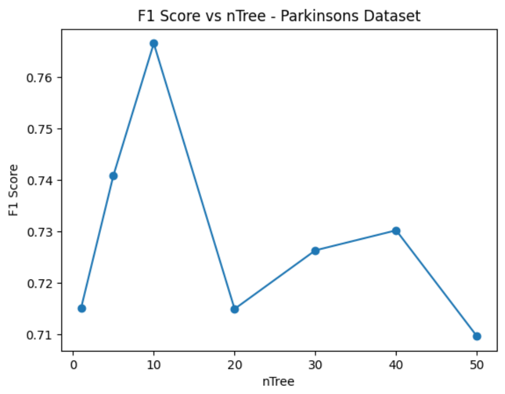
Regularization parameters are used to reduce the overfitting of complex models. With reasonable values of lambda and reasonable increase in those values, we should ideally see an increase in the performance for complex models.

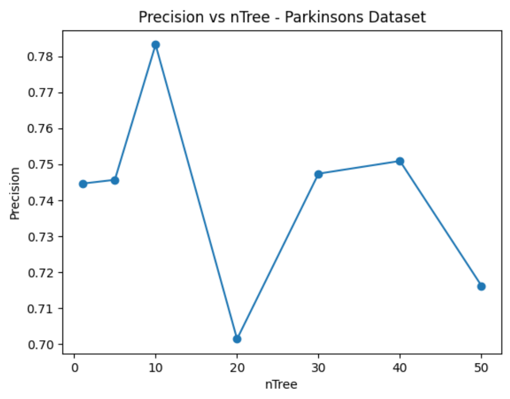
Neural networks with more layers and a greater number of neurons in each layer are expected to perform well because they take complex patterns from the dataset and give us better results. If we make our neural network more complex, it will overfit, and the accuracy drops. So, we must choose a reasonable number of layers and neurons to train our model.

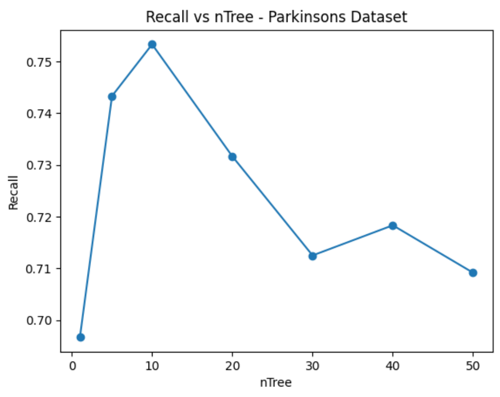
We notice that the neural network performs better for simpler architectures as compared to more complex architectures. This could be because of overfitting. Usually, we should see an increase in performance for complex networks when we increase the value of lambda. But here, we are seeing the reverse. The performance is decreasing when we increase regularization constant. This means all our models are mostly simple and we are penalizing the model by adding more regularization constant.

2. Random Forest









nTree	Accuracy	Precision	Recall	F1 Score
1	0.78166667	0.74463825	0.69666667	0.7150586
5	0.81333333	0.7456975	0.74333333	0.74087486
10	0.81833333	0.78323596	0.75333333	0.76659913
20	0.815	0.70146907	0.73166667	0.71492777
30	0.80666667	0.74740196	0.7125	0.72633225
40	0.82333333	0.75092188	0.71833333	0.73022109
50	0.81166667	0.71628246	0.70916667	0.70960127

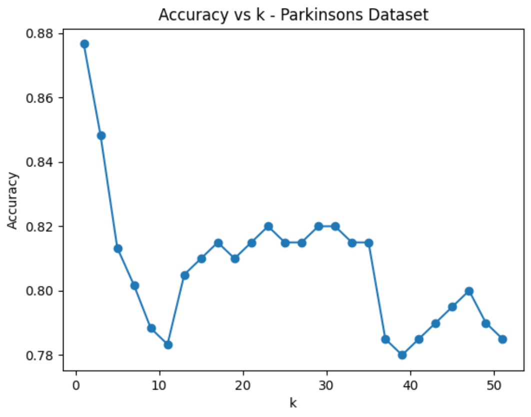
Best hyperparameter - 10
Accuracy - 0.81833333, F1 Score - 0.76659913 (Best F1 Score and accuracy is also high)

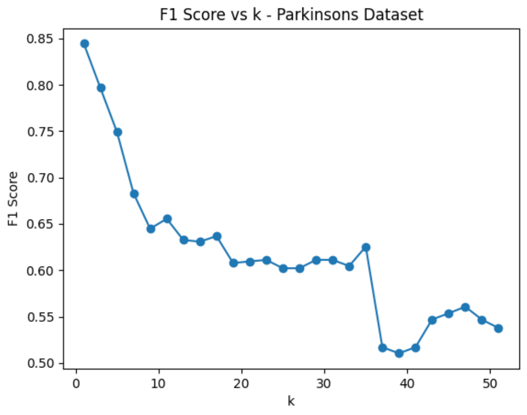
Graph Interpretation

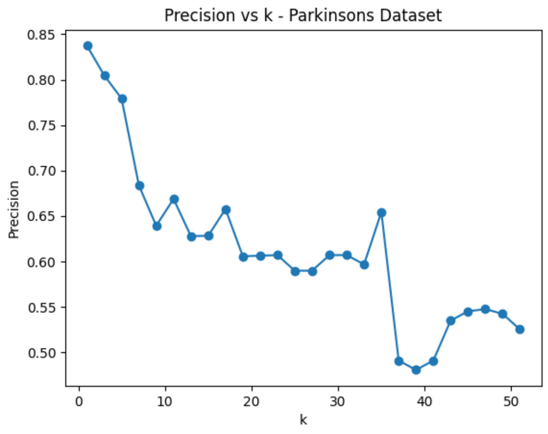
Here, we notice that the F1 Score is maximum at ntree = 10 and the accuracy is also good at this value of ntree. The accuracy is maximum at ntree = 40, but there is very little difference in accuracies when compared between ntree = 10 and ntree = 40. Hence, we chose ntree = 10 as the best hyperparameter for this dataset. We notice sudden dips at ntree = 20 for F1 Score and ntree = 30 for accuracy. This could be because -
1. We are not training 50 trees for each fold and taking predictions from nTree trees for each nTree value. I am training a new set of nTree trees every time with new bootstrap datasets. Hence the graphs would be little random and we can see low accuracies even for higher nTrees values. Example we see a dip at nTree = 40
2. Also since we are randomly creating bootstrap datasets we can assume that few trees might be fed with really easy data which results in less accuracy.

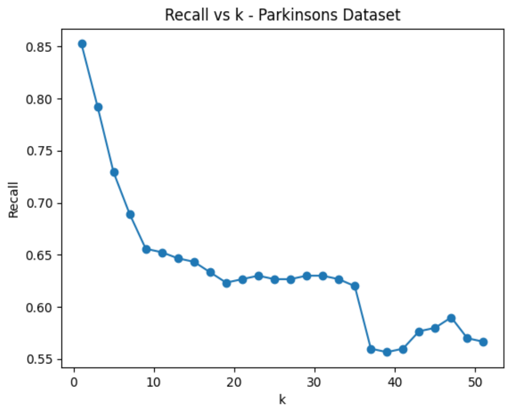
3. We are randomly choosing features at each node

3. K Nearest Neighbours









k	Accuracy	Precision	Recall	F1 Score
1	0.87666667	0.83729167	0.8525	0.84431509
23	0.82	0.60703431	0.63	0.61122016
31	0.82	0.60703431	0.63	0.61122016

Best value of k = 1
(Tested for k =1 to k = 51, for all odd values of k)

Graph Interpretation

From the graphs we can infer that the accuracy is maximum for a very low value of k = 1. The accuracy and the F1 Score is maximum at this value of k. The accuracies and the F1 Scores keep decreasing as the values of k keeps increasing.

<u>Telecom Customers Dataset:</u>

1. Neural Networks

(alpha = 4)

Architecture 1: [27, 8, 4]

Lambda	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**Converged J**
0	0.36154015	0.35780158	0.35517998	0.35597706	1.94943448
0.1	0.37093509	0.3482191	0.35832707	0.35173142	2.0541605
0.2	0.37378438	0.37832283	0.36404749	0.36632616	2.08545025

Architecture 2: [27, 16, 4]

Lambda	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**Converged J**
0	0.34470847	0.34387425	0.34070678	0.34208996	1.8132334
0.1	0.36190319	0.33748777	0.35423936	0.34283665	2.08512136
0.2	0.34460946	0.3516817	0.34197944	0.34062111	2.14263246

Architecture 3: [27, 16, 8, 4]

Lambda	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**Converged J**
0	0.33345435	0.320225	0.33027089	0.31522672	2.080689
0.1	0.30462046	0.28942074	0.3040458	0.29318625	2.16252459

0.2	0.31280528	0.22678574	0.30229197	0.25124463	2.2245697

Architecture 4: [27, 16, 16, 4]

Lambda	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**Converged J**
0	0.32973597	0.31567494	0.32549776	0.31790068	1.9924157
0.1	0.35176018	0.3516163	0.34328371	0.34579167	2.11504307
0.2	0.32418042	0.18231529	0.31472433	0.22482673	2.2476402

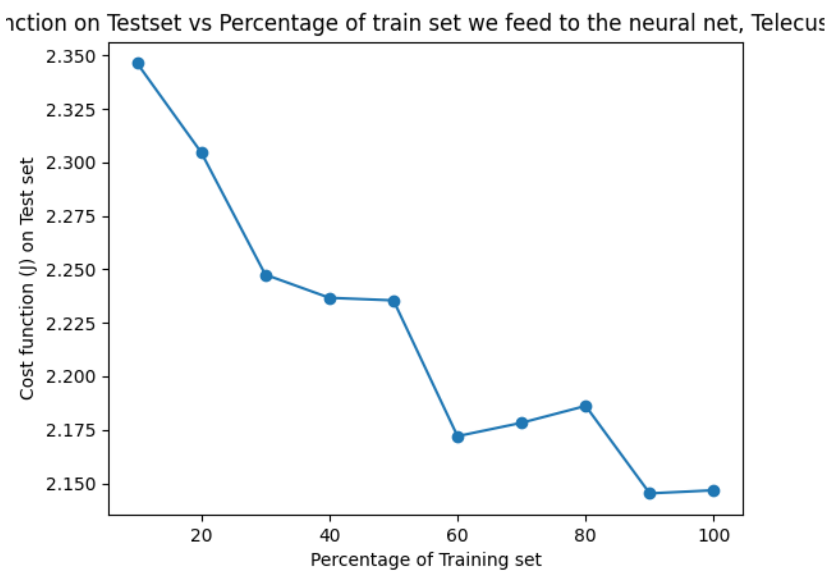
Architecture 5: [27, 16, 8, 4, 4]

Lambda	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**Converged J**
0	0.34343234	0.26238225	0.33235141	0.28303424	2.1774786
0.1	0.31526953	0.24281788	0.30846676	0.26804586	2.18322668
0.2	0.30573157	0.11872451	0.2832787	0.1618478	2.2376134

Architecture 6: [27, 16, 8, 8, 4]

Lambda	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**Converged J**
0	0.29932893	0.21324552	0.29280153	0.22982627	2.11922597
0.1	0.31727173	0.19348779	0.30927279	0.22806236	2.210037
0.2	0.28358636	0.08182053	0.2578869	0.12223355	2.2511004

Best Architecture : Architecture 1: [27, 8, 4]
Lambda - 0.2



Graph Interpretation:

The cost function on the test set is decreasing as we increase the data we feed to the neural network. This is expected because, the more we feed the neural network, the better it performs on the test set, thus reducing the cost function.

Theory and Observations:

We used alpha = 4 for the dataset. When we trained with alpha = 10, the cost function was overshooting, and when we trained with alpha = 1, the cost function was decreasing by a very slight value and taking more iterations to train. Hence, we chose alpha = 4 in which the learning was reasonably fast and did not cause any overshooting.

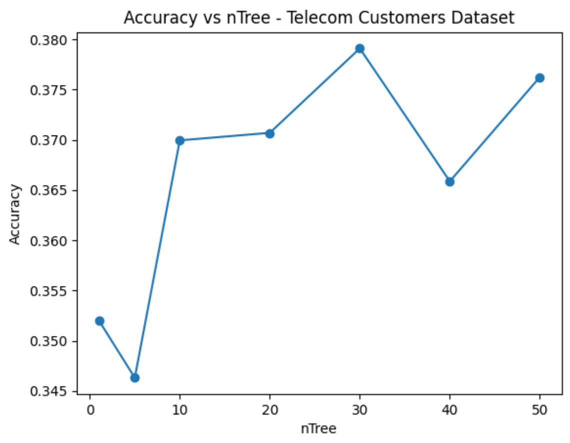
We chose 6 architectures for all the datasets by increasing the number of neurons and number of layers gradually.

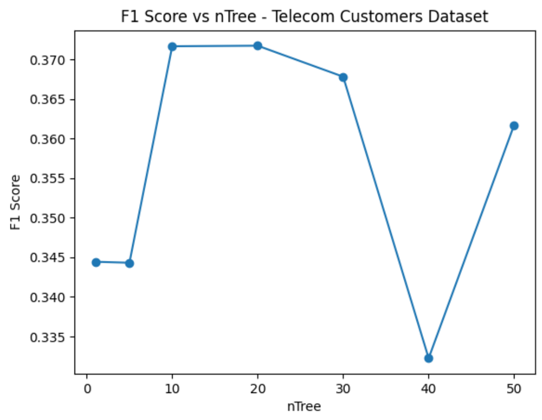
Regularization parameters are used to reduce the overfitting of complex models. With reasonable values of lambda and reasonable increase in those values, we should ideally see an increase in the performance for complex models.

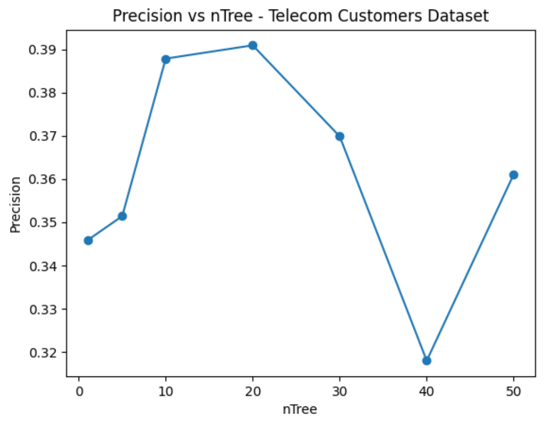
Neural networks with more layers and a greater number of neurons in each layer are expected to perform well because they take complex patterns from the dataset and give us better results. If we make our neural network more complex, it will overfit, and the accuracy drops. So, we must choose a reasonable number of layers and neurons to train our model.

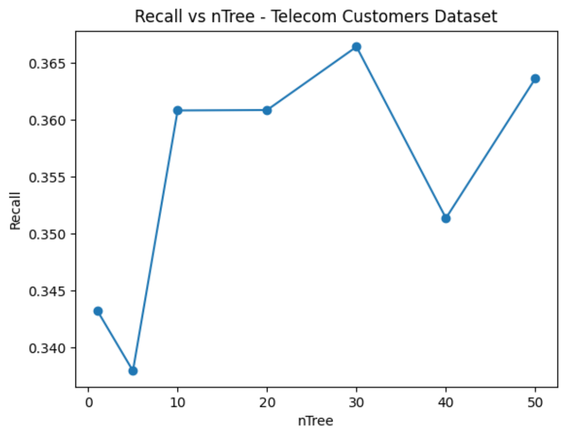
We notice that the neural network performs better for simpler architectures as compared to more complex architectures. This could be because of overfitting. Usually, we should see an increase in performance for complex networks when we increase the value of lambda. But here, we are seeing the reverse. The performance is decreasing when we increase regularization constant. This means all our models are mostly simple and we are penalizing the model by adding more regularization constant.

2. Random Forests









nTree	Accuracy	Precision	Recall	F1 Score
1	0.3520242	0.34583296	0.34327558	0.3444485

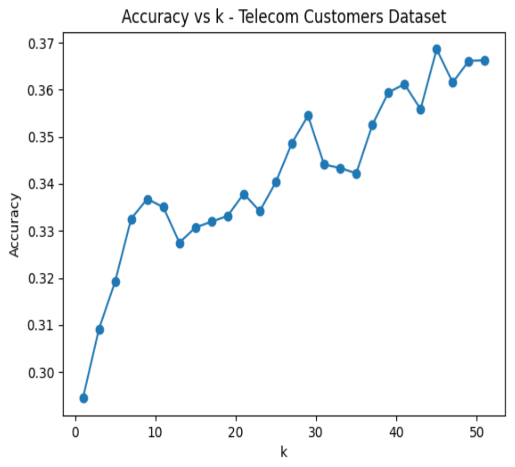
5	0.34630363	0.35141524	0.3379594	0.34432304
10	0.36994499	0.38783419	0.36084114	0.3716831
20	0.37069307	0.39090681	0.36087918	0.37175126
30	0.37909791	0.36984984	0.36644032	0.36783527
40	0.36586359	0.31799681	0.35138033	0.33225309
50	0.37624862	0.36108988	0.36365592	0.36174328

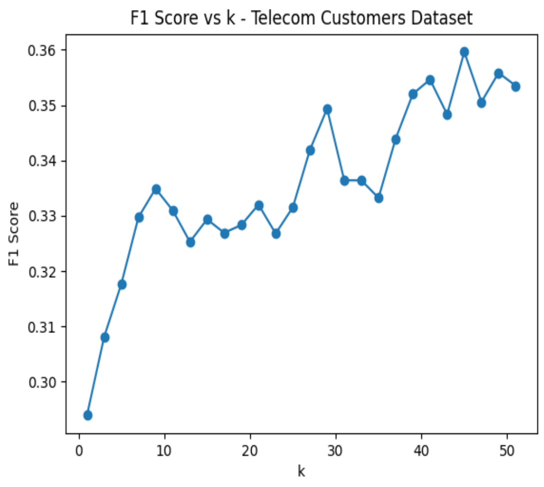
Best hyperparameter - 30
Accuracy - 0.37909791, F1 Score - 0.36783527 (Good Accuracy and F1 Score)

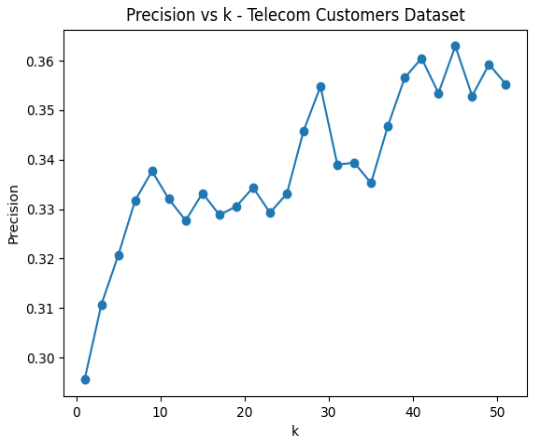
Graph Interpretation

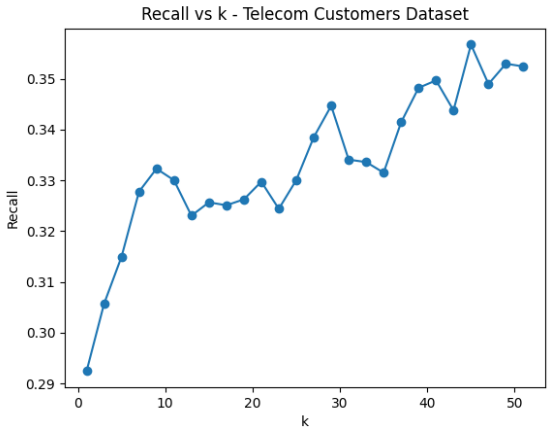
From the graph, we can see that the model performs best at ntree = 30. The performance is best at this value because the accuracy is maximum at this value of ntree, while the F1 Score is also good at this value of ntree. We notice sudden dips at ntree = 40 because -
1. We are not training 50 trees for each fold and taking predictions from nTree trees for each nTree value. I am training a new set of nTree trees every time with new bootstrap datasets. Hence the graphs would be little random and we can see low accuracies even for higher nTrees values. Example we see a dip at nTree = 40
2. Also since we are randomly creating bootstrap datasets we can assume that few trees might be fed with really easy data which results in less accuracy.
3. We are randomly choosing features at each node

3. K Nearest Neighbours









k	Accuracy	Precision	Recall	F1 Score
9	0.33678768	0.33768752	0.33230957	0.33489527
29	0.35448845	0.35480399	0.34468896	0.34933301
45	0.36859186	0.3629485	0.35675346	0.35959486

Best Value of k = 45
(Tested k from k =1 to k=51, for all odd values of k)

Graph Interpretation

From the graphs we can see that the model performs best at k = 45. At this value of k, the accuracy and the F1 Score of the model is maximum.

Final Evaluation Table

	Digits		Titanic		Loan		Parkinson		Telecust	
	Accuracy	**F1 Score**	**Accuracy**	**F1 Score**	**Accuracy**	**F1 Score**	**Accuracy**	**F1 Score**	**Accuracy**	**F1 Score**
Neural Nets	0.94444086	0.94683978	0.82386364	0.81377661	0.77626812	0.72644249	0.88833333	0.84439695	0.37378438	0.36632616
Random Forests	0.92875965	0.93307197	0.82045455	0.8087192	0.80163043	0.7557609	0.81833333	0.76659913	0.37909791	0.36783527
K-NN	0.97757613	0.97858365	0.81931818	0.8091802	0.76992754	0.71081875	0.87666667	0.84431509	0.36859186	0.35959486

Primary Observations -

1. K-NN performs best for the Digits dataset - Best k value - 3
2. Neural Nets perform best for the Titanic dataset. - Architecture 5: [9, 8, 6, 4, 2] with Lambda : 0
3. Random forests perform best for Loan dataset. - Best hyperparameter - 50
4. Neural Nets perform best for the Parkinson dataset. - Architecture 2: [22, 16, 2] with Lambda : 0
5. Random Forests performs best for Telecom customer dataset.Best hyperparameter - 30

Other Observations:

1. While training neural networks, we observed a continuous decrease in the cost function, because we are updating the weights in such a way that the cost function decreases. We move the weights in the opposite direction of the gradient and arrive at a local minimum. Hence, the cost function converges as we keep training the neural network.
2. Neural networks perform best on the numerical data since they can draw the hidden patterns inside the numbers. Hence we see neural networks doing good on Digits and Parkinsons data.
3. Random forests perform better when there are more categorical features. We see random forests doing good on Loan and Telecom Customers data.