ML-Final Report

- Aberami Jayanthi Anbazhakan, Asritha Reddy Gade

1.Handwritten Digits Recognition Dataset

We have implemented the Decision Tree, Random Forest and k-NN algorithms on the Handwritten Digits Recognition Dataset.

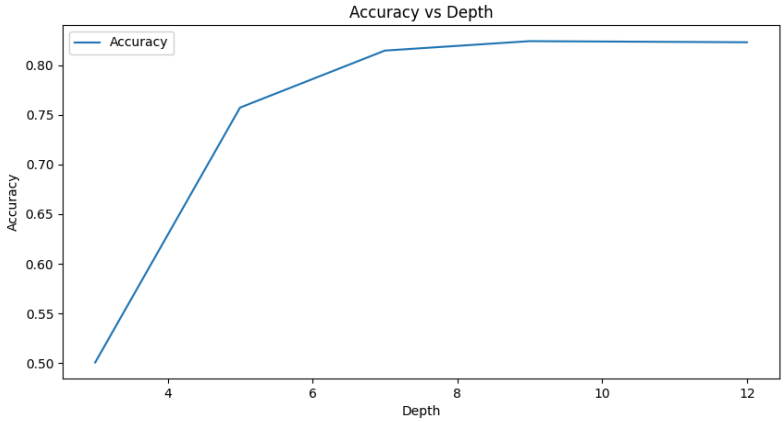
1.1 Decision Tree

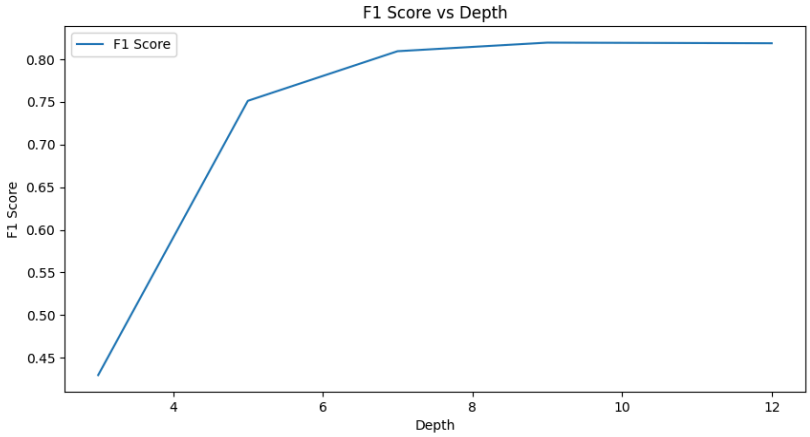
Decision Tree classifiers are capable of managing complex and non-linear relationships, which makes them a suitable choice for recognizing handwritten digits. Decision Tree's ability to handle numeric attributes without requiring extensive preprocessing makes it a more suitable choice because each instance in the dataset has 64 numerical attributes. Furthermore, since this dataset involves a multiclass output, Decision Trees are an appropriate option. We used Aberami's Decision Tree implementation throughout the project.

1.1.1 Experiment and Analysis

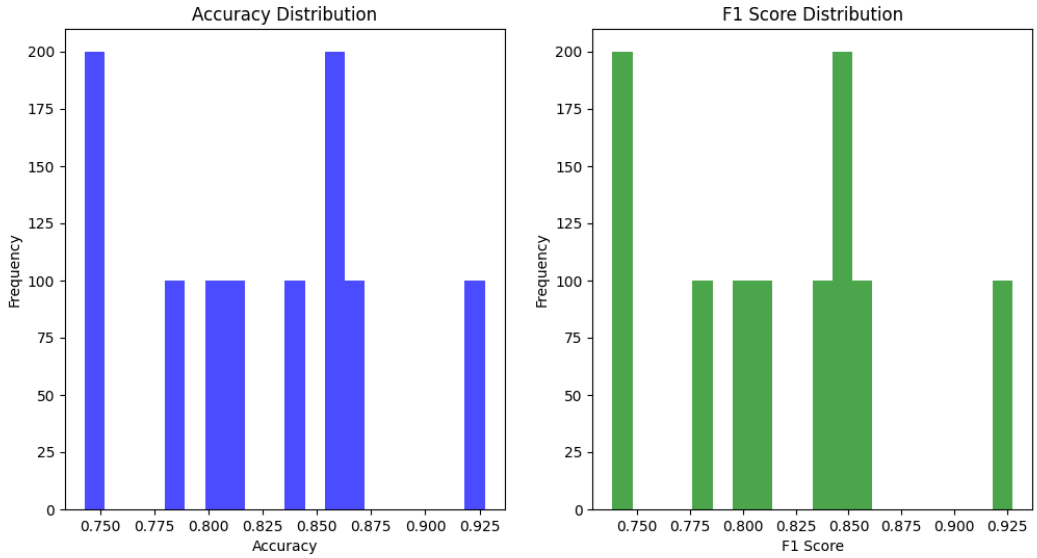
In our study, we used the 'depth of the tree' as the hyperparameter to be tested. We evaluated the model at various depths, including 3, 5, 7, 10, and 12. The model exhibited its highest accuracy (82%) and maximum F1 score (0.82) at a depth of 10.

Depth of Tree	**Accuracy**	**F1 Score**
3	0.5008	0.4297
5	0.7572	0.7514
7	0.8146	0.8096
10	***0.8241***	***0.8196***
12	0.8230	0.8189





After finding the optimal hyperparameter, we conducted 100 iterations using the Decision Tree model set to the depth of 10. The model consistently showed an average accuracy of 82% with a standard deviation of 0.0538, and an average F1 score of 82, also with a standard deviation of 0.0552.



Average Accuracy: 0.8235, Standard Deviation: 0.0538
Average F1 Score: 0.8191, Standard Deviation: 0.0552

1. 2 Random Forest

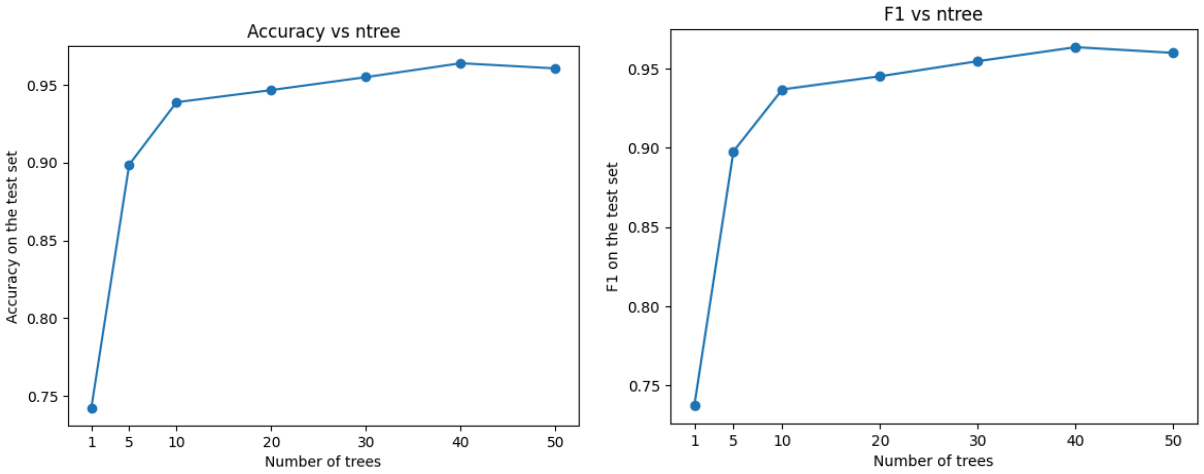
Random Forests are a good choice for handwritten digit recognition because the dataset involves high-dimensional data, as each pixel in an image represents a separate feature. Random Forests can effectively manage this high dimensionality by randomly selecting subsets of features at each split in the learning process. This randomness helps in building diverse trees and reduces the variance of the model, resulting in better performance. In addition, random Forests provide insights into which features (pixels in this case) are most important for making the classification decision. This can be useful for understanding which parts of the images are key for recognizing digits. We used Asritha's Random Forest implementation throughout the project.

1.2.1 Experiment and Analysis

In the experiments, we evaluated the performance of a Random Forest algorithm on classifying handwritten digits by altering the 'number of trees' hyperparameter.

The results from the experiment indicate a clear trend where both accuracy and F1 score improve as the number of trees in the Random Forest increases. The increments in performance are notable from 1 tree to 10 trees, suggesting that adding more trees generally enhances model performance. However, beyond this point, although there are slight fluctuations in accuracy, adding more trees does not significantly enhance performance. The highest accuracy was recorded at 40 trees, showing only a marginal improvement compared to the performance with 30 trees. After reaching 40 trees, the performance slightly declines at 50 trees. The optimal choice for the number of trees is 40 because it provides the highest accuracy and F1 score without significant trade-offs.

n_tree	Accuracy	F1 Score
1	0.7424	0.7375
5	0.8987	0.8978
10	0.9388	0.9368
20	0.9466	0.9451
30	0.9549	0.9547
40	***0.9638***	***0.9635***
50	0.9605	0.9599



1.3 k-Nearest Neighbors - EXTRA CREDIT

The k-NN algorithm is fundamentally simple and it can effectively classify a new digit by comparing it to known examples. Further, images' pixel data can often be compared effectively using simple distance metrics like Euclidean distance, which k-NN uses to determine the "closeness" of different images. In the context of handwritten digits, this means that images of the same digit are likely to have similar pixel patterns. Finally, the k-NN algorithm can be quite robust against noise in the data by averaging the labels of the k-nearest neighbors, the algorithm reduces the impact of outliers. We used Aberami's k-NN implementation throughout the project.

1.3.1 Experiments and Analysis

For this experiment, all the numeric attributes of the dataset were preprocessed using the 'min-max' normalization technique. The K-Nearest Neighbors (k-NN) classifier's performance

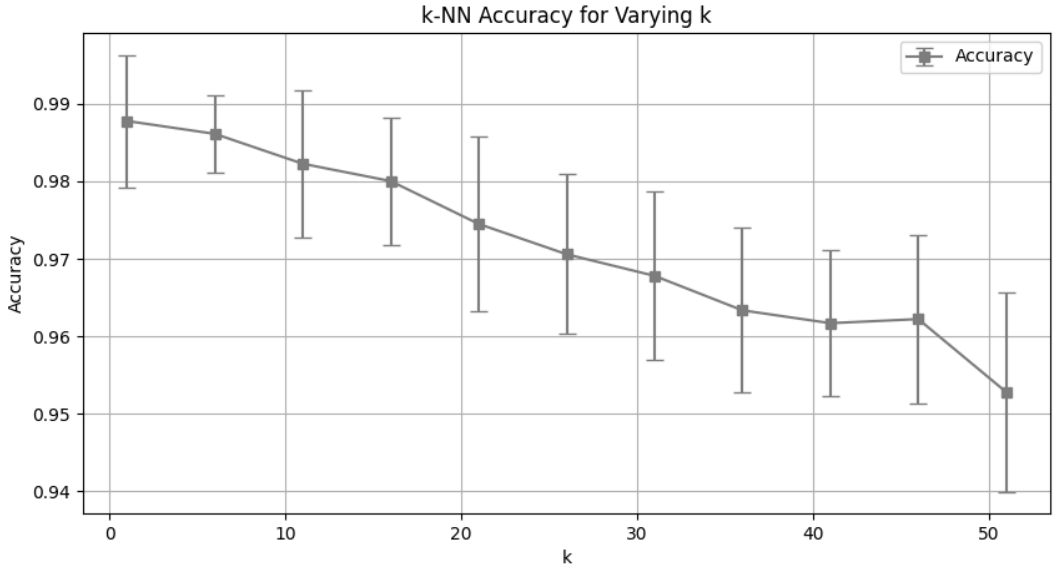
was tested based on different values of the hyperparameter 'k' ranging from 1 to 50 with the step size of 5.

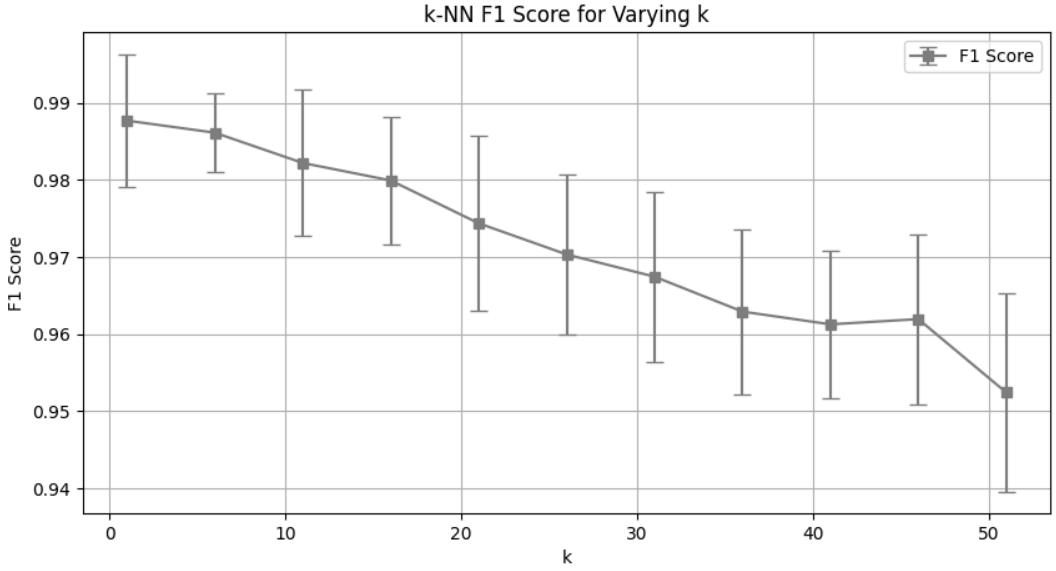
The results from the k-Nearest Neighbors (kNN) experiments on the digits dataset show a clear trend where both accuracy and F1 score generally decrease as the number of neighbors (k) increases. The highest performance is observed at k=1 with an accuracy of 95.92% and an F1 score of 94.51%.

In k-NN, a smaller *k* value means that the classification is more sensitive to noise in the dataset, but for our algorithm, *k*=1 achieves the best results for the dataset. This suggests that the closest single neighbor provides the most reliable prediction for this particular dataset. As *k* increases, the accuracy and F1 score generally decrease, indicating potential over-smoothing where the model starts considering more distant and potentially less relevant examples.

In a real-world implementation, choosing k=1 would be advisable based on this data. Despite the common concern of overfitting with a very small k in kNN, the high accuracy and F1 score at k=1 suggest that the model performs exceptionally well with a single nearest neighbor in this particular case. This setup seems to best capture the underlying patterns in the data without introducing noise or irrelevant information that higher values of k might include. Choosing k=1 ensures the highest prediction quality as indicated by the performance metrics.

k	**Accuracy**	**F1 Score**
1	**0.9878**	**0.9877**
6	0.9861	0.9862
11	0.9823	0.9822
16	0.9800	0.9800
21	0.9745	0.9744
26	0.9706	0.9704
31	0.9678	0.9675
36	0.9634	0.9629
41	0.9617	0.9613
46	0.9622	0.9620
51	0.9528	0.9525





The Titanic Dataset

We have implemented the Decision Tree, Neural Networks and k-NN algorithms on the Titanic Dataset.

2.1 Decision Tree

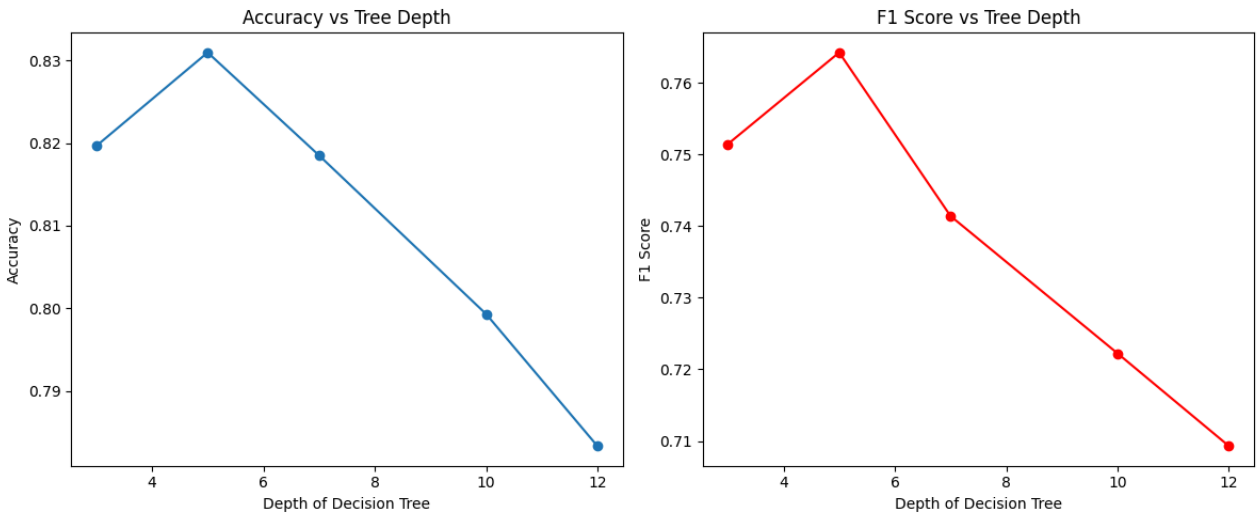
We chose the Decision Tree implementation to predict the chances of survival on the Titanic journey because the Decision Tree can handle both numerical (e.g., age, fare) and categorical attributes (e.g., gender, passenger class) without requiring extensive pre-processing. Further, Decision Trees can provide insights into which features are most important in predicting the outcome - for instance, it can help us figure out how features like 'Class', Gender, or 'Age' affect survival rates. We used Aberami's Decision Tree implementation to evaluate the Titanic Dataset.

2.1.1 Experiment and Analysis

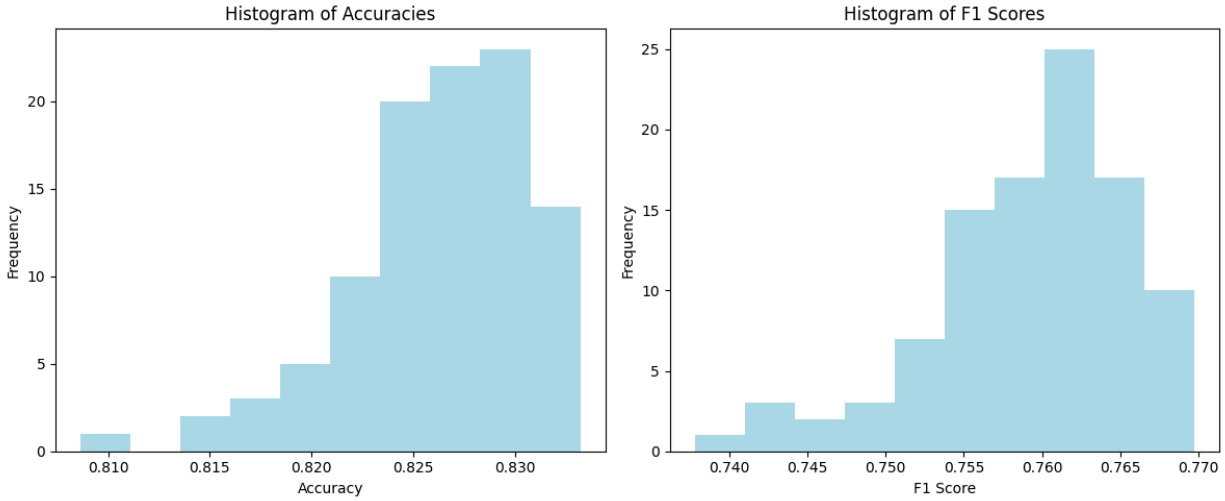
The Titanic dataset has a combination of numeric and categorical attributes. The numeric attributes such as 'Age', 'Siblings/Spouses Aboard', 'Parents/Children Aboard' and 'Fare' were normalized using min-max normalization technique. The other categorical attributes such as 'Gender' and 'Passenger class', were one-hot encoded into numeric values. Lastly, the attribute 'Name' of the passenger was dropped since it would be unique to each individual and would not provide any additional information for the tree to generalize. Depth of the tree is considered to be the hyperparameter and the performance of the Decision Tree classifier was evaluated for various tree depths [3, 5, 7, 10, 12].

Depth of Tree	**Accuracy**	**F1 Score**
3	0.820551696742708	0.719020663569137
5	**0.8319164680512996**	**0.7655642126547658**
7	0.8191797752808988	0.7417400195124079
10	0.7982005447735785	0.7228920793256368
12	0.7803109749177164	0.7099584369919975

Upon experimenting, it was observed that the evaluation metrics of the tree starts increasing and reaches a peak when depth is 5 and then decreases with increasing depth. It indicates that a reasonably complex model with a small depth is capturing the pattern in the data. The model reaches its peak performance of about 83% accuracy and 76% F1 score at depth of 5.



Further, at the depth of 5, two histograms were constructed using the model. The model was iterated 100 times to construct the accuracy and F1 score distributions.



Average Accuracy: 0.8261, Std Dev: 0.0044
Average F1 Score: 0.7593, Std Dev: 0.0063

2.2 k-NN

The k-Nearest Neighbors (k-NN) algorithm is well-suited for the Titanic survival prediction task due to its ability to handle both numeric and categorical data attributes through preprocessing. It also adapts to manage non-linear relationships without bias about data distributions. Its method of instance-based learning makes it robust to noise and effective in capturing the complex interactions of variables like age, gender, and class that determine survival outcomes. We used Aberami's k-NN implementation throughout the project.

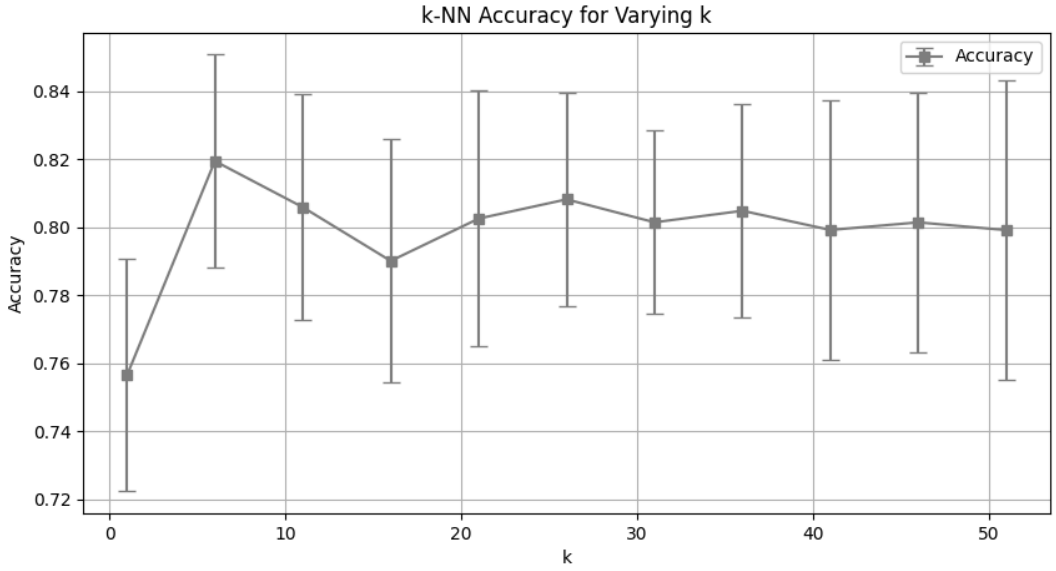
2.2.1 Experiment and Analysis

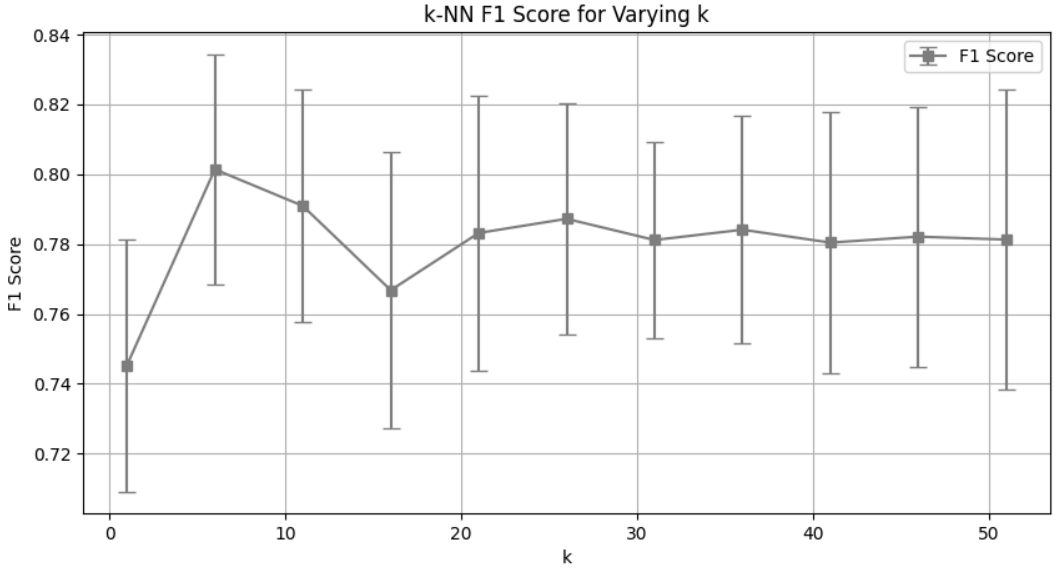
For this experiment, all the numeric attributes of the dataset were preprocessed using the ‘min-max’ normalization technique and the categorical attributes such as passenger class and gender were one-hot encoded to numeric values. The K-Nearest Neighbors (k-NN) classifier’s performance was tested based on different values of the hyperparameter ’k’ ranging from 1 to 50 with the step size of 5.

The kNN results on the Titanic dataset indicate a pattern where both accuracy and F1 score tend to peak at moderate values of k before plateauing or slightly decreasing as k continues to increase. The optimal performance is observed at k=6, with an accuracy of 81.95% and an F1 score of 80.14. Beyond this point, while the performance remains relatively stable, it does not surpass the peak achieved at k=6.

The optimal k value would be k=6 as it balances between avoiding overfitting and maintaining an effective predictive accuracy and F1 scores balance.

k	**Accuracy**	**F1 Score**
1	0.7590	0.7483
6	***0.8195***	***0.8014***
11	0.8059	0.7909
16	0.7901	0.7668
21	0.8026	0.7832
26	0.8082	0.7872
31	0.8015	0.7812
36	0.8049	0.7841
41	0.7992	0.7804
46	0.8015	0.7821
51	0.7992	0.7813





2.3 Neural Networks - EXTRA CREDIT

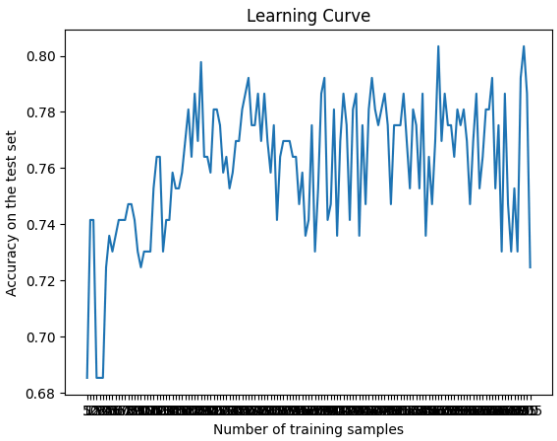
Neural networks can learn these complex relationships directly from the data, providing a potentially higher predictive accuracy. They further have the inherent ability to perform feature learning and find the most relevant features for making predictions during the training process. This is particularly useful in a dataset like the Titanic, where the importance and interaction of features may not be immediately apparent. We used Asritha's Neural Network implementation throughout the project.

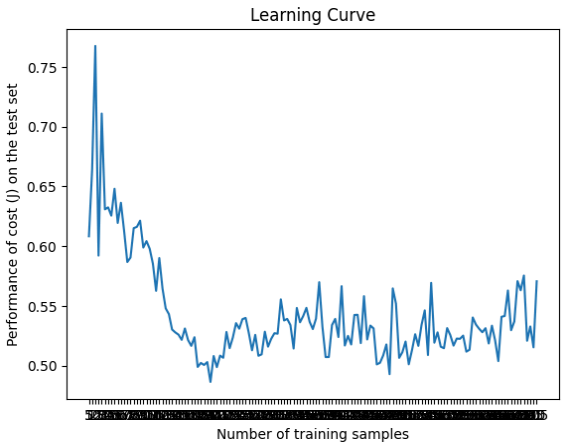
2.1.1 Experiment and Analysis

For our study, we selected the ‘hidden layers’ as the hyperparameter to be tested. From the tables below, we can see that the best neural network architecture was the one with the 12 neurons in the hidden layer with an accuracy of 81% and a F1 score of 0.80.

Architecture	**Average Accuracy**	**Average F1 Score**
epochs=1000, learning_rate=0.01, regularization_param=0.001, layers=[input,4,3, output]	0.8038213596640562	0.7869080585100696
epochs=1000, learning_rate=0.01, regularization_param=0.001, layers=[input,6,7,4,output]	0.7936060038588129	0.7759326455148654
epochs=1000, learning_rate=0.01, regularization_param=0.001, layers=[input,12,output]	**0.8161689933038246**	**0.8014111197665693**

After identifying the best neural network architecture, it was used to generate a learning curve where the y axis shows the network’s performance (J) on a test set, and where the x axis indicates the number of training samples given to the network in the step size of 5. It can be observed from the graph that the network’s performance improves as the number of training examples grows and the cost function, J decreases with the number of training instances presented to the network increases.





3. Loan Eligibility Prediction Dataset

We have implemented the Decision Tree, Neural Networks and k-NN algorithms on the Loan Eligibility Prediction Dataset.

3.1 Decision Tree

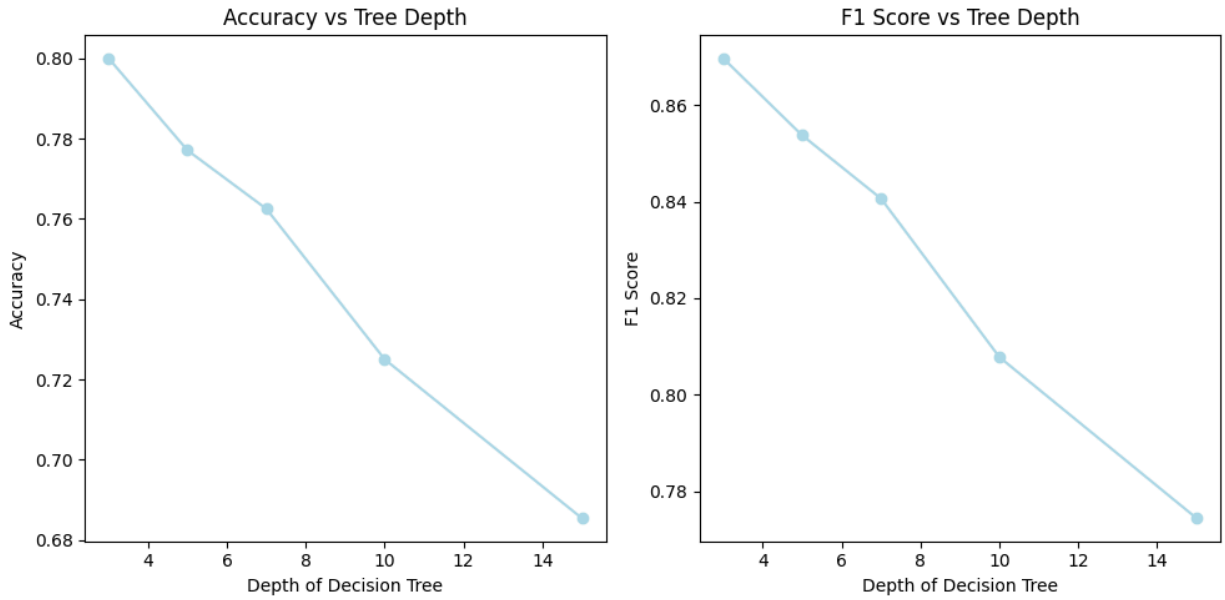
We chose the Decision Tree implementation to predict whether a given person should qualify for a loan because the Decision Tree can handle both numerical (e.g., income, credit history) and categorical attributes (e.g., gender, marital status) without requiring extensive pre-processing. Further, Decision Trees can provide insights into which features are most important in predicting the outcome - for instance, it can help us figure out how features like 'Income', Gender, or 'Age' affect approval rates. We used Aberami's Decision Tree implementation to evaluate the Titanic Dataset.

3.1.1 Experiment and Analysis

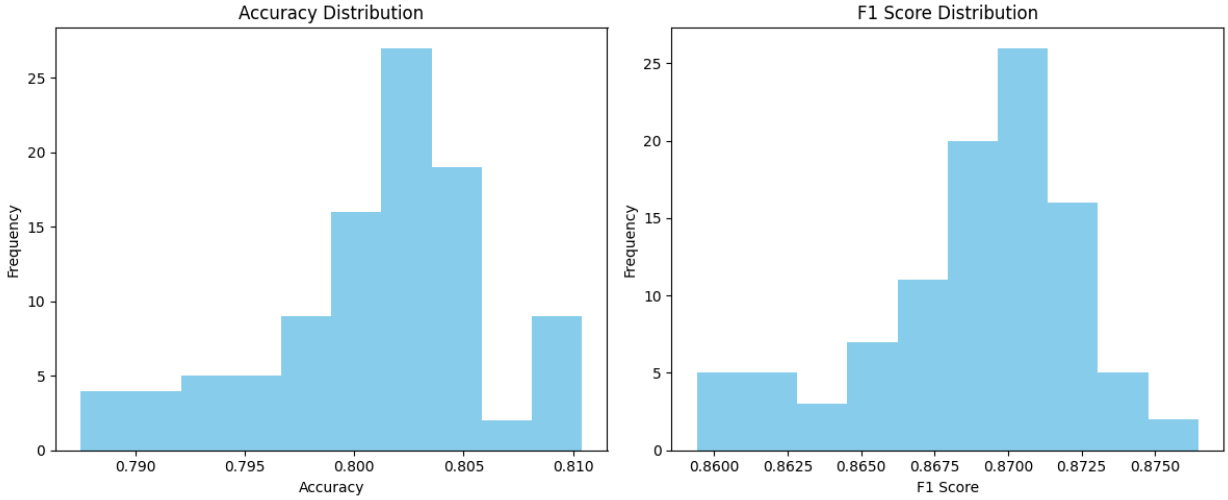
The Loan Eligibility dataset has a combination of numeric and categorical attributes. The numeric attributes were normalized using min-max normalization technique. The other categorical attributes such as 'Gender'', were one-hot encoded into numeric values. Lastly, the attribute 'Loan_ID' of the applicant was dropped since it would be unique to each individual and would not provide any additional information for the tree to generalize. Depth of the tree is taken as the hyperparameter and the performance of the Decision Tree classifier was evaluated for various tree depths [3, 5, 7, 9, 12].

Depth of Tree	**Accuracy**	**F1 Score**
3	0.800000	0.869678
5	**0.777083**	**0.853730**
7	0.762500	0.840627
9	0.739583	0.820399
12	0.710417	0.795910

Even though we got the best accuracy and f1 scores for depth of 3, we consider taking 5 trees to be a good number to avoid any underfitting.



Further, at the depth of 5, two histograms were constructed using the model. The model was iterated 100 times to construct the accuracy and F1 score distributions.



Summary Statistics for Accuracy:
Mean Accuracy: 0.8008, Standard Deviation: 0.0047
Summary Statistics for F1 Score:
Mean F1 Score: 0.8688, Standard Deviation: 0.0036

3.2 Neural Network

Neural networks can learn these complex relationships directly from the data, providing a potentially higher predictive accuracy. They further have the inherent ability to perform feature learning and find the most relevant features for making predictions during the training process.

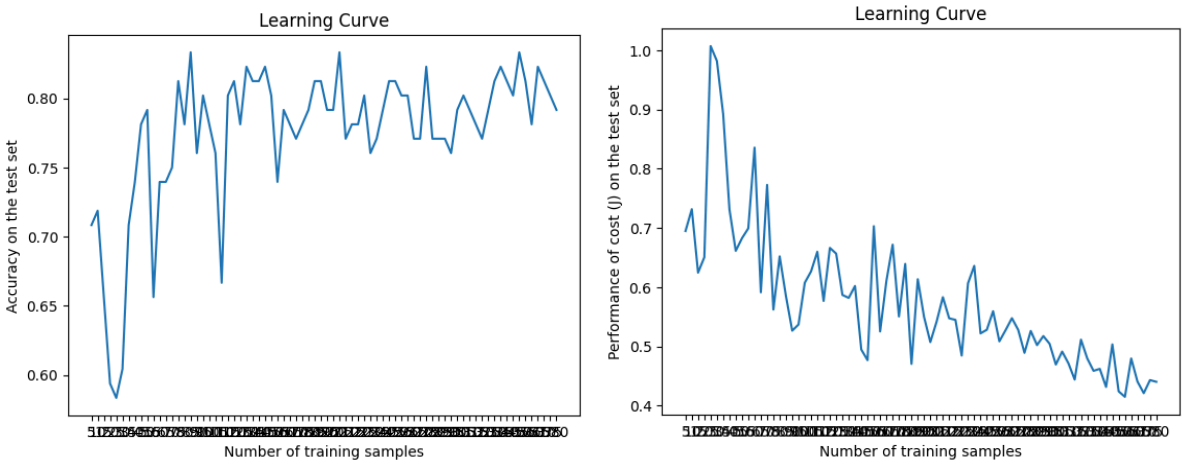
This is particularly useful in a dataset like the Loan Eligibility Prediction, where the importance and interaction of features may not be immediately apparent. We used Asritha's Neural Network implementation throughout the project.

3.2.2 Experiment and Analysis

For our study, we selected the 'hidden layers' as the hyperparameter to be tested. From the tables below, we can see that the best neural network architecture was the one with the 8 hidden neurons in the first layer, 12 neurons in the second hidden layer and 15 neurons in the third hidden layer with an accuracy of 78.75% and a F1 score of 0.71.

Architecture	Accuracy	F1 Score
epochs=1000, learning_rate=0.01, regularization_param=0.001, layers=[input,4,3, output]	0.7729166666666666	0.706923938013128
epochs=1000, learning_rate=0.01, regularization_param=0.001, layers=[input,8,12,15,output]	**0.7875**	**0.7155337422616024**
epochs=1000, learning_rate=0.01, regularization_param=0.001, layers=[input,9,output]	0.7395833333333334	0.6628068876326324

After identifying the best neural network architecture, it was used to generate a learning curve where the y axis shows the network's performance (J) on a test set, and where the x axis indicates the number of training samples given to the network in the step size of 5. It can be observed from the graph that the network's performance improves as the number of training examples grows and the cost function, J decreases with the number of training instances presented to the network increases. It may further be noted that the accuracy of the model's predictions also increases as more training instances are introduced.



3.3 k-Nearest Neighbors - EXTRA CREDIT

The k-Nearest Neighbors (k-NN) algorithm is well-suited for the Loan Eligibility Prediction task due to its ability to handle both numeric and categorical data attributes through preprocessing. It also adapts to manage non-linear relationships without bias about data distributions. Its method of instance-based learning makes it robust to noise and effective in capturing the complex interactions of variables like age, gender, and income that determine approval outcomes. We used Aberami's k-NN implementation throughout the project.

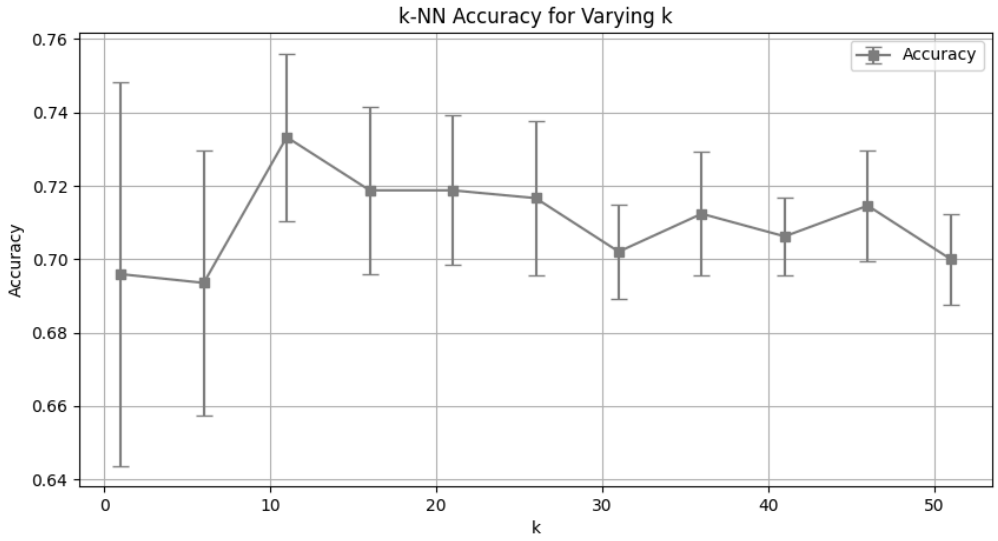
3.3.1 Experiment and Analysis

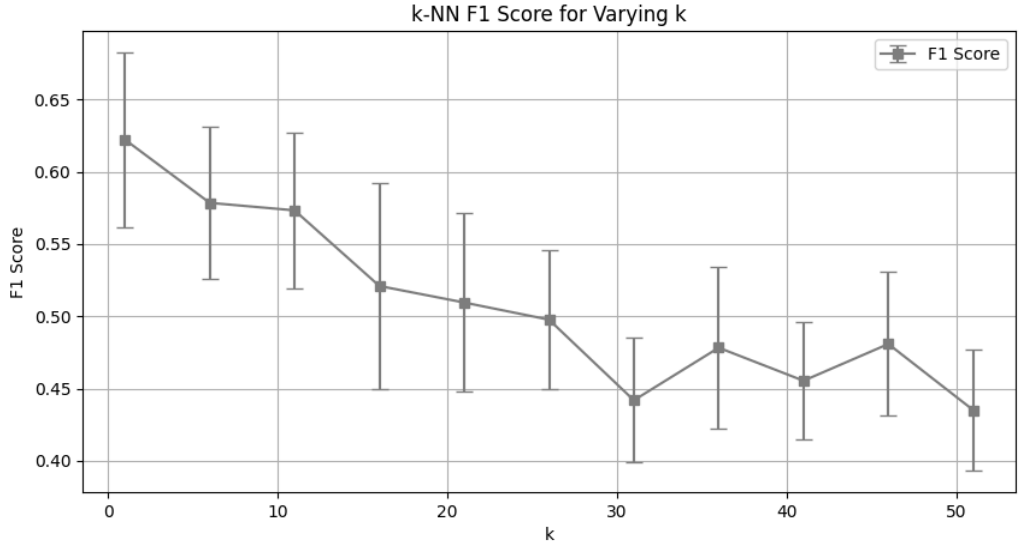
For this experiment, all the numeric attributes of the dataset were preprocessed using the 'min-max' normalization technique and the categorical attributes such as marital status and gender were one-hot encoded to numeric values. The K-Nearest Neighbors (k-NN) classifier's performance was tested based on different values of the hyperparameter 'k' ranging from 1 to 50 with the step size of 5.

The kNN results on the loan eligibility prediction dataset indicate a pattern where both accuracy and F1 score tend to peak at moderate values of k before plateauing or slightly decreasing as k continues to increase. The optimal performance is observed at k=11, with an accuracy of 73% and an F1 score of 0.57. Beyond this point, while the performance remains relatively stable, it does not surpass the peak achieved at k=11.
The optimal k value would be k=11 as it balances between avoiding overfitting and maintaining an effective predictive accuracy and F1 scores balance.

k	**Accuracy**	**F1 Score**
1	0.6960	0.6223
6	0.6936	0.5785
11	**0.7333**	**0.5733**

16	0.7188	0.5210
21	0.7188	0.5096
26	0.7167	0.4977
31	0.7021	0.4419
36	0.7124	0.4783
41	0.7062	0.4555
46	0.7146	0.4809
51	0.7000	0.4352





4. The Oxford Parkinson's Disease Detection Dataset

We have implemented the Decision Tree, Random Forest and k-NN algorithms on the Oxford Parkinson's Disease Detection Dataset.

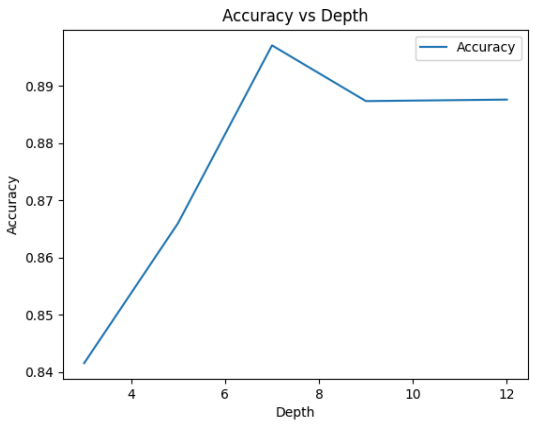
4.1 Decision Tree

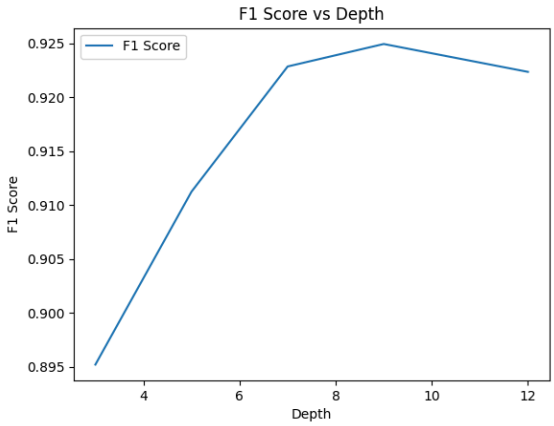
Decision Tree classifiers are capable of managing complex and non-linear relationships, which makes them a suitable choice for predicting patients with Parkinson's disease. Decision Tree's ability to handle numeric attributes without requiring extensive preprocessing makes it a more suitable choice because the dataset is composed of numerical attributes. We used Aberami's Decision Tree implementation throughout the project.

4.1.1 Experiment and Analysis

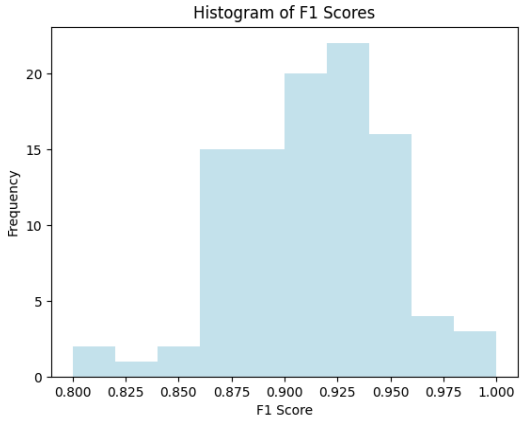
In our study, we used the 'depth of the tree' as the hyperparameter to be tested. We evaluated the model at various depths, including 3, 5, 7, 9, and 12. The model exhibited its highest accuracy (89.71%) and maximum F1 score (0.9229) at a depth of 7.

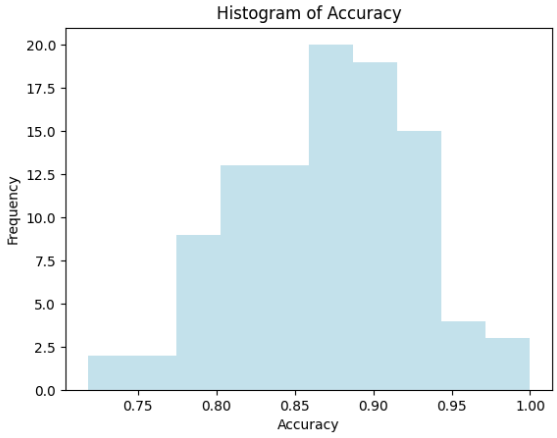
Depth of Tree	**Accuracy**	**F1 Score**
3	0.8416	0.8952
5	0.8661	0.9112
7	**0.8971**	**0.9229**
9	0.8874	0.9249
12	0.8876	0.9224





After finding the optimal hyperparameter, we conducted 100 iterations using the Decision Tree model set to the depth of 10. The model consistently showed an average accuracy of 86.9% with a standard deviation of 0.0519, and an average F1 score of 0.9123, also with a standard deviation of 0.0366.





Average Accuracy: 0.8690, Standard Deviation: 0.0519
Average F1 Score: 0.9123, Standard Deviation: 0.0366

4.2 Random Forest

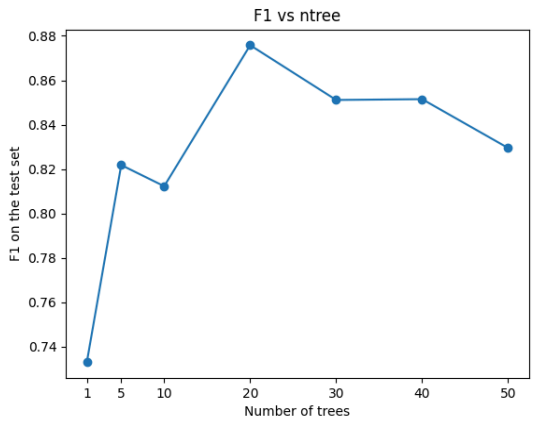
Random Forests is an effective choice for the Oxford Parkinson's Disease Detection dataset. Random Forests are ensemble methods that are robust to overfitting, a typical issue in medical datasets with only a limited number of samples, as in this case around 200 instances. Also, Random Forests can handle both numerical and categorical variables and are adept at modeling complex, non-linear decision boundaries, potentially crucial in separating Parkinson's patients from healthy individuals. We used Asritha's Random Forest classifier throughout the project.

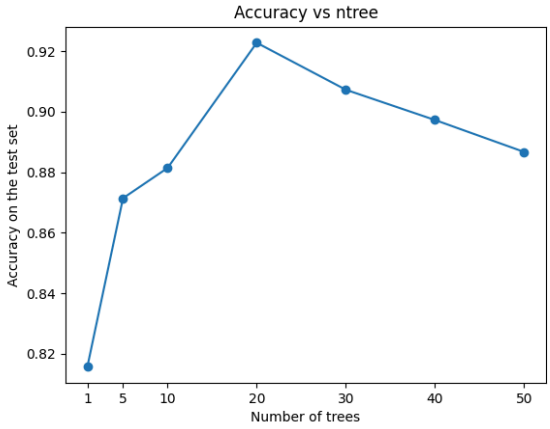
4.2.1 Experiments and Analyses

For our experiments, the performance of a Random Forest classifier on predicting whether an individual is a patient or not was tested with different numbers of trees.
From the results, we can observe that as the number of trees increases from 1 to 20, both the accuracy and F1-score improve. This improvement suggests that a more complex model, with more trees, captures the data's complexities better.

The highest accuracy (92.28%) and F1-score (0.8758) are achieved when the number of trees is 20. Interestingly, increasing the number of trees beyond 20 doesn't improve the model's performance. Instead, we observe a decline in both metrics, indicating that overly complex models might not generalize well to unseen data.

ntree	Accuracy	F1 Score
1	0.8157	0.7329
5	0.8714	0.8218
10	0.8814	0.8122
20	**0.9228**	**0.8758**
30	0.9073	0.8511
40	0.8973	0.8515
50	0.8867	0.8295





4.3 k-Nearest Neighbors - EXTRA CREDIT

k-NN is straightforward to implement, making it a valuable tool in medical applications where clarity and simplicity in model operations and decisions are essential. k-NN is effective with

smaller datasets as in the case with the Parkinson's dataset because it does not make any assumptions about the underlying data distribution. We used Aberami's k-NN implementation throughout the project.

4.3.1 Experiment and Analysis

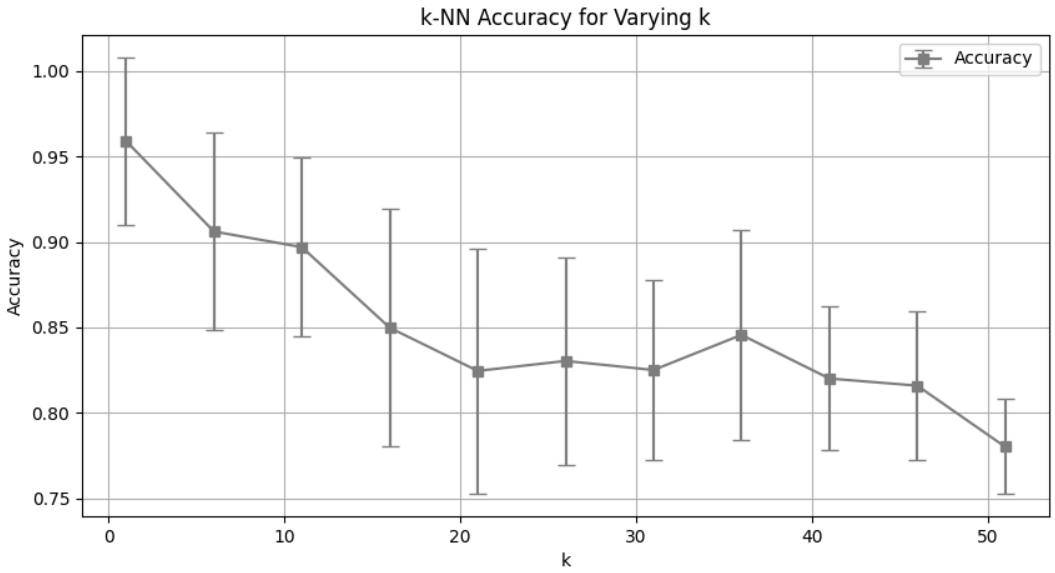
For this experiment, all the numeric attributes of the dataset were preprocessed using the 'min-max' normalization technique. The K-Nearest Neighbors (k-NN) classifier's performance was tested based on different values of the hyperparameter 'k' ranging from 1 to 50 with the step size of 5.

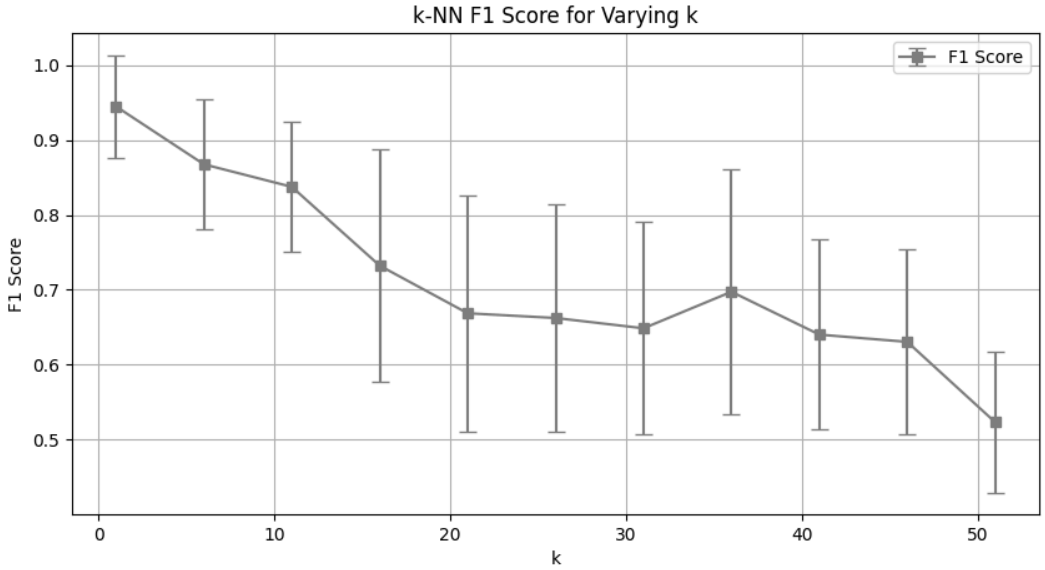
The results from the k-Nearest Neighbors (kNN) experiments on the digits dataset show a trend where both accuracy and F1 score generally decrease as the number of neighbors (k) increases. **At k=1**, the model achieves the highest accuracy (95.92%) and F1 score (94.51%), indicating good performance with the most direct neighbor comparisons. **As k increases** (from 6 to 51), there is a consistent decline in both accuracy and F1 score. This drop is more pronounced after k=11, suggesting that incorporating more neighbors leads to the model considering more noise in the data, thereby reducing its predictive accuracy and precision-recall balance

This pattern illustrates the typical behavior of kNN, where a small k value tends to produce higher accuracy and F1 scores by focusing on the nearest or most similar instances, which are often more predictive. Larger k values, while potentially offering more stability against noise in the data, can dilute the effect of the most relevant instances by averaging over a broader range of less similar neighbors. In a scenario requiring high predictive performance with a focus on both accuracy and maintaining a strong balance between precision and recall, a smaller k value, specifically k=1, appears most effective based on this dataset. This choice would be optimal for scenarios where the highest level of detail and sensitivity in predictions is crucial, even at the risk of overfitting, which is a lesser concern given the high performance at k=1.

k	**Accuracy**	**F1 Score**
1	**0.9592**	**0.9451**
6	0.9061	0.8677
11	0.8970	0.8376
16	0.8498	0.7323
21	0.8245	0.6687
26	0.8304	0.6624
31	0.8251	0.6487
36	0.8456	0.6974
41	0.8201	0.6402

46	0.8159	0.6305
51	0.7801	0.5231





EXTRA CREDIT

5. UCI’s Adult Income Census Dataset

The dataset we used for extra credit is the UCI "Census Income" dataset, also known as the "Adult" dataset. This dataset was extracted from the 1994 Census database by Ronny Kohavi and Barry Becker. It consists of over 48,000 instances, each described by a set of 14 attributes, including age, work class, education, marital status, occupation, relationship, race, sex, capital gain.

The primary task associated with this dataset is to predict whether an individual earns more than $50,000 per year based on the census attributes, making it a binary classification problem. The target attribute for the dataset is income, which is categorized into two classes: ">50K" and "<=50K".

We have implemented the Random Forest, Neural Network and k-NN algorithms on the UCI’s Adult Income Census Dataset.

5.1 KNN

k-NN is a straightforward algorithm that predicts the class of a data point based on the majority vote of its nearest neighbors. It's non-parametric, meaning it makes no underlying assumptions about the distribution of data, which can be particularly useful in real-world scenarios where such assumptions might fail. Since k-NN uses distance metrics to find the nearest neighbors, all features must be scaled to ensure that no attribute dominates the distance calculations due to its scale.
The Adult dataset includes both categorical and continuous variables. k-NN is effective because it can handle this mix by using distance metrics and the categorical data is converted using one-hot encoding to numerical. We used Aberami’s k-NN implementation throughout the project.

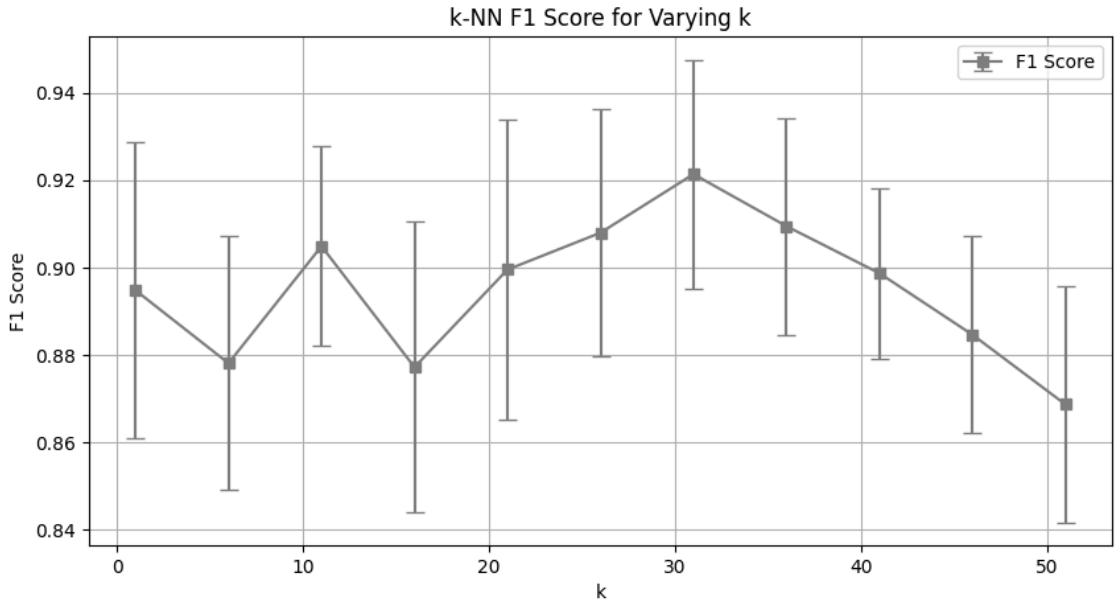
5.1.1 Experiment and Analysis

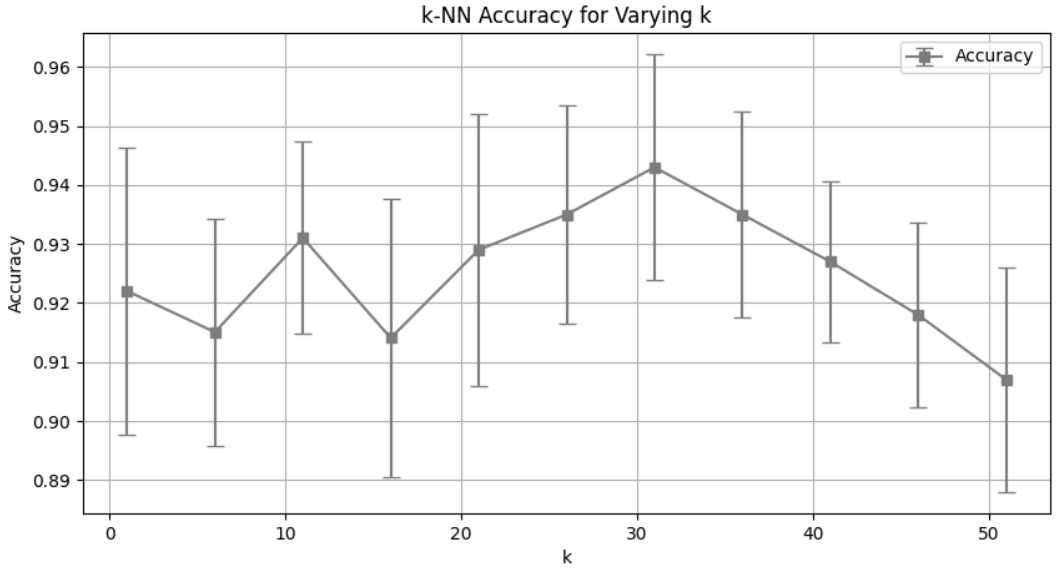
For this experiment, all the numeric attributes of the dataset were preprocessed using the ‘min-max’ normalization technique and the categorical attributes were one-hot encoded to numeric values. The K-Nearest Neighbors (k-NN) classifier’s performance was tested based on different values of the hyperparameter ’k’ ranging from 1 to 50 with the step size of 5.

The results from the k-Nearest Neighbors (kNN) experiments on the digits dataset show a trend where both accuracy and F1 score fluctuate till the value of n = 31, and then gradually decrease as the number of neighbors (k) increases, suggesting that incorporating more neighbors leads to the model considering more noise in the data, thereby reducing its predictive accuracy and precision-recall balance

From this table, the highest accuracy is observed at k=31 with an accuracy of 0.9430 and an F1 score of 0.9214. Both these metrics are the highest among all the provided values, indicating that k=31 yields the best overall prediction performance according to your data.

k	Accuracy	F1 Score
1	0.9220	0.8948
6	0.9151	0.8782
11	0.9310	0.9049
16	0.9141	0.8773
21	0.9290	0.8996
26	0.9350	0.9080
31	**0.9430**	**0.9214**
36	0.9350	0.9095
41	0.9270	0.8987
46	0.9180	0.8847
51	0.9070	0.8687





5. 2 Random Forest

The Random Forest algorithm is particularly useful for the Adult Income dataset because of its robustness, accuracy, and ability to handle various data types and complexities. It also captures interactions between variables automatically, which can be crucial for accurately predicting income based on factors like education level combined with occupation type. We used Asritha's Random Forest implementation throughout the project.

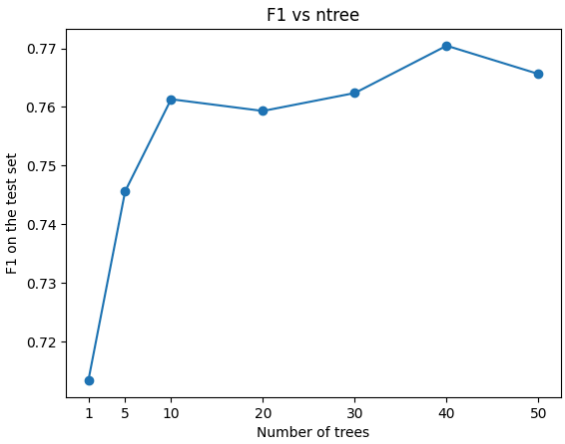
1.2.1 Experiment and Analysis

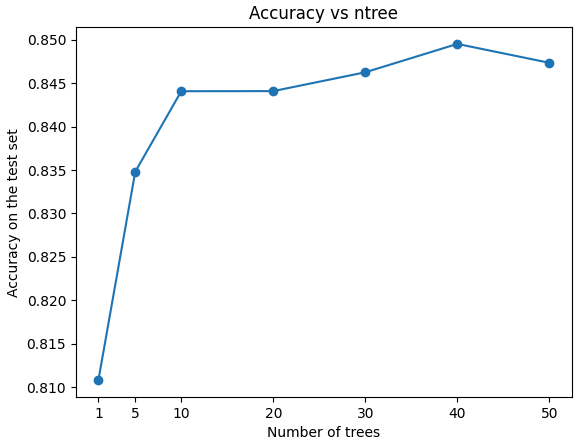
In the experiments, we evaluated the performance of a Random Forest algorithm on classifying handwritten digits by altering the 'number of trees' hyperparameter.

The results from the experiment indicate a clear trend where both accuracy and F1 score improve as the number of trees in the Random Forest increases. The increments in performance are notable from 1 tree to 10 trees, suggesting that adding more trees generally enhances model performance. However, beyond this point, although there are slight fluctuations in accuracy, adding more trees does not significantly enhance performance. The highest accuracy was recorded at 40 trees, showing only a marginal improvement compared to the performance with 30 trees. After reaching 40 trees, the performance slightly declines at 50 trees. The optimal choice for the number of trees is 40 because it provides the highest accuracy and F1 score without significant trade-offs.

n_tree	**Accuracy**	**F1 Score**
1	0.8108	0.7134

5	0.8348	0.7456
10	0.8441	0.7613
20	0.8441	0.7593
30	0.8462	0.7624
40	**0.8495**	**0.7704**
50	0.8473	0.7656





5.3 Neural Network - EXTRA CREDIT

Neural networks have the inherent ability to perform feature learning and find the most relevant features for making predictions during the training process. They further excel at capturing non-linear relationships between input features, which is often the case in census data where the influence of various demographic and economic factors on outcomes like income is not straightforward.. We used Asritha's Neural Network implementation throughout the project.

3.2.2 Experiment and Analysis

For our study, we selected the 'hidden layers' as the hyperparameter to be tested. From the tables below, we can see that the best neural network architecture was the one with the 8 hidden neurons in the hidden layer with an accuracy of 78.84% and a F1 score of 0.705.

Architecture	Accuracy	F1 Score

epochs=1000, learning_rate=0.01, regularization_param=0.001, layers=[input,4,7, output]	0.7605725825611784	0.5774376688590325
epochs=1000, learning_rate=0.01, regularization_param=0.001, layers=[input,3,4,5,output]	0.7360982418626753	0.5313144570069903
epochs=1000, learning_rate=0.01, regularization_param=0.001, layers=[input,8,output]	**0.7884622238061297**	**0.7051414714759384**

After identifying the best neural network architecture, it was used to generate a learning curve where the y axis shows the network's performance (J) on a test set, and where the x axis indicates the number of training samples given to the network in the step size of 5. It can be observed from the graph that the network's performance improves as the number of training examples grows and the cost function, J decreases with the number of training instances presented to the network increases. It may further be noted that the accuracy of the model's predictions also increases as more training instances are introduced.
Here, we can observe some gaps in the learning curve because the dataset consists of a lot of missing values, which are dealt with by replacing with other values but some discrepancies are still observed.