I. About Data

A. *Data Set 1*

This dataset was sourced from Kaggle and poses a binary classification challenge. The target variable takes on values of 0 or 1, indicating whether an individual's income falls below or exceeds $50,000. The dataset includes a range of demographic, education, and work-related attributes for each individual, which can be leveraged to predict their income level.

Upon initial examination, there were 8 categorical columns and 5 numerical columns. After cleaning the data by removing rows with missing values, we were left with a total of 30,162 instances for analysis.

To prepare the data for modeling, we transformed the categorical variables into dummy or indicator variables, expanding the feature set to a total of 105 variables.

Regarding the target variable distribution, approximately 75.9% of the instances are labeled 'Yes' (1), indicating income exceeding $50,000, while the remaining 24.1% are labeled 'No' (0), representing income less than or equal to $50,000.

B. *Data Set 2*

This dataset was obtained from Kaggle, and it presents a binary classification problem where the target variable takes on values 0 or 1, representing 'male' and 'female,' respectively. The dataset comprises information related to facial features, including attributes like forehead size, nose characteristics, and lip size and shape. In total, the dataset contains seven features and a label column.

The dataset consists of 5,001 instances, all of which have complete data without any null values. The target variable's distribution is perfectly balanced, with 50% of the instances labeled as 'Male' (1) and the other 50% as 'Female' (0).

C. *Interesting Data Sets*

The two datasets I'm working with present an interesting contrast: the first dataset is highly unbalanced, while the second is well-balanced. Both datasets are sufficiently large, providing ample data instances. This diversity in class distribution makes them particularly intriguing for analyzing the results of different classification models.

Before delving into the model evaluations, I've already completed some essential data preprocessing steps. This included handling null values and converting categorical variables into dummy or indicator variables. Throughout this assignment, I've leveraged popular Python libraries such as NumPy, Pandas, Scikit-learn, and Matplotlib for efficient data manipulation and visualization.

For each classification model, my approach consists of three phases:

Initial Assessment: I use Scikit-learn's cross-validated function to assess the model's performance with default settings across various training set sizes. This initial evaluation provides valuable insights into the model's behavior.

Hyperparameter Tuning: Following the initial assessment, I perform hyperparameter tuning by systematically iterating through different hyperparameter values. This fine-tuning process is crucial for optimizing model performance.

Final Evaluation: To gauge the model's true potential, I once again employ the cross-validated function, but this time with the finely tuned hyperparameter values. This step provides a comprehensive assessment of the model's capabilities.

D. *Performance Metric*

To evaluate model performance, I rely on the F1 score as a metric. This choice is particularly valuable when dealing with unbalanced class distributions, as is the case in dataset 1. The F1 score strikes a balance between precision and recall, making it a robust measure for assessing classification models in such scenarios.

II. Models

A. *Decision Tree*

TABLE I
DECISION TREE TEST SCORE

	Data Set 1		Data Set 2	
	Param Val	F1 Score	Param Val	F1 Score
Initial score		0.8815		0.9759
max depth	10	0.9027	6	0.9772
min samples leaf	30	0.9019	10	0.9740
ccp alpha		0.9108		0.9666
final f1 score		0.9154		0.9858

When applying the DecisionTreeClassifier to both DS1 and DS2 datasets, I observed a common issue—overfitting. This outcome was expected since Decision Trees tend to become increasingly complex as they grow deeper, resulting in a low bias and high variance model. It's crucial to strike a balance between model complexity and generalization.

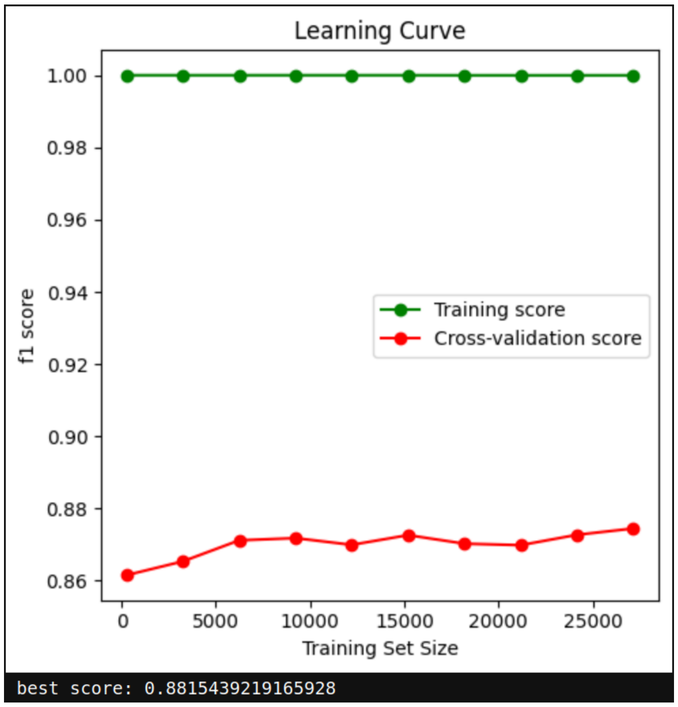
To address the overfitting problem, I employed pre-pruning and post-pruning techniques independently. The results are visualized in Fig(1,6).

Max Depth Validation Curve (Fig 2,7): The validation curve for maximum depth (max depth) shows that increasing the depth initially improves the F1 score. However, as the tree continues to grow deeper, overfitting becomes evident. This highlights the importance of limiting the maximum depth to achieve better generalization.

Min Sample Leaf Validation Curve (Fig 3,8): For the minimum samples per leaf (min sample leaf) parameter, we observed that as we increase this value, the training and test scores converge. This behavior indicates that a higher min sample leaf value prevents the tree from becoming overly complex, leading to better generalization.

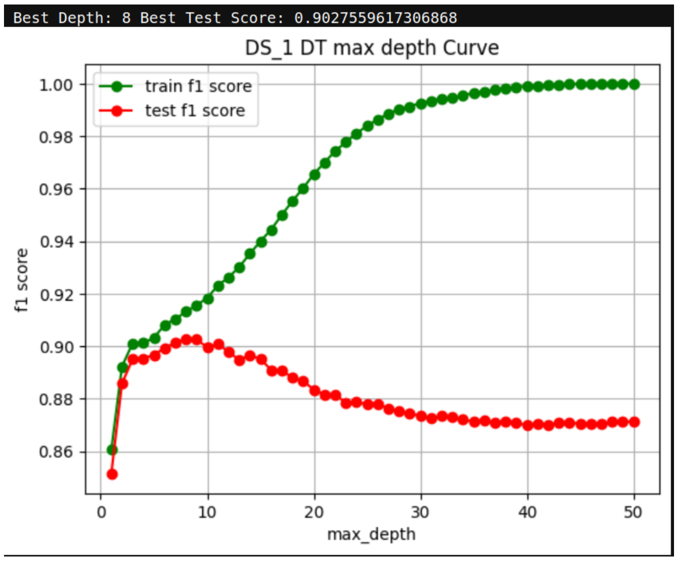
CCP_alpha (Complexity Parameter) Validation Curve (Fig 4,9): Increasing the CCP_alpha value results in more nodes being pruned after the model fitting. This process is crucial in controlling the tree's complexity and enhancing its ability to generalize.

Finally, Fig(5,10) illustrates the Decision Tree after post-pruning, showcasing the result of our efforts to combat overfitting.



```
best score: 0.8815439219165928
```

Fig. 1. DS1 DT1 Initial learning curve.



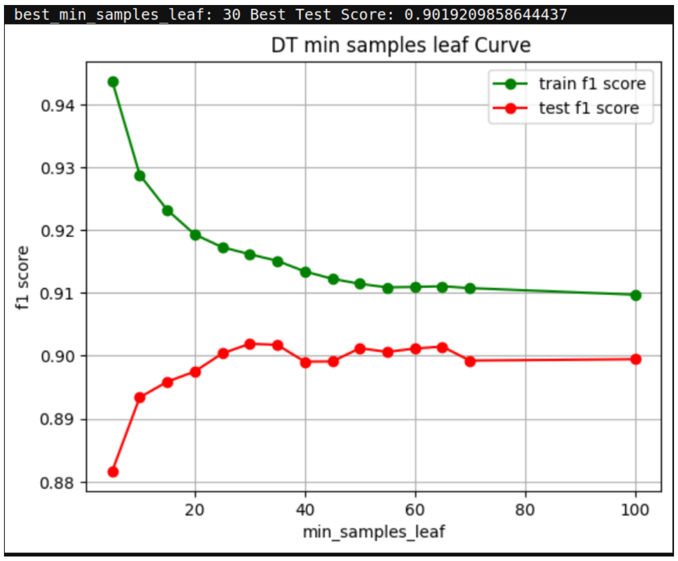
```
Best Depth: 8 Best Test Score: 0.9027559617306868
```

Fig. 2. DS1 DT1 max depth validation curve.

B. Boosting

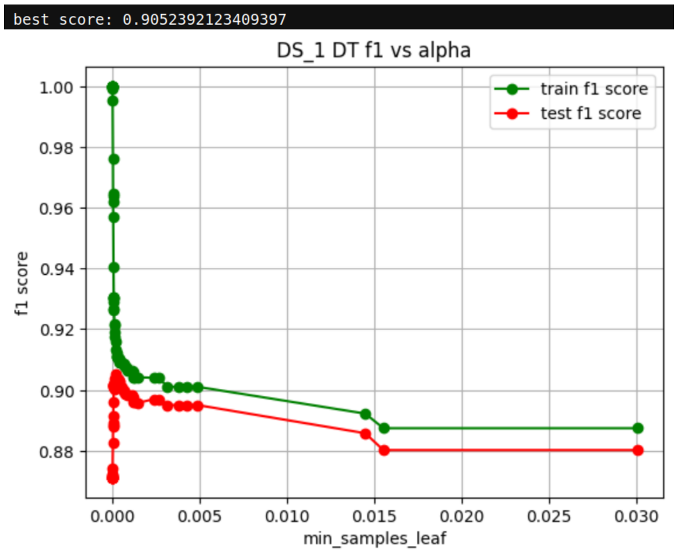
Table II and figures 11 and 13 depict the initial learning curves for DS1 and DS2, while Figures 12 and 14 showcase the validation curves for the number of estimators (the count of weak learners) using both datasets.

In the initial learning curves (Figures 11 and 13), we observe a characteristic pattern: the F1 score starts with a high value for the test set but a lower value for the cross-validation set. As the dataset size increases, the F1 score converge and then levels off, remaining nearly parallel to the x-axis. This



```
best_min_samples_leaf: 30 Best Test Score: 0.9019209858644437
```

Fig. 3. DS1 DT1 min sample leaf validation curve.

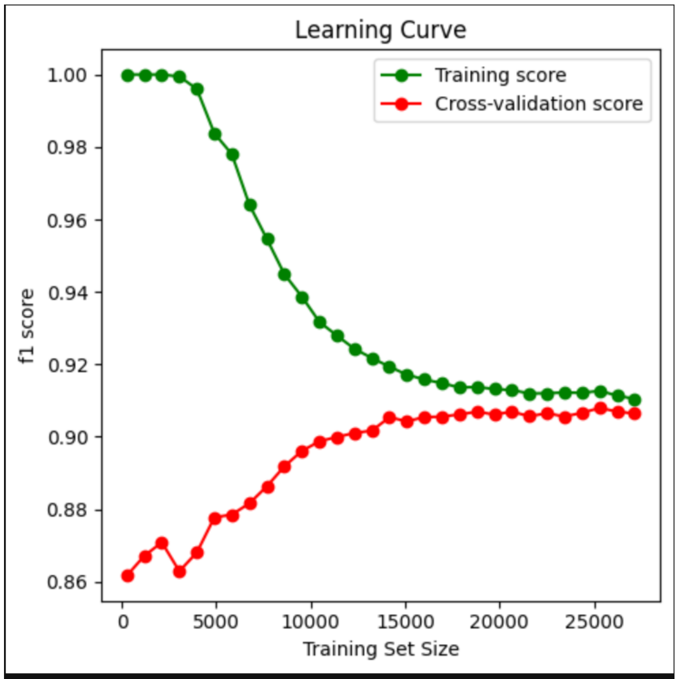


```
best score: 0.9052392123409397
```

Fig. 4. DS1 DT1 alpha validation curve.

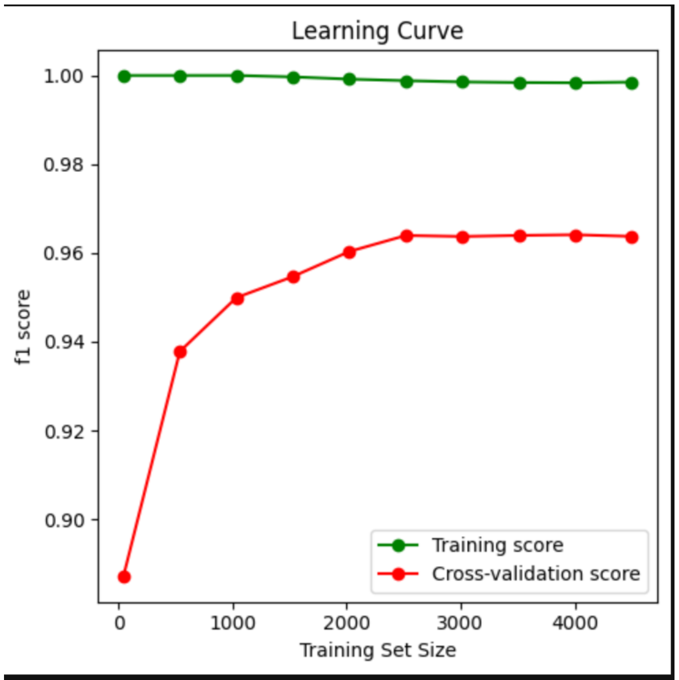
TABLE II
BOOSTING TEST SCORE

	Data Set 1		Data Set 2	
	Param Val	F1 Score	Param Val	F1 Score
Initial score		0.9148		0.9859
Weak learners	98	0.9104	6	0.9772
final f1 score		0.9192		0.9879



```
best score: 0.9154691462383769
```

Fig. 5. DS1 DT1 Final Learning curve after Hyperparameter Tuning.



```
best score: 0.9759036144578314
```

Fig. 6. DS2 learning curve.

```
Best Depth: 6 Best Test Score: 0.9772570640937285
```

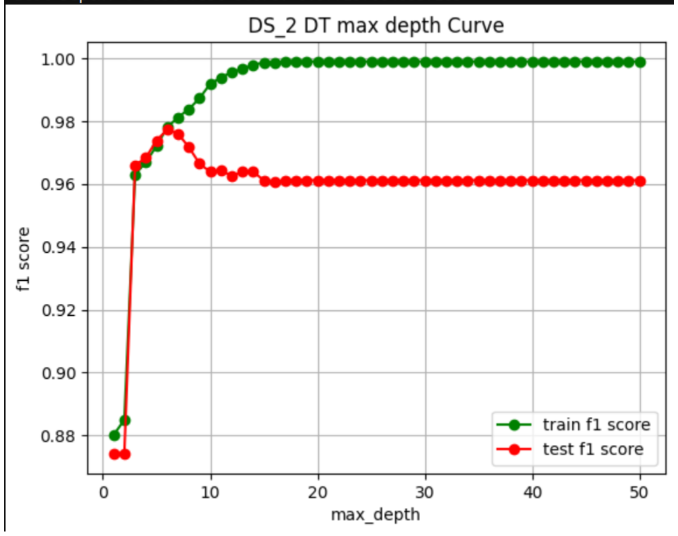


Fig. 7. DS2 max depth validation curve.

```
best_min_samples_leaf: 10 Best Test Score: 0.9740791268758526
```

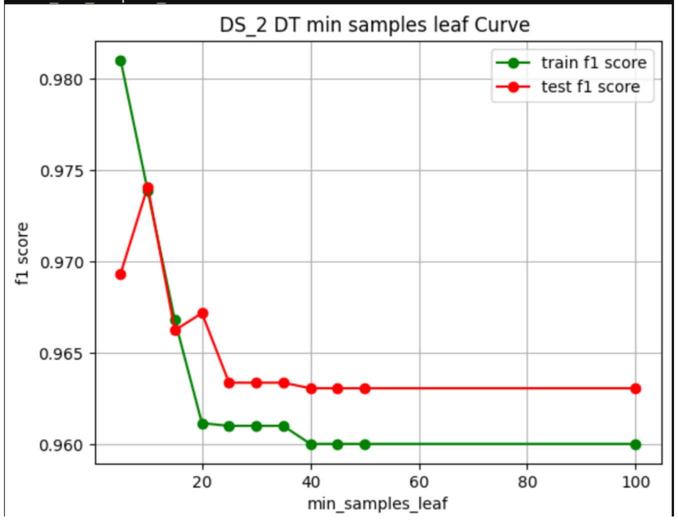


Fig. 8. DS2 min sample leaf validation curve.

behavior can be attributed to the boosting algorithm's reliance on the number of weak learners. As the model receives more data, it rapidly improves, reaching a point where additional data has minimal impact on its performance.

The validation curves for the number of estimators (Figures 12 and 14) further validate this observation. Both models achieve their peak performance with relatively few estimators. Increasing the number of weak learners beyond this point does not significantly enhance the model's performance.

In summary, these findings highlight the effectiveness of boosting models in leveraging a limited number of weak learners to achieve strong performance, and how the models reach a plateau in performance as they receive more data or estimators

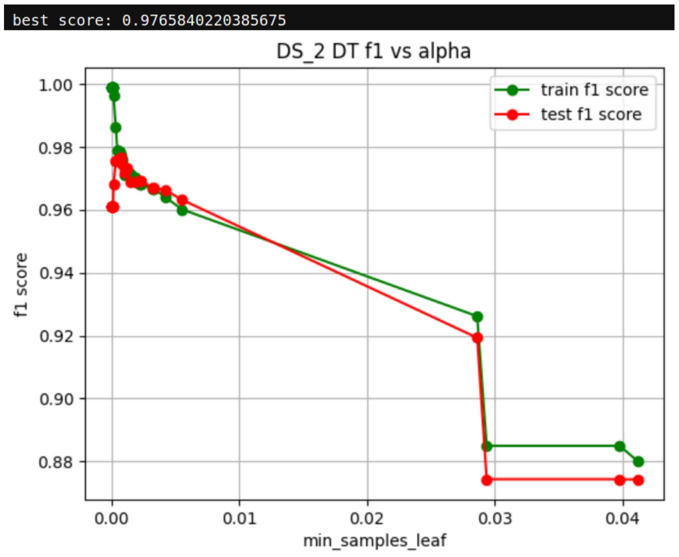
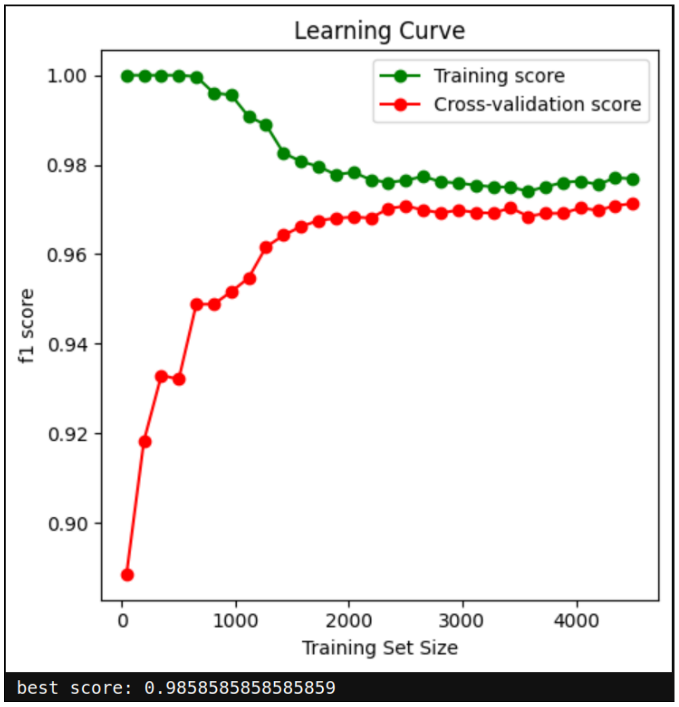

```
best score: 0.9765840220385675
```

Fig. 9. DS2 DT alpha validation curve.



```
best score: 0.9858585858585859
```

Fig. 10. DS2 max depth validation curve.

TABLE III
KNN TEST SCORE

	Data Set 1		Data Set 2	
	Param Val	F1 Score	Param Val	F1 Score
Initial score		0.8592		0.9858
n neighbors	25	0.8742	18	0.9747
final f1 score		0.8801		0.9858

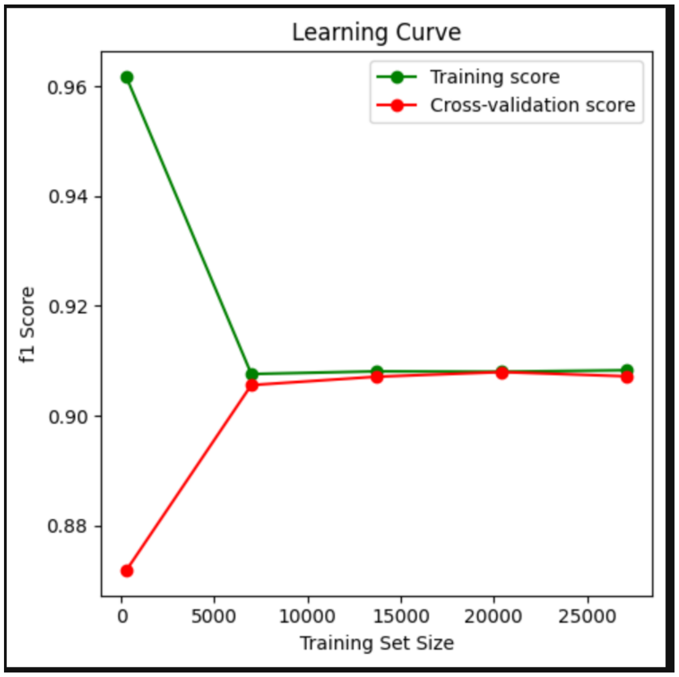


Fig. 11. DS1 boosting learning curve.

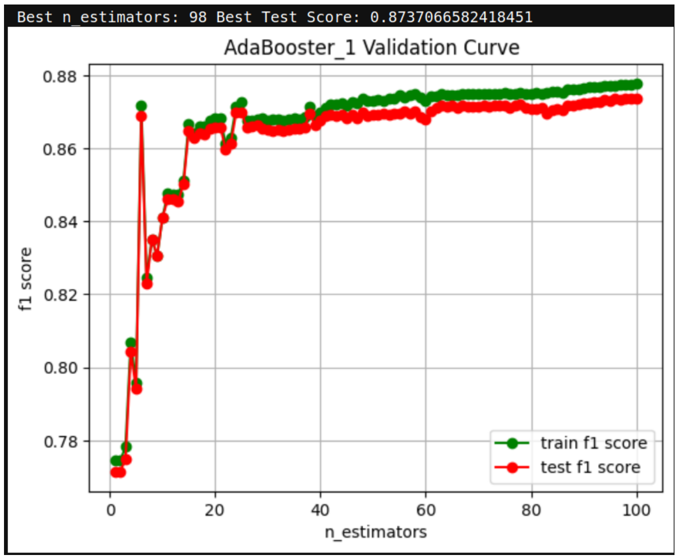
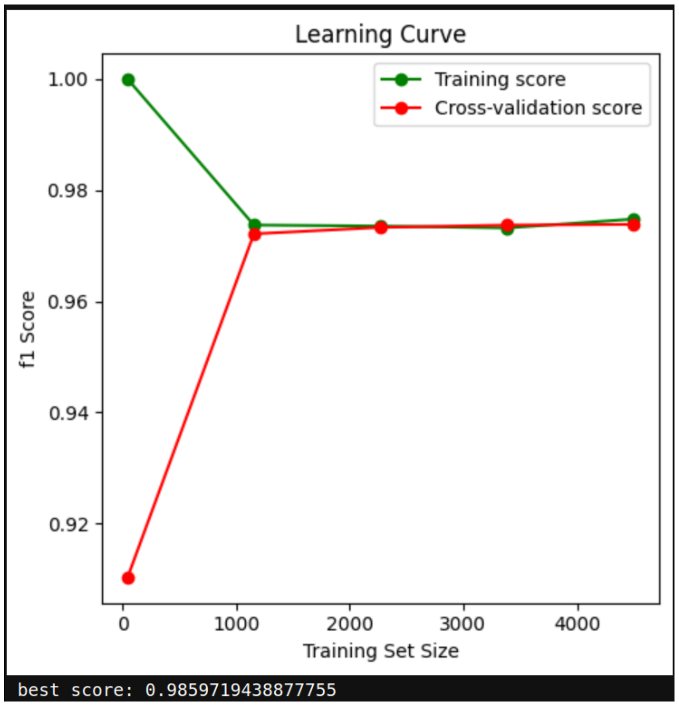

```
Best n_estimators: 98 Best Test Score: 0.8737066582418451
```

Fig. 12. DS1 boosting n estimaor

C. KNN

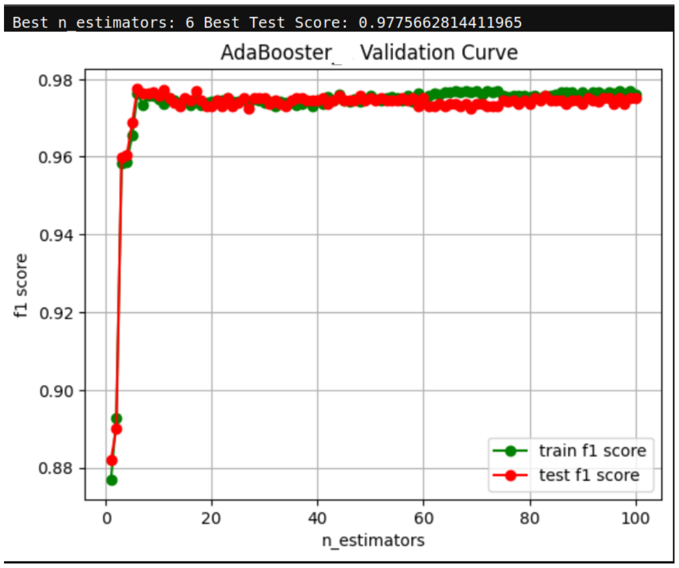
In Table III and Figures 15 and 18, we present the KNN learning curve using default values. An evident trend emerges as we increase the amount of training data—the model starts to exhibit overfitting. This phenomenon indicates that KNN, in its default configuration, tends to become overly complex when presented with more data.

To mitigate this overfitting issue, we turn our attention to the number of neighbors, denoted as 'n_neighbors.' Figures 16 and 19 show the validation curve for K (n_neighbors). Notably, as we increase the value of K, the gap between the F1 scores



```
best score: 0.9859719438877755
```

Fig. 13. DS2 boosting learning curve.



```
Best n_estimators: 6 Best Test Score: 0.9775662814411965
```

Fig. 14. DS2 boosting n estimators

of the training and test sets gradually diminishes. This trend suggests that a higher K value makes the model less sensitive to noise and reduces the risk of overfitting. In other words, it encourages a more robust and generalized performance.

Finally, in Figures 17 and 21, we present the KNN model after applying fine-tuning to the 'n_neighbors' parameter. This refinement ensures that the model's choice of K aligns with the data and addresses the overfitting problem, resulting in a more reliable and stable classification performance.

In summary, adjusting the 'n_neighbors' hyperparameter has proven effective in controlling overfitting in the KNN model, leading to improved generalization and more robust predictions

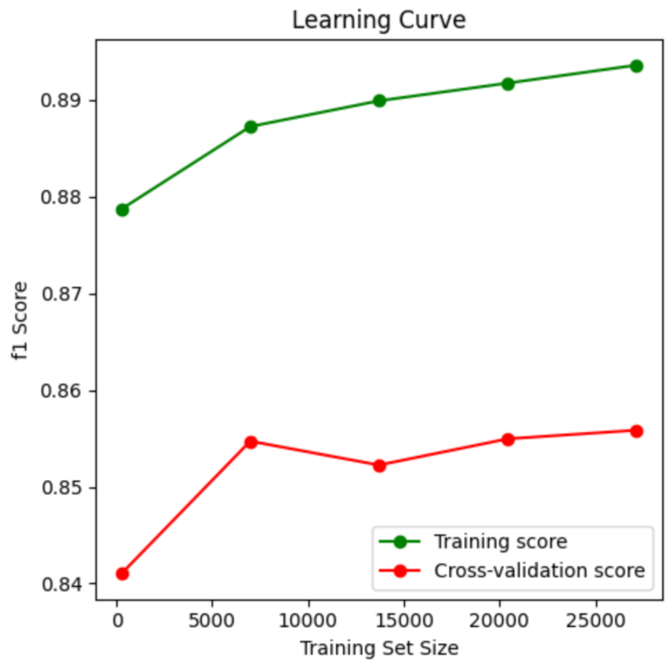
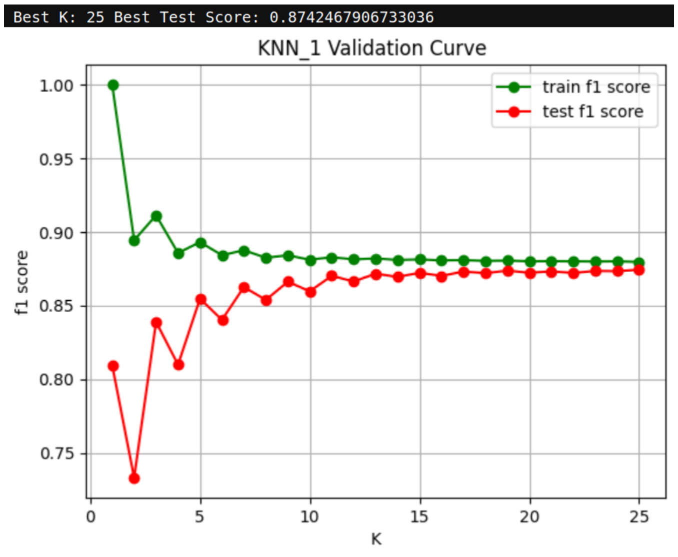


Fig. 15. DS 1 KNN learning curve.

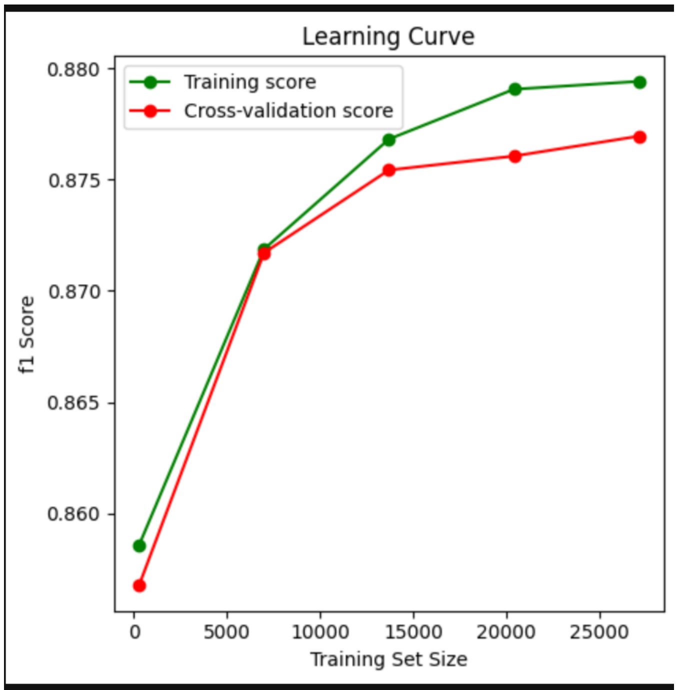


```
Best K: 25 Best Test Score: 0.8742467906733036
```

Fig. 16. DS 1 K validation curve

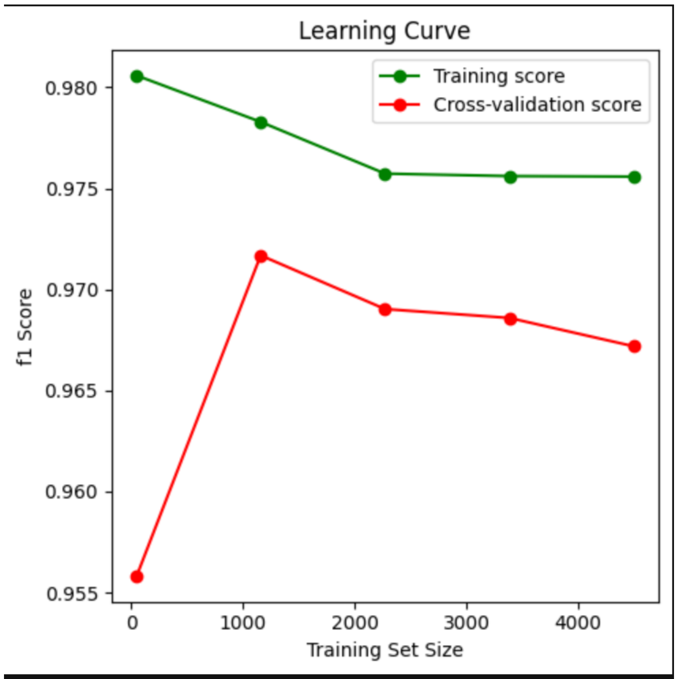
D. SVM

In Table IV and Figures 21 and 22, we present the initial learning curves of SVM with default settings applied to both Dataset 1 and Dataset 2. It's noteworthy that the training and test scores initially start apart but gradually converge. This convergence suggests that the default SVM kernel, the Radial



```
best score: 0.8801417043888997
```

Fig. 17. DS 1 KNN final learning curve



```
best score: 0.9858585858585859
```

Fig. 18. DS 2 KNN learning curve.

```
Best K: 18 Best Test Score: 0.9747095010252905
```

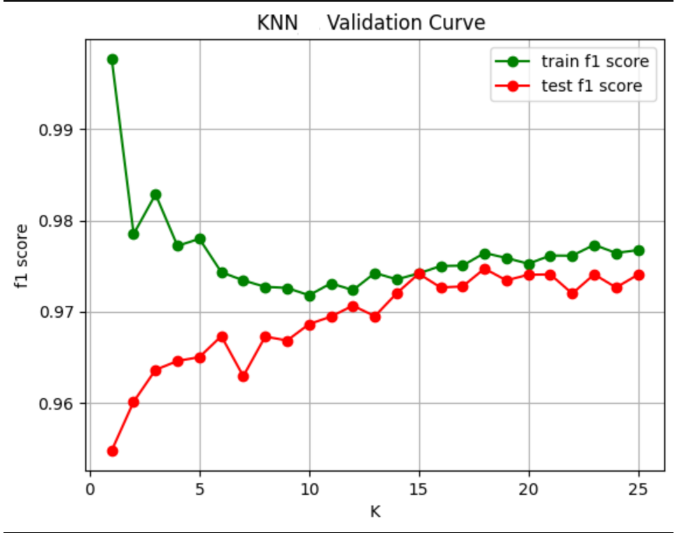
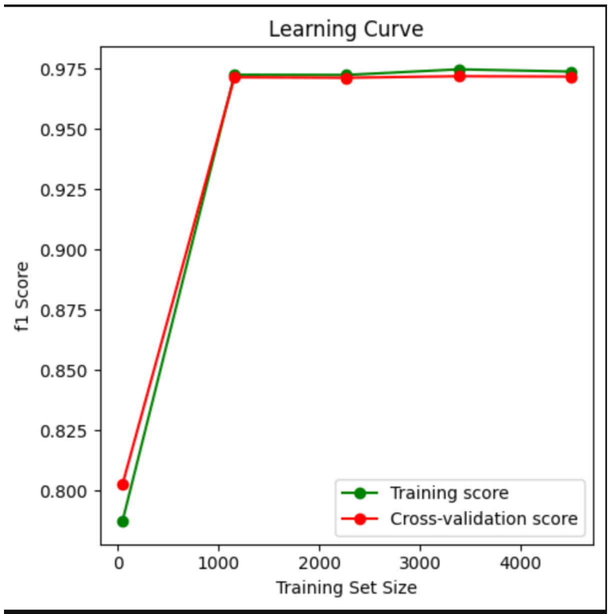


Fig. 19. DS 2 K validation curve



```
best score: 0.9858012170385395
```

Fig. 20. DS 2 KNN final learning curve

TABLE IV
SVM TEST SCORE

	Data Set 1		Data Set 2	
	Param Val	F1 Score	Param Val	F1 Score
Initial score		0.87927		0.9838
kernel	rbf	0.875991	rbf	0.96893
kernel	linear	0.874352	linear	0.9676567
kernel	poly	0.86809	poly	0.96736
kernel	sigmoid	0.76440	sigmoid	0.43419

Basis Function (RBF), effectively adapts and generalizes as more data becomes available.

When we switch from the default RBF kernel to the Sigmoid kernel for both datasets, it becomes evident that the Sigmoid kernel initially yields lower scores, especially noticeable in Dataset 2. Sigmoid kernels are better suited for datasets that exhibit a sigmoidal shape, and their performance can be sensitive to hyperparameter tuning. Other kernels, such as the polynomial (Poly) and linear kernels, outperform the Sigmoid kernel on both Dataset 1 and Dataset 2. To fine-tune SVM, we explored different values of the hyperparameter 'C' with both the Poly and Sigmoid kernels. In Table V and VI, as well as Figures 23, 24, 25, and 26, we delve into the validation curves for the hyperparameter 'C,' using both the Poly and Sigmoid kernels on both datasets. Our goal was to optimize the performance of the Sigmoid kernel, primarily by fine-tuning the 'C' value, as detailed in the accompanying table. We have observed that fine-tuning the 'C' hyperparameter has not led to a significant change in the F1 score when using the polynomial (Poly) kernel. However, it's noteworthy that the SVM model with the Sigmoid function demonstrates a substantial improvement in the F1 score on both datasets

TABLE V
SIGMOID TEST SCORE

	Data Set 1		Data Set 2	
	Param Val	F1 Score	Param Val	F1 Score
Initial score		0.7644		0.43419
C	0.001	0.8560	0.0001	0.6568

TABLE VI
POLY TEST SCORE

	Data Set 1		Data Set 2	
	Param Val	F1 Score	Param Val	F1 Score
Initial score		0.86809		0.96736
C	5	0.86915	1000	0.96853

E. *Neural Network (NN)*

TABLE VII
NN TEST SCORE

	Data Set 1		Data Set 2	
	Param Val	F1 Score	Param Val	F1 Score
Initial score		0.8968		0.9714
HiddenLayer 7	10	0.9024	150	0.9702
HiddenLayer 2	7	0.9031	100	0.9721
final f1 score		0.9009		0.9721

Initially, when the MLPClassifier was applied to Data Set 1, it exhibited significant instability, with F1 scores for both the training and test data fluctuating between 0.6 and 0.8. A likely culprit was the model's sensitivity to feature scaling. To mitigate this, I applied MinMaxScaler to the datasets, resulting in a relatively stable model with an F1 test score of approximately 0.89 (Fig 27).After iterative trying different

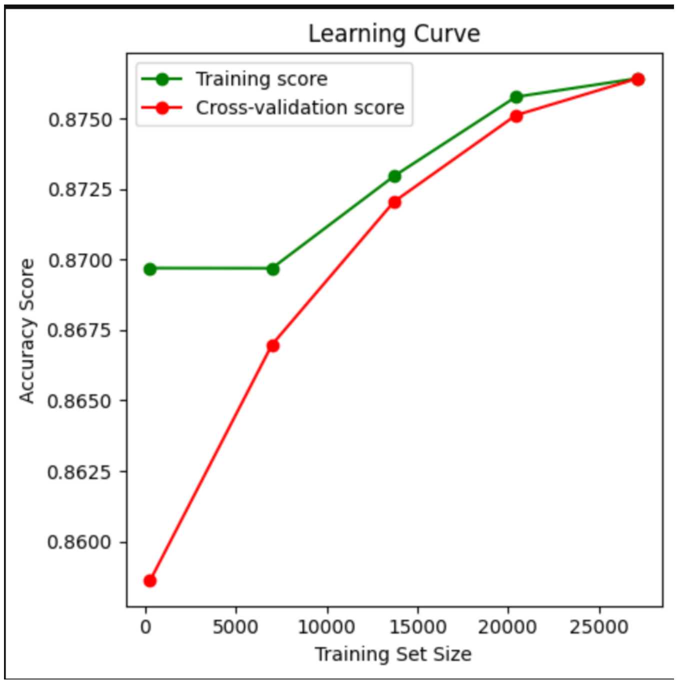
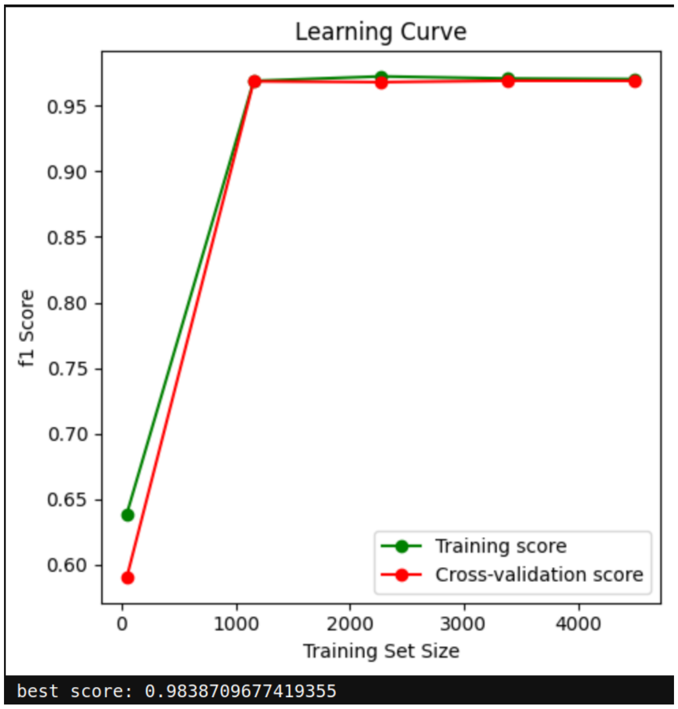


Fig. 21. DS1 SVM learning curve.



```
best score: 0.9838709677419355
```

Fig. 22. DS 2 SVM learning curve.

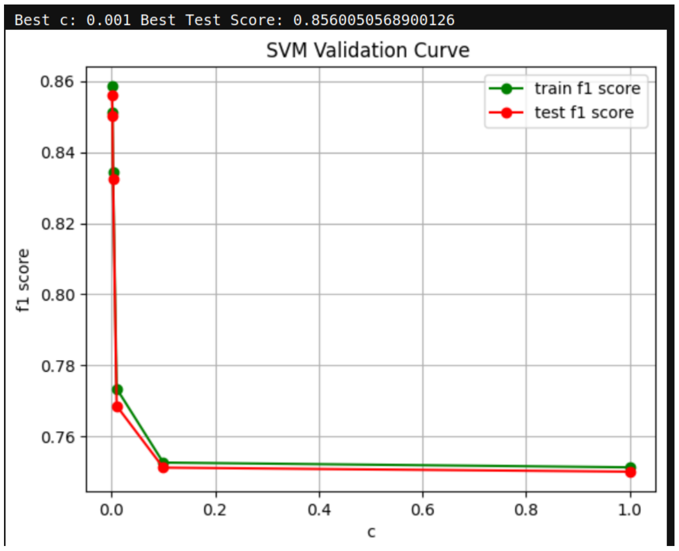


Fig. 23. DS 1 Sigmoid validation curve.

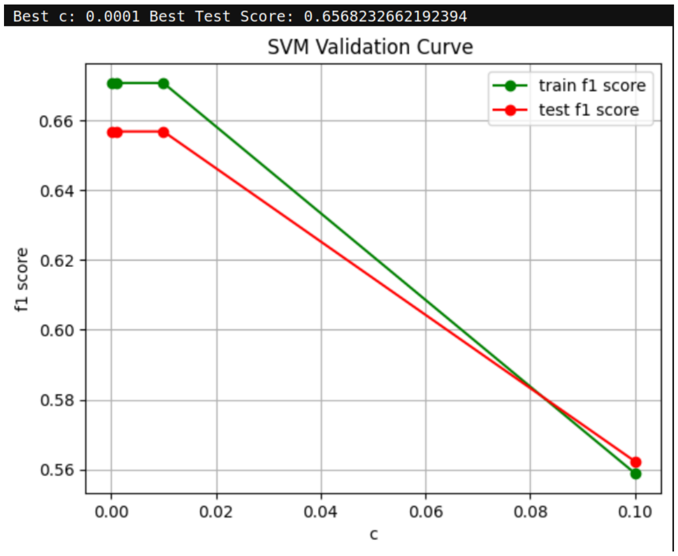


Fig. 25. DS 2 SVM Sigmoid validation curve.

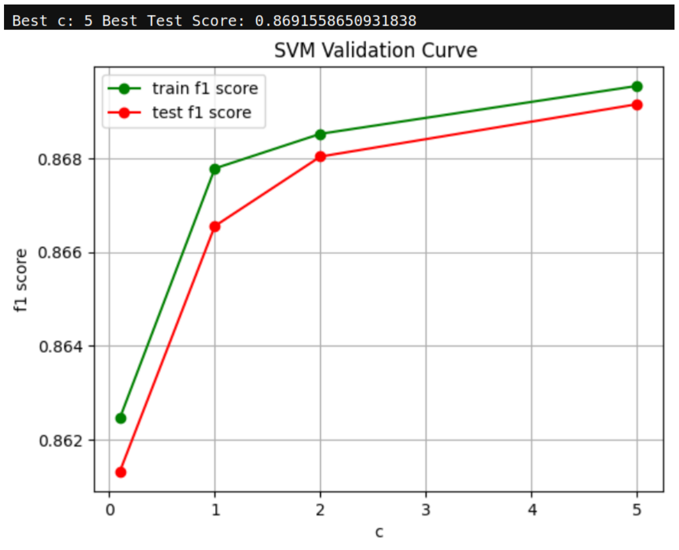


Fig. 24. DS 1 Poly validation curve.

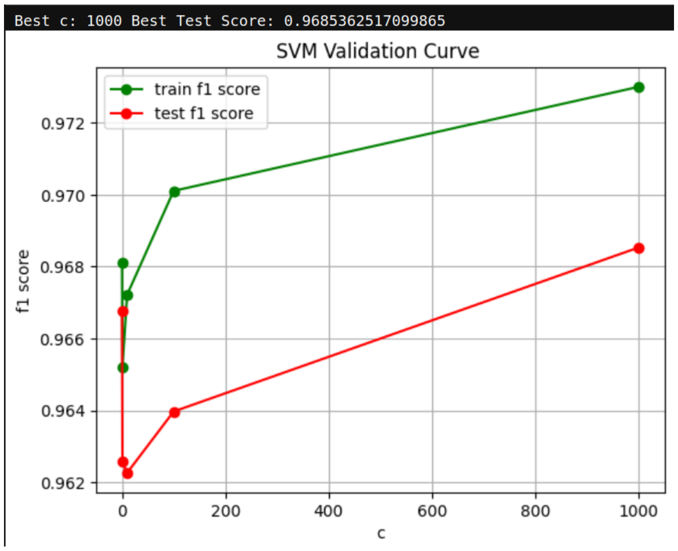


Fig. 26. DS 2 SVM Poly validation curve.

value of hidden layer 1 (Fig 26) and then hidden layer 2 (Fig 28,29) model f1 score stabalise around 0.90 (Fig 30).

When the MLPClassifier was applied to Data Set 2, I observed an upward trend in the F1 score for the training data and an initially upward, followed by a horizontally stable trend in the F1 score for the test data (Fig 31). Despite spending considerable time fine-tuning the hidden layers, the model showed limited improvement in the F1 test score (Fig 32,33,34)

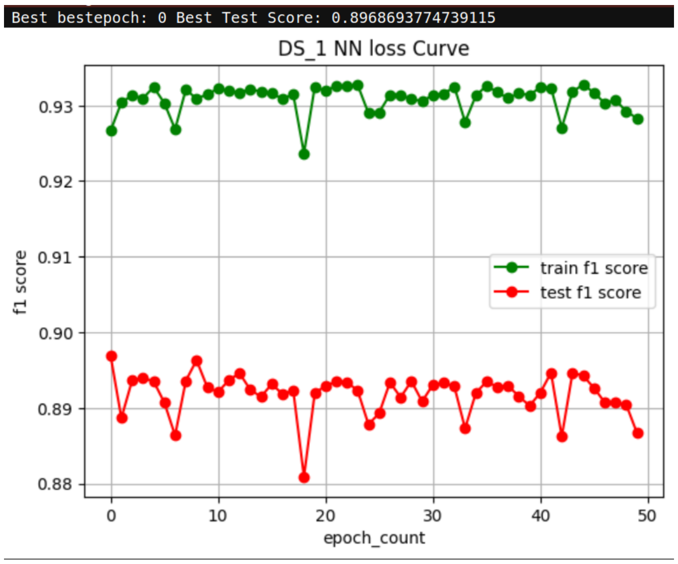
In summary, it's apparent that fine-tuning the hidden layers on both datasets did not yield any improvement in the F1 score; instead, it started to exhibit signs of overfitting. This observation can be attributed to the initial F1 score being very similar to the F1 scores of other models. Essentially, this indicates that the model reached its performance ceiling, and further improvements with the given data may not be achievable.

III. Model Fit Time

The model fit times, calculated using fine-tuned hyperparameters on the same datasets and computer setup, reveal in-

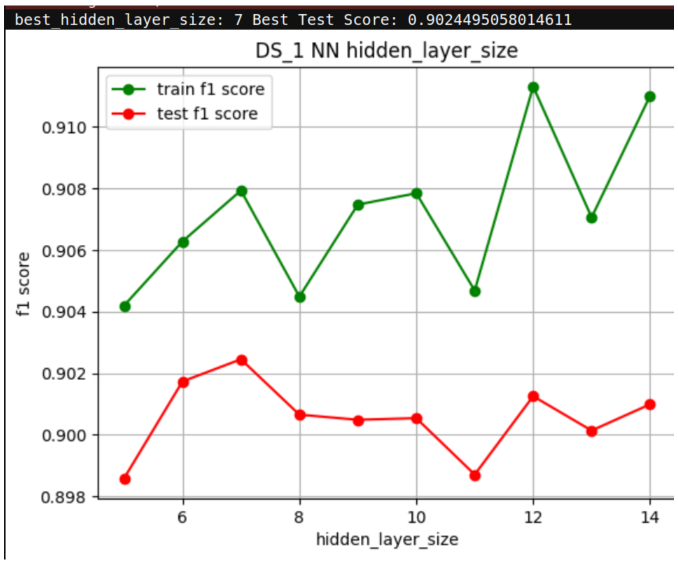
TABLE VIII
Model Fit Time

Model	Data Set1	Data Set2
Decision Tree	0.1808	0.003761
Booster	1.6987	0.01309
KNN	0.08538	0.00632
NN	39.2756	5.07370
SVM sigmoid	42.1155	0.61264
SVM poly	163.6230	0.1210



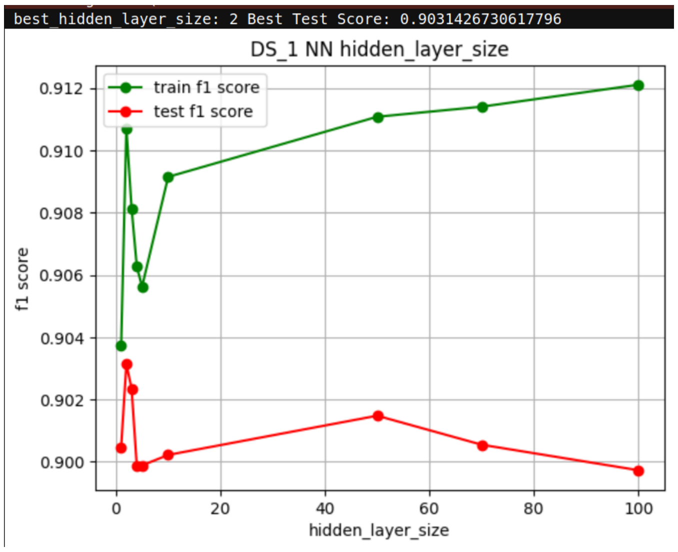
```
Best bestepoch: 0 Best Test Score: 0.8968693774739115
```

Fig. 27. DS1 NN Initial loss curve.



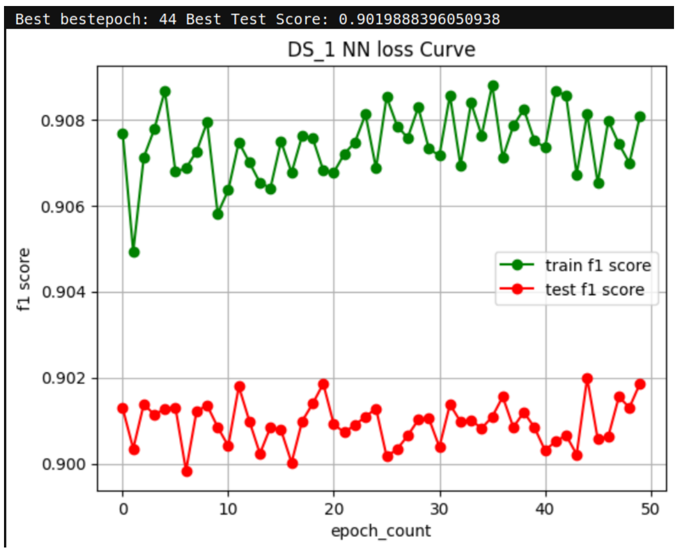
```
best_hidden_layer_size: 7 Best Test Score: 0.9024495058014611
```

Fig. 28. DS1 NN Hiddenlayer 1.



```
best_hidden_layer_size: 2 Best Test Score: 0.9031426730617796
```

Fig. 29. DS1 NN Hiddenlayer 2



```
Best bestepoch: 44 Best Test Score: 0.9019888396050938
```

Fig. 30. DS1 final loss curve

teresting insights into the computational efficiency of different algorithms:

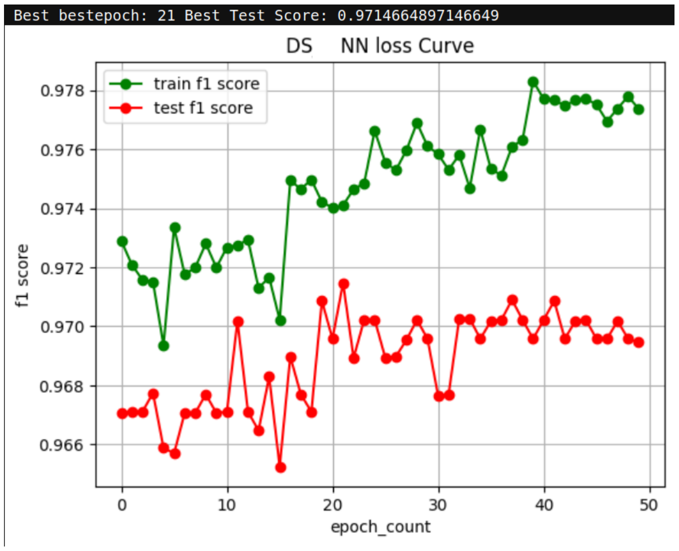
KNN: KNN exhibits the lowest fit time among the models. This is primarily because KNN is a lazy learning algorithm. It doesn't explicitly build a model during training but instead stores the entire dataset, which has an O(nd) time complexity for loading data into memory. There is no additional computational cost associated with training.

Decision Tree: The fit time of Decision Trees is the second lowest. It depends on the number of data points multiplied by the number of features and the depth of the tree. Data Set 1 takes more time than Data Set 2 because it contains more data points and features.

Booster: AdaBoost typically have a higher fit time compared to Decision Trees. Its training time complexity depends on the number of data points, the number of features, and the number of weak learners (boosting iterations). Data Set 1, with 98 weak learners, requires more time than Data Set 2, which has only 6.

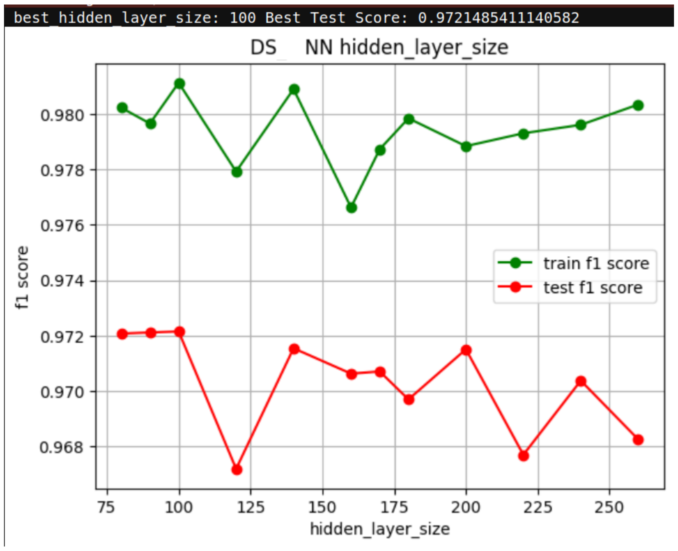
Neural Network (NN): The fit time of Neural Networks depends on factors such as the number of layers, neuron sizes, the number of training iterations, and the number of training samples. In Data Set 1, we have a large dataset but relatively small hidden layers. In Data Set 2, we have large hidden layers but a smaller dataset compared to Data Set 1.

SVM (Support Vector Machine): SVM exhibits the highest fit time, particularly for Data Set 1. SVM fit time depends on the number of data points, the choice of kernel function, and the data scaling. Sigmoid kernels are simpler and computationally less demanding than polynomial (poly) kernels.



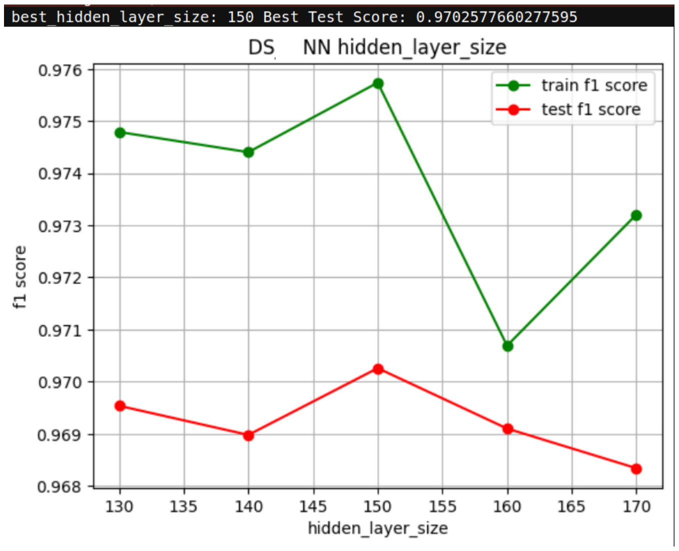
```
Best bestepoch: 21 Best Test Score: 0.9714664897146649
```

Fig. 31. DS2 NN Initial loss curve.



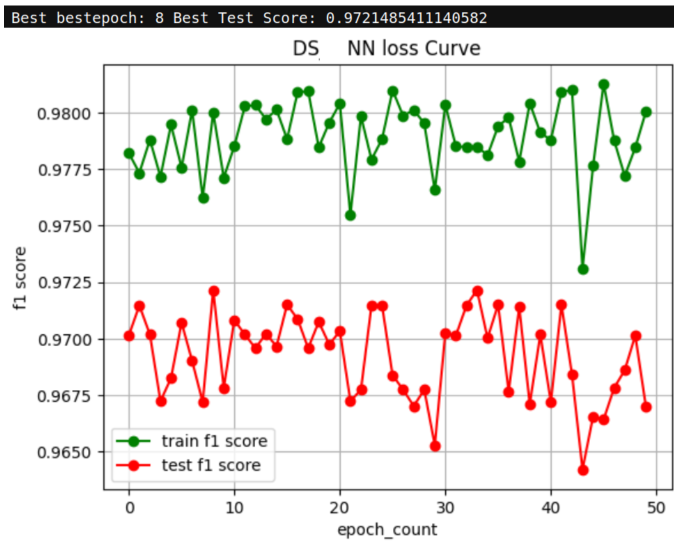
```
best_hidden_layer_size: 100 Best Test Score: 0.9721485411140582
```

Fig. 33. DS2 NN Hidden layer 2.



```
best_hidden_layer_size: 150 Best Test Score: 0.9702577660277595
```

Fig. 32. DS2 NN Hidden layer 1.



```
Best bestepoch: 8 Best Test Score: 0.9721485411140582
```

Fig. 34. DS2 NN final loss curve.

After applying data scaling to Data Set 1, the fit time for sigmoid kernel SVM reduces to 26.17980 and poly to 18.7931, highlighting the sensitivity of SVM kernels to data scaling.

In summary, the observed fit times highlight the computational characteristics of each model and their sensitivity to dataset size, complexity, and preprocessing steps like data scaling."

IV. Model Comparison

Data Preprocessing: Certain models exhibit varying performance based on the quality of input data. Notably, the curse of dimensionality negatively impacted KNN's performance, making it the least effective among all models when applied to Data Set 1, which boasts over 100 features. In contrast, this issue did not affect its performance on Data Set 2, which comprises just seven features. Furthermore, implementing feature scaling on Data Set 1 led to improved results for NN and SVM. Interestingly, models like Decision Tree and AdaBoost appeared impervious to data quality concerns.

F1 Score: When we examine the F1 scores of models on Data Set 2, we observe consistently high performance, with all models scoring between 0.96 and 0.98. This likely stems from the dataset's balanced nature and its limited number of features. However, on Data Set 1, we see significant disparities in performance. Decision Tree consistently outperformed other models, possibly due to rigorous hyperparameter fine-tuning, which enabled us to select the most optimal configuration. Conversely, KNN posted the lowest performance on Data Set 1, possibly due to the dataset's extensive feature set, a characteristic to which KNN is particularly sensitive.

Model Fit Time: Examining model fit times provides an alternative perspective on model performance. Decision Tree and KNN consistently delivered quicker results than other models. Meanwhile, NN and SVM exhibited notably longer fit times, indicating substantial computational demands.

References

Other then course lecture and book

- https://www.kaggle.com/
- https://towardsdatascience.com/
- https://medium.com/
- https://datascience.stackexchange.com/
- https://chat.openai.com/