COMPSCI 589 - Machine Learning
Final Project_Spring 2025
Eunbi Yoon, Eric Nunes

*** Clarify whose implementation was used**
For the regular credit portion, Eunbi was responsible for Dataset 1 and Dataset 2, while Eric handled Dataset 3 and Dataset 4. To verify the implementation, please refer to the folder named "Eunbi_Regular" for Datasets 1 and 2, and "Eric_Regular" for Datasets 3 and 4.
All extra credit components were solely designed and implemented by Eunbi, and the corresponding code can be found in the "Eunbi_Extra" folder.

*** Dataset 1**
Since it takes a lot of time to load dataset from sklearn, in the datasets folder inside of “Eunbi Regular”, I load the dataset1 the same as other dataset using load_digits.py.

Regular Credits

1. Choose Algorithm and Why

1-1. Choose Neural Network and Random Forest For Dataset 1, Why

First, choose the Neural Network because its hidden layers can effectively learn complex nonlinear patterns within the data. Especially in high-dimensional pixel-based data, where feature interactions matter, a multi-layer Neural Network is well-suited to capture these dependencies.
On the other hand, the Random Forest algorithm, which ensembles multiple decision trees, was chosen for its robustness against overfitting, interpretability, and minimal need for data preprocessing. Random Forest tends to perform well when many features are relatively independent, such as pixel values, and it is known for its strong performance across various classification tasks.

1-2. Choose KNN Algorithm and Random Forest For Dataset 2, Why

The Parkinson’s dataset exhibits a significant class imbalance, with positive cases accounting for approximately 75% of all samples. Given this imbalance, we selected the K-Nearest Neighbors (KNN) and Random Forest algorithms for evaluation. KNN, being a locality-based classifier, is particularly sensitive to imbalanced distributions, making it a valuable baseline for observing how such imbalance affects prediction performance. On the other hand, Random Forest utilizes bootstrapping and majority voting, which enables it to maintain relatively robust performance even in the presence of class imbalance. The two algorithms offer contrasting perspectives—sensitivity versus robustness—on imbalanced data, making them complementary choices for comprehensive evaluation.

1-3. Choose kNN and Random Forest For Dataset 3, Why

For the Rice dataset, we chose the kNN and RF algorithms due to their advantages and the given data. The data appears to be moderately balanced (a roughly 60/40 split among the two rice types), and is composed entirely of numbers. For this reason, I selected a kNN algorithm for its ability to cover distances well (with numeric entities) and for it being a simple model that has the capability of discriminating between two classes. I also selected the random forest model for being able to handle well with numeric attributes and being able to find more complex patterns at a reasonable pace compared to other algorithms.

1-4. Choose kNN and Random Forest For Dataset 4, Why

The credit approval dataset appears to be close to balanced (a split of approximately 55/45), and is composed of both numeric and categorical features. We selected a random forest for its ability to accommodate multimodal forms of data. I also selected kNN for its simplicity and for it being good as a discriminator since the dataset deals with a lot of categorical features.

2. Various Hyper-Parameter Settings Evaluation

2-1. Dataset 1(Digits) - Random Forest

Stopping Criteria : Minimal information gain = 0.00001 or Max depth = 5

Hyper Parameter	**Performance Evaluation**	
ntree	**Accuracy**	**F1 Score**
1	0.7499	0.7640
5	0.8123	0.7550
10	0.6806	0.7275
20	0.8176	0.8553
30	0.8504	0.8445
40	0.9357	0.9246
50	0.8823	0.8775

2-2. Dataset 1(Digits) - Neural Network

Alpha size = 0.1 / Mini-Batch Gradient Descent (batch size = 64) / Stopping Criteria : m_size=50

Hyper Parameter		**Performance Evaluation**	
Hidden Layer	**Regularization (Lambda)**	**Accuracy**	**F1 Score**
[64,32,16,8,4]	0.1	0.5250	0.4824
[64,32,16,8,4]	0.001	0.6940	0.6517
[64,32,16,8,4]	0.000001	0.7608	0.6321
[64,32,16, 8]	0.1	0.9916	0.9914
[64,32,16]	0.1	0.9944	0.9944
[32]	0.000001	0.9945	0.9943
[64]	0.000001	0.9972	0.9973

2-3. Dataset 2(Parkinsons) - KNN Algorithm

Hyper Parameter	**Performance Evaluation**	
k	**Accuracy**	**F1 Score**
1	0.9381	0.9582
3	0.9334	0.9557
5	0.9037	0.9379
9	0.8984	0.9346
19	0.8670	0.9184
39	0.8357	0.9005
51	0.8002	0.8831

2-4. Dataset 2(Parkinsons) - Random Forest

Stopping Criteria : Minimal information gain = 0.00001 or Max depth = 5

Hyper Parameter	Performance Evaluation	
ntree	Accuracy	F1 Score
1	0.8046	0.8836
5	0.8515	0.8943
10	0.8826	0.9178
20	0.8712	0.9094
30	0.9076	0.9309
40	0.9179	0.9369
50	0.9082	0.9254

2-5. Dataset 3 (Rice) - kNN

Hyper Parameter	Performance Evaluation	
k	Accuracy	F1 Score
1	0.8806	0.9126
3	0.9048	0.9074
5	0.9048	0.9021
9	0.9048	0.9076
15	0.9073	0.9128
19	0.9048	0.9100
39	0.9128	0.9073
51	0.9102	0.9073

2-6. Dataset 3 (Rice) - Random Forest

Hyper Parameter	Performance Evaluation	
ntree	Accuracy	F1 Score
5	0.9092	0.9074
10	0.9110	0.9100
20	0.9176	0.9162
30	0.9179	0.9163
50	0.9202	0.9187
100	0.9223	0.9207

2-7. Dataset 4 (Credit Approval) - kNN

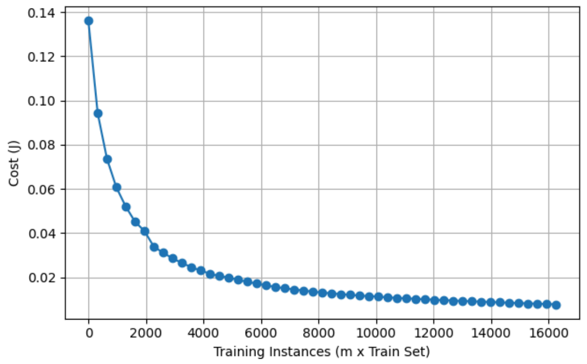
Hyper Parameter	Performance Evaluation	
k	Accuracy	F1 Score
1	0.7261	0.8172
3	0.8325	0.8630
5	0.8174	0.8633
7	0.8174	0.8939
9	0.8174	0.8630
19	0.8020	0.8351
39	0.8020	0.8497
51	0.8020	0.8351

2-8. Dataset 4 (Credit Approval) - Random Forest

Hyper Parameter	Performance Evaluation	
ntree	Accuracy	F1 Score
5	0.8300	0.8291
10	0.8406	0.8419
20	0.8419	0.8412
30	0.8545	0.8543
40	0.8606	0.8605
50	0.8588	0.8587

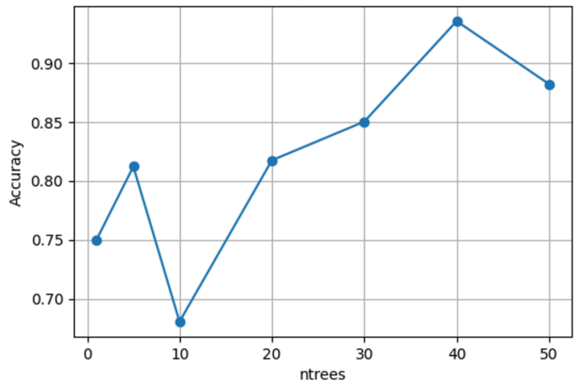
3. Construct learning curves/graphs with best parameter

3-1. Dataset1 - Neural Network



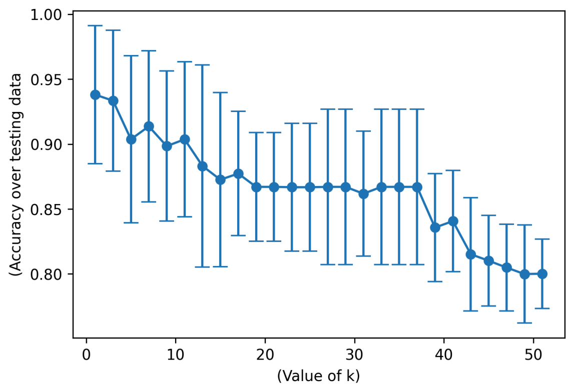
The graph shows a steadily decreasing cost J as the number of training instances increases, which indicates that the neural network is successfully learning from the data. The curve follows an exponentially decaying pattern and appears to converge smoothly without signs of plateauing prematurely. This suggests that the optimization process is not getting stuck in a poor local minimum, and that the network generalizes well as more data is seen. Since the dataset contains only numerical attributes (pixel intensities), the neural network is well-suited for this type of input due to its ability to model complex, high-dimensional relationships effectively.

3-2. Dataset1 - Random Forest



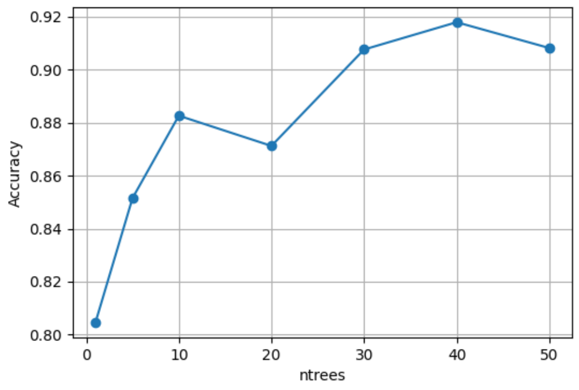
In contrast, the Random Forest graph exhibits non-monotonic behavior. While accuracy generally improves as the number of trees increases—peaking around ntrees=40—it fluctuates at smaller values of ntrees and slightly declines at ntrees = 50. This pattern indicates that initially, additional trees help reduce bias and improve performance, but beyond a certain point, the ensemble begins to overfit or adds little new information, possibly due to redundancy among trees. This behavior reflects the bias–variance tradeoff, where too many trees can increase variance if not properly regularized. Despite this, Random Forest still performs well on this dataset, which is expected because the input features are purely numerical and well-suited to decision tree splits.

3-3. Dataset2 - KNN Algorithm



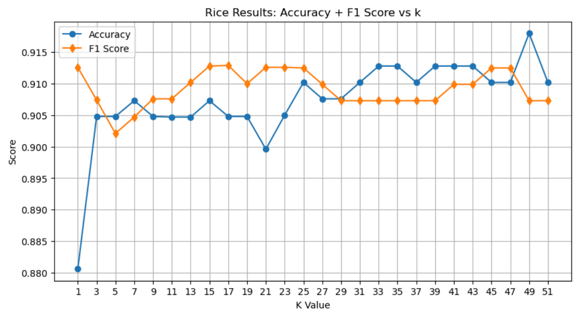
The KNN graph shows that the model performs best at smaller values of k, with accuracy gradually decreasing as k increases. This behavior is typical in datasets where local neighborhood information is highly relevant for classification. Parkinson’s dataset consists of only numerical biomedical voice features, which makes distance-based methods like KNN suitable. However, when kkk increases too much, the algorithm becomes overly generalized and less sensitive to subtle distinctions between healthy and diseased voices. Additionally, the wide error bars at higher kkk values suggest inconsistency across folds, likely due to class imbalance in the dataset.

3-4. Dataset2 - Random Forest



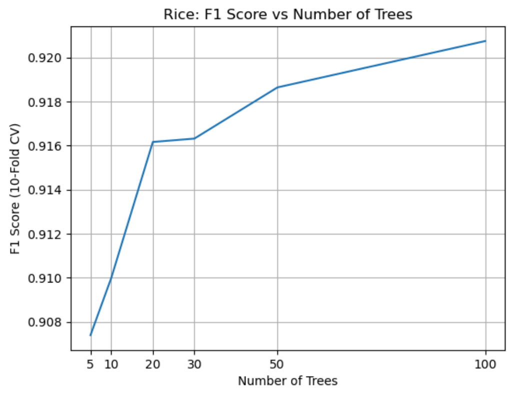
The Random Forest graph shows a clear trend of performance improvement with more trees, stabilizing around ntrees=40. This indicates the model benefits from ensemble learning, with more trees reducing variance and improving generalization. Since all features in the Parkinson’s dataset are numerical, decision tree splits can effectively capture the thresholds between the two classes (healthy vs diseased). Random Forest amplifies this advantage through feature bagging and bootstrap aggregation. Importantly, Random Forest remains robust even under class imbalance, which is a key challenge in the Parkinson's dataset (where class 1 dominates). This is because decision trees within the forest can focus on different subsets of the data, and class weight balance can implicitly emerge across trees. In contrast, KNN makes predictions based on local majority voting, which can be easily biased when one class dominates, especially in small neighborhoods. This makes Random Forest a more stable choice for imbalanced binary classification tasks.

3-5. Dataset3 - kNN



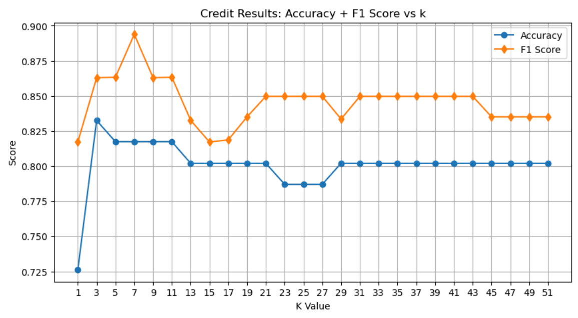
Here, we can observe that there is a little bit of fluctuation of both the accuracy and F1 values up to k=51; keep in mind that these fluctuations are still relatively small, with the greatest difference being about 0.03, and the average difference between the values being less than 0.01 difference. This would suggest that even at k=51, the model would still be roughly as competitive as a model at, say, k=29, and even a few of the lower k-value models seem to have slightly less accuracy than when k is in the range of 30-50. Indeed, the decrease in accuracy and F1 values is only more pronounced and visible by the time k reaches the range of 150. That being said, because the difference in accuracies/F1-Scores between lower and higher k values is within just 2%, as well as to ensure that the model doesn't rely on too many of its neighbors, I propose that the best model has a *k*=15, though I would also believe that choosing any model with a k greater than 15 within the given sample would also be acceptable; just remember to favor simpler models over ones that rely on more neighbors.

3-6. Dataset3 - Random Forest



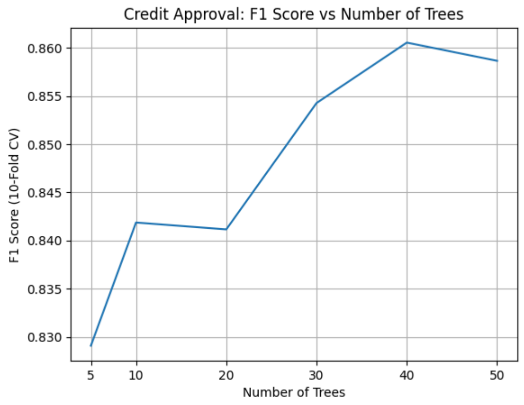
Here, we can observe that even from a quick search for the number of decision trees, the amount of improvement regarding the F1 score changes quite a bit. Although the actual optimal number of trees possibly lies somewhere between 50 and 100 trees (an increase of 50 trees yields an improvement of only about 0.2%), out of the different hyperparameters tested, the 100 tree RF model did the best for both accuracy and F1, and thus would make for the best model to choose. The reason for needing so many trees and not seeing an outright stagnation in performance can be attributed to the fact that the RF has many different sets of data to consider, as opposed to some of the other datasets with only a few hundred rows worth of data.

3-7. Dataset4 - kNN



Unlike in the kNN graph of 3-5, we can see a much more pronounced decline as we increase *k*. Of course, these values will continue to decrease as *k* increases beyond k=51. From this, I would say that the most preferable model is the k=7 model due to achieving stronger results than most other models (particularly in the F1 score) while being a more minimal model than others. I would also consider k values between 3 and 11 to also be acceptable choices for this type of model. When we increase k, the model becomes more generalized but prone to underfitting - it’s less noticeable here primarily due to the testing sizes being relatively small due to the stratified k-folds making testing sets very small.

3-8. Dataset4 - Random Forest



Here, we can observe the results start to stabilize around the 40-50 trees mark. In this case, we can say that although results may improve slightly beyond 50 estimators, we also want to avoid overfitting, in which case the 40 tree model arguably does best, as it has the best accuracy and

F1 metrics than any other model, while being as competitive as the 50 tree model. If one is very concerned about overfitting, I would also consider the 30 tree model as acceptable.

4. Summarize

4-1. Performance of Each Algorithm

	Dataset 1 (Digits)		Dataset 2 (Parkinsons)		Dataset 3 (Rice)		Dataset 4 (Credit)	
	Accuracy	F1-Score	Accuracy	F1-Score	Accuracy	F1-Score	Accuracy	F1-Score
KNN Algorithm	■	■	0.9334	0.9557	0.9073	0.9128	0.8174	0.8939
Random Forest	0.9357	0.9246	0.9179	0.9369	0.9223	0.9207	0.8606	0.8605
Neural Network	0.9972	0.9973	■	■	■	■	■	■

4-2. Which Algorithm had the highest performance

If I have to choose one algorithm, I will choose a random forest because of our implementation result. In general, we found that the Random Forest algorithm tended to give consistently high-quality performance across all datasets. As shown in Table 4-1 and Extra Credit Question 1, it consistently achieved both accuracy and F1-scores above 85%, making it a reliable choice for various applications. Its ensemble structure and ability to handle both numerical and categorical features contributed to its robustness, especially on structured datasets like Credit Approval and Rice.

Neural Networks were also strong contenders. They performed exceptionally well on the Digits dataset, likely due to its high-dimensional, clean, and well-separated pixel features. However, Neural Networks tended to struggle on smaller or noisier datasets like Parkinsons or Credit Approval, demonstrating their sensitivity to sample size and feature complexity. While they can be very powerful, they also require more data and tuning to perform reliably.

K-Nearest Neighbors (KNN), although conceptually simple and intuitive, generally produced the weakest performance. It showed relatively strong results on well-balanced, numeric datasets such as Rice, but suffered more on datasets with class imbalance or mixed feature types, due to its reliance on distance metrics and local neighborhood voting.

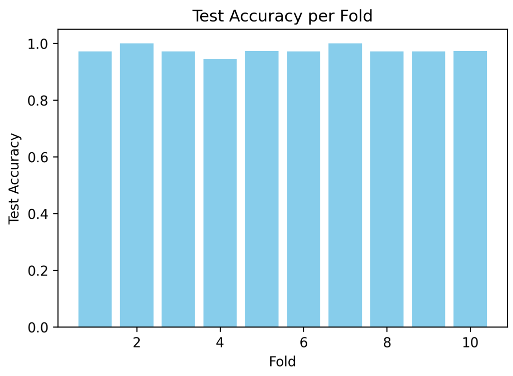
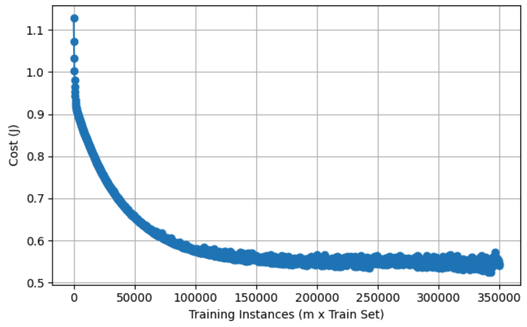
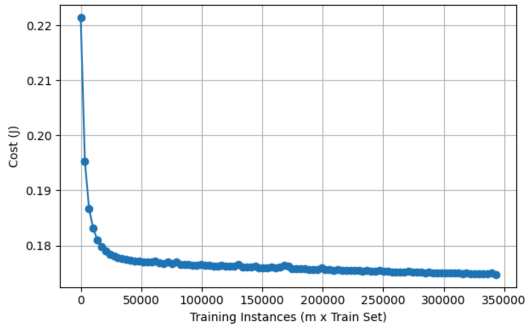
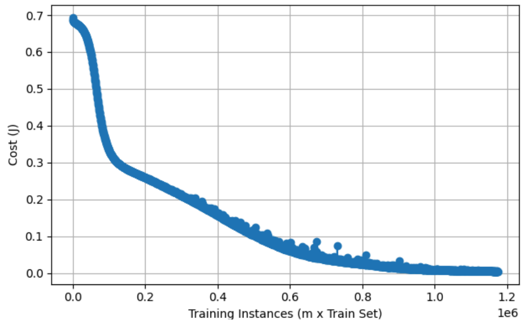
Extra Credits

Extra 1. Additional Algorithm

Etra 1-1. Evaluation

	Dataset 1 (Digits)		Dataset 2 (Parkinsons)		Dataset 3 (Rice)		Dataset 4 (Credit)	
	Accuracy	F1-Score	Accuracy	F1-Score	Accuracy	F1-Score	Accuracy	F1-Score
Decision Tree	0.9751	0.9864						
Random Forest								
Neural Network			0.7541	0.8597	0.9286	0.9378	0.8393	0.8240

Extra 1-2. Cost J/Graph

	Dataset 1 - KNN Algorithm	Dataset 2 - Neural Network
Hyper Param	X	Lambda=0.1, Hidden Layer=[22,64,32,1]
Other Param	Min information gain=0.00001 or Max depth=5	Alpha=0.1, Mini-Batch(32), Stop=m size(2000)
Graph		
	Dataset 3 - Neural Network	**Dataset 4 - Neural Network**
Hyper Param	Lambda=1e-6, Hidden Layer=[64]	Lambda=1e-6, Hidden Layer=[64,32,16,8]
Other Param	Alpha=0.1, Mini-Batch(128), Stop=m size(100)	Alpha=0.1, Mini-Batch(128), Stop=m size(2000)
Graph		

Extra 2. New Challenging Dataset

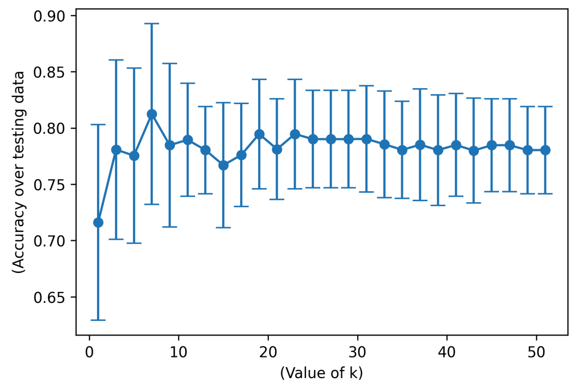
Extra 2-1. Which Dataset and Why

For the Extra Credit task, I chose to work with the UCI Heart Disease dataset because it fully satisfies the criteria specified in the assignment. Most importantly, this dataset features a multi-class target variable, where the label column contains more than two categories (e.g., 0 to 4), each representing a different level of heart disease severity. This aligns with the assignment's requirement that the outcome should not be binary but instead involve multiple classes. In addition to the target variable structure, this dataset includes both numerical and categorical features, such as age, cholesterol level, and maximum heart rate (numerical), as well as chest pain type, sex, and exercise-induced angina (categorical). This variety allows for a comprehensive preprocessing pipeline and testing of classification algorithms under realistic conditions. The dataset also has an appropriate size — 303 samples with 13 features — which is large enough for meaningful analysis using techniques like stratified k-fold cross-validation but small enough to be manageable for building custom models from scratch. Furthermore, the UCI Heart Disease dataset is a widely recognized benchmark in the machine learning community, which adds credibility and comparability to any evaluation performed on it. To check the implementation dataset, go to the "dataset" folder and run "load_heart.py" then you will get the "heart_disease.csv" file to use in the algorithms below.

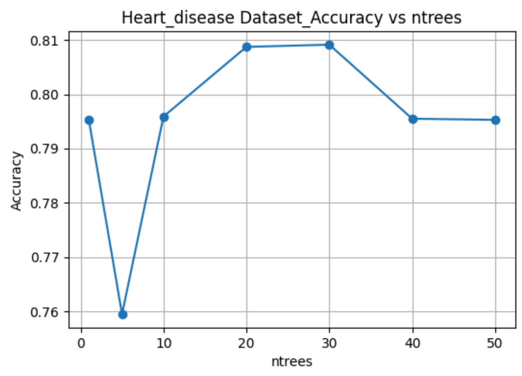
Extra 2-2. Evaluation

	Hyperparameter	Accuracy	F1-Score
KNN Algorithm	k=7	0.8124	0.5411
Random Forest	ntrees=30	0.8091	0.3794

KNN Algorithm



Random Forest



The UCI Heart Disease dataset includes both numerical and categorical features with a multi-class target, making it well-suited for algorithms like K-Nearest Neighbors (KNN) and Random Forest. KNN leverages local similarity, while Random Forest is robust to diverse feature types and works well on moderately sized datasets. Both are interpretable, efficient, and effective without requiring extensive preprocessing. In contrast, a Neural Network was not chosen due to the dataset's small size, which increases the risk of overfitting, and because neural networks demand complex training procedures and hyperparameter tuning. Given the nature of the dataset and the goals of this project, KNN and Random Forest were appropriate choices, while a Neural Network would have been unnecessarily complex. The UCI Heart Disease dataset is small and imbalanced, with most labels being class 0. Although both KNN and Random Forest showed strong overall performance, the class imbalance in the dataset could skew predictions toward the majority class. KNN performed better in terms of F1-score (0.5411) because it captures local patterns and can better detect minority classes. Random Forest achieved similar accuracy (0.8091) but had a lower F1-score (0.3794) due to its tendency to favor the majority class. This shows that KNN handled the class imbalance more effectively for this dataset.

Extra 3. Ensemble Algorithm

Extra 3-1. Algorithm Structure

I designed and implemented a custom ensemble learning algorithm that combines the three algorithms used in the previous sections: Neural Network, Random Forest, and K-Nearest Neighbors (KNN). For each dataset, first perform 10-fold stratified cross-validation. Then, within each fold, obtain model-specific predictions by calling the following three functions: run_knn_single_fold(), run_tree_single_fold(), run_nn_single_fold(). Each of these three models makes independent predictions on the test split. For every test instance, then apply majority voting across the three model predictions to generate the final ensemble prediction. Finally, the ensemble predictions are compared to the true labels to evaluate accuracy and F1 score.

Extra 3-2. Hyperparameter Settings

When run this ensemble algorithm, other dataset accuracy and f1 score are both over 90% except dataset 1. To make an algorithm working well with any dataset, set this algorithm's hyperparameter the same as the best hyperparameter in the previous regular points 2 dataset1 for random forest and neural network. An Imbalanced dataset like dataset 2 needs a small nearest-k value so I choose k=5 which can be a reasonable number for both imbalance and balance dataset.

Extra 3-3. Code Implementation

To run the ensemble algorithm, please navigate to the ensemble_algorithm folder and execute the script en.py.This script automatically runs the ensemble on all four datasets (digits, parkinsons, rice, credit_approval) and outputs accuracy and F1 score for each. The results are saved as Excel files (e.g., ensemble_digits_metrics.xlsx), which can be used for reporting.

Extra 3-4. Evaluation

	Accuracy	**F1-Score**
Dataset 1 (Digits)	0.6750	0.6863
Dataset 2 (Parkinsions)	0.9692	0.9796
Dataset 3 (Rice)	0.9850	0.9869
Dataset 4 (Credit Approval)	0.9234	0.9167

The ensemble algorithm achieved strong performance on most datasets, with over 90% accuracy and F1-score except for the Digits dataset. Digits involves a 10-class classification task with high-dimensional pixel features, which makes it more challenging for KNN and shallow neural networks. In contrast, Parkinsons and Rice datasets are binary classification problems with well-separated numerical features, allowing the ensemble to perform exceptionally well. Even the Credit Approval dataset, which contains mixed-type attributes, was handled effectively thanks to normalization and one-hot encoding, resulting in stable and reliable predictions.

Extra 4. New Type of Algorithm

Extra 4-1. Algorithm Structure

This algorithm implements multinomial linear regression which was chosen over binary logistic regression because the digits dataset contains 10 distinct classes (digits 0 to 9). Binary logistic

regression can only handle two classes, however, multinomial regression learns a separate weight vector for each class and predicts the class with the highest computed score (logit).
It begins by preprocessing the dataset: numeric input variables are normalized, and categorical variables are converted into numerical form through one-hot encoding. The training process uses gradient descent to iteratively update the model's weights and biases. In each iteration (or epoch), the algorithm computes predictions for all training samples, compares them to the true labels, calculates the loss (based on the difference), and adjusts the weights in a direction that minimizes this loss. If the change in loss between iterations becomes negligible, the training stops. The structure of this training process mirrors the classic implementation of linear regression via gradient descent, as outlined in the reference lecture15-slide116. For each weight, the algorithm calculates a gradient and updates it using the learning rate. The code also includes gradient clipping to prevent instability from excessively large updates. After training, the model is evaluated using 10-fold stratified cross-validation to ensure robust performance. For each fold, the model's predictions are compared to the actual labels, and two key metrics are computed: accuracy and macro F1 score, which is appropriate for multi-class evaluation. The average performance across all folds is reported, and the results are saved to a text file.

Extra 4-2. Hyperparameter Settings

Learning_rate = 0.0001 determines the step size taken during gradient descent. A value too high may cause the model to diverge, while a value too low could result in slow convergence. Epochs = 1000 specifies the number of times the entire training dataset is used for updating the model parameters. Higher values can improve learning but also increase training time. epsilon = 1e-6 used as a convergence threshold. If the change in loss between consecutive epochs is smaller than this value, the algorithm stops early to save computation. K_FOLD_SIZE = 10 controls the number of folds in stratified cross-validation. Used 10-fold CV to ensure reliable evaluation, especially on imbalanced datasets, by preserving class distribution in each split.

Extra 4-3. Code Implementation

To run the multinomial linear regression algorithm, navigate to the linear_regression folder and execute the script reg.py.This script automatically runs the ensemble on all four datasets (digits, parkinsons, rice, credit_approval) and outputs accuracy and F1 score for each. The results are saved as text files “linear_regression_results.txt”, which can be used for reporting.

Extra 4-4. Evaluation

	Accuracy	**F1-Score**
Dataset 1 (Digits)	0.9310	0.9235
Dataset 2 (Parkinsons)	0.7450	0.4259
Dataset 3 (Rice)	0.9152	0.9137
Dataset 4 (Credit Approval)	0.7978	0.7868

In the evaluation of the multinomial linear regression model across four datasets, we observed varying levels of performance depending on the nature of each dataset.
For the Digits dataset, the model achieved excellent results, with an accuracy of 93.10% and an F1 score of 92.35%. This dataset contains clean, high-dimensional pixel features that represent handwritten digits from 0 to 9. The class boundaries are well-separated, and the data is

balanced, making it an ideal case for a linear model. As a result, the model was able to generalize effectively and consistently classify digits across all classes.

In contrast, the Parkinsons dataset showed relatively lower performance, especially in terms of F1 score. While the accuracy reached 74.50%, the F1 score dropped to 42.59%. This large gap suggests that the dataset may be imbalanced or that some classes are harder to distinguish. Given the small size and subtle variations in features, the model likely struggled with false negatives, leading to lower recall. This demonstrates a limitation of using simple linear models on small and noisy biomedical datasets.

The Rice dataset exhibited strong results, with an accuracy of 91.52% and an F1 score of 91.37%. This dataset is known to be clean, structured, and linearly separable, which aligns well with the assumptions of linear regression. The high and balanced scores indicate that the model performed well across both classes and was not biased toward either.

Lastly, for the Credit Approval dataset, the model showed solid generalization despite challenges. This dataset includes a mix of numeric and categorical variables, and it may contain noise or missing values. Nonetheless, the model achieved an accuracy of 79.78% and an F1 score of 78.68%. These results suggest that the preprocessing steps—such as normalization and one-hot encoding—effectively prepared the data, allowing the model to handle the complexity and make reliable predictions.

Overall, the evaluation highlights that the performance of multinomial linear regression is strongly influenced by the structure, balance, and clarity of the dataset. Clean and separable datasets like Digits and Rice yield high scores, while more ambiguous or imbalanced datasets like Parkinsons present challenges that linear models alone cannot fully overcome.

In addition to class imbalance, the relatively low F1-score on the Parkinsons dataset can also be attributed to the inherent limitations of multinomial linear regression. As a linear model, it assumes that the decision boundaries between classes are linearly separable, which may not hold true in biomedical datasets like Parkinsons that contain subtle and nonlinear patterns in voice features. These nonlinear interactions cannot be captured by a linear hypothesis, leading to misclassifications, particularly in borderline or overlapping cases.