COMPSCI 589 - Final Project - Spring 2024

Due May 10, 2024, 11:55 pm Eastern Time

1 Instructions

- **This final project should be done in groups of two students**. When making your submission on Gradescope, please do not forget to add all students who worked on this assignment. Do that both when submitting your code and your final report.
- We strongly recommend that you use LaTeX to prepare your submission. The assignment should be submitted on Gradescope as a PDF with marked answers via the Gradescope interface. The source code should be submitted via the Gradescope programming assignment as a .zip file. Include with your source code instructions for how to run your code.
- You may *not* use any machine learning-specific libraries in your code, e.g., TensorFlow, PyTorch, or any machine learning algorithms implemented in scikit-learn. You may use libraries like numpy and matplotlib. If you are not certain whether a specific library is allowed, do ask us.
- All submissions will be checked for plagiarism using two independent plagiarism-detection tools. Renaming variable or function names, moving code within a file, etc., are all strategies that *do not* fool the plagiarism-detection tools we use. If you get caught, all penalties mentioned in the syllabus *will* be applied—which may include directly failing the course with a letter grade of "F".
- The tex file for this assignment (which you can use if you decide to write your solution in LaTeX), as well as the datasets, can be found here.
- The automated system will not accept assignments after 11:55pm on May 10.
- Notice that you **cannot** use your "free late days" on this assignment.

2 Final Project — Overall Goals

- The goal of this final project is to compare the performance of different machine learning algorithms you implemented throughout the semester by testing them on four new, challenging datasets.
- By performing these comparisons, you will gain practical experience in selecting an algorithm (based on a given problem) and optimizing its hyper-parameters so that it works well. In addition, these analyses will give you insight into how to solve novel machine learning problems—problems where no one tells you which algorithms may work best nor how to adjust their hyper-parameters.
- **This project should be done in groups of two students**.
- **Notice that you may not use existing machine learning libraries for this assignment: you *must* use the implementations that you or your partner designed throughout the semester**.
- On each question below, please clearly indicate whose implementation was used; i.e., make sure that you explicitly mention the name of the student whose code was used to tackle each particular dataset.
- We expect the amount of time and effort you put into this assignment will be slightly *less* than that required to solve one homework. The reason is that this final project will not require implementing new algorithms, other than the ones you already implemented during the semester. Also, you will be working in groups and so it should be easier to split up tasks evenly.

- **Alternative: "Custom Project"**. You may replace the tasks mentioned above with a "custom project" that is more closely aligned with your interests. It has to be ML-related and use some of the techniques we discussed in class. For reasons of fairness, this project *cannot* be used as part of any other UMass activity for which you are given credit (e.g., other courses or independent studies). A custom project related to a research project that you are conducting with a UMass professor (or with anyone else) is acceptable if you are not already getting graded for it. If you would like to work on a custom project, please check with the instructor (bsilva@cs.umass.edu) if the custom project you have in mind would be acceptable. If the project is deemed acceptable, you should let the instructor know, *before you start working on the project*, which tasks will be assigned to each student in the group.

3 Datasets

As part of this final project, you will be analyzing four datasets.

3.1 The Hand-Written Digits Recognition Dataset

The goal, here, is to analyze 8x8 pictures of hand-written digits (see, e.g., Fig. 1) and classify them as belonging to one of 10 possible classes. Each class is associated with one particular digit: $0, 1, \ldots, 9$. In this dataset, each instance is composed of $8 \times 8 = 64$ numerical attributes, each of which corresponding to the grayscale value of one pixel in the image being classified. This dataset is composed of 1797 instances.

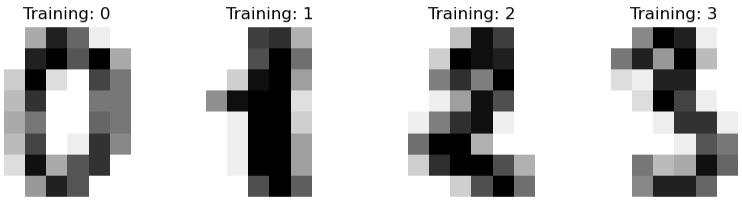


Figure 1: Examples of hand-written digits that you will be classifying.

To access this dataset, and for sample code describing how to load and pre-process it, please visit Scikit-learn's website. Further, please find below (in Fig. 2) a simple example of how to load this dataset; here, the instances are saved in *digits_dataset_X* and their corresponding classes are saved in *digits_dataset_y*.

```
from sklearn import datasets
import numpy as np
import matplotlib.pyplot as plt

digits = datasets.load_digits(return_X_y=True)
digits_dataset_X = digits[0]
digits_dataset_y = digits[1]
N = len(digits_dataset_X)

# Prints the 64 attributes of a random digit, its class,
# and then shows the digit on the screen
digit_to_show = np.random.choice(range(N),1)[0]
print("Attributes:", digits_dataset_X[digit_to_show])
print("Class:", digits_dataset_y[digit_to_show])

plt.imshow(np.reshape(digits_dataset_X[digit_to_show], (8,8)))
plt.show()
```

Figure 2: Example of how to load the Digits dataset from Scikit-learn, and how to show information about a random instance/digit in the dataset.

3.2 The Titanic Dataset

On April 15, 1912, the largest passenger liner ever made collided with an iceberg during her maiden voyage. When the Titanic sank, it killed 1502 out of 2224 passengers and crew. One of the reasons the shipwreck resulted in such loss of life was that there were not enough lifeboats for the passengers and crew. Although luck was involved in surviving the sinking, some groups of people were more likely to survive than others.

The goal, here, is to predict whether a given person was likely to survive this tragedy. The dataset contains data corresponding to 887 real passengers of the Titanic. Each row represents one person. The columns describe different attributes of the person, including whether they survived, their age, their passenger class, sex, and the fare they paid. The target class (*Survived*) is encoded in the first column of the dataset. The Titanic dataset can be found in this assignment's zip file. Notice that this dataset combines both numerical and categorical attributes.

3.3 The Loan Eligibility Prediction Dataset

Here, the goal is to automatically (and accurately) predict whether a given person should qualify for a loan. Each instance is described by 13 attributes, including the Loan_ID, the gender of the applicant, whether the applicant is married, their income, information about their credit history, etc. The binary target class to be predicted is *Loan_Status*: whether a given applicant's request for a loan was approved or not. Notice that this dataset contains 8 categorical attributes, 4 numerical attributes, and a Loan_ID attribute. There is a total of 480 instances in the dataset. The Loan dataset can be found in this assignment's zip file.

3.4 The Oxford Parkinson's Disease Detection Dataset

This dataset is composed of a range of biomedical voice measurements from 31 people, 23 of whom are patients with Parkinson's disease. Each row in the dataset corresponds to the voice recording from one of these individuals. Each attribute corresponds to the measurement of one specific property of the patient's voice—for example, their average vocal frequency or several measures of frequency variation. The goal, here, is to predict whether a particular person is healthy or whether it is a patient with Parkinson's. The binary target class to be predicted is *Diagnosis*: whether the patient is healthy (class 0) or a patient with Parkinson's (class 1). There are 195 instances in this dataset. All 22 attributes are numerical. The Parkinson's dataset can be found in this assignment's zip file.

⋆ Before running experiments to analyze each dataset above, manually verify if it makes sense to consider all features/attributes. E.g., should the learning algorithm be given the identifier of a loan? Or the name of a person? If you choose to remove one or more attributes, explain why you decided to do so in your report.

⋆ Notice that some of the datasets above include both categorical and numerical attributes. Algorithms such as neural networks (as well as others) require numerical inputs. The standard way of converting categorical inputs to numerical inputs is by using by one-hot encoding technique. For a quick and high-level introduction to one-hot encoding, as well as examples on how to use Scikit's libraries to perform this type of conversion, please visit this website.

4 Experiments and Analyses

For each dataset, you should:

1. Evaluate the performance of at least two algorithms you studied and/or implemented during the semester (e.g., k-NN, Decision Trees, standard Naive Bayes, Random Forests, Neural Networks, etc).[1] You should discuss which algorithms you decided to test on each dataset and why.
2. You should evaluate the performance of each algorithm on a given dataset in terms of its accuracy and F1 score. Use stratified cross-validation with $k = 10$.
3. To obtain the best performance possible, you should carefully adjust the *hyper-parameters* of each algorithm when deployed on a dataset. For example, you may have to experiment with different values of k when optimizing the k-NN algorithm; different values of *ntree* when optimizing a Random Forest; different architectures and regularization parameters when optimizing a neural network; etc.
4. Given the observation above, you should first show, in a table, the performance of each algorithm (on a given dataset) under a few selected hyper-parameters. You should evaluate at least 3 hyper-parameter settings. After analyzing the performance of each algorithm under different hyper-parameters, identify the best hyper-parameter setting—that is, the set of hyper-parameters that resulted in the best performance for the corresponding algorithm on a particular dataset.
5. For each dataset, and considering the best hyper-parameter setting for each selected algorithm, construct relevant learning curves and/or graphs. These should be similar to the learning curves/graphs you constructed in the homework relevant to the particular algorithm you are evaluating. For example, if you choose to deploy a Random Forest to tackle one of the datasets, construct a graph relating its performance and the number of trees in the ensemble; if you choose to use a neural network, construct a learning curve showing the value of the cost function, J, as a function of the number of training instances presented to the network. Assuming that you evaluate two algorithms on each of the four datasets, these analyses should result in (at least) 8 graphs. Briefly discuss and interpret these graphs. For example: Why do you think a particular algorithm may be converging to a poor local optimum when deployed on a given dataset? Why do you think a specific algorithm performs better on datasets containing only numerical attributes? and so on.

After conducting the analyses above for each dataset, you should summarize your results in a table. In particular, you should create a table showing, for each dataset, the performance (accuracy and F1-score) of each of the algorithms. Your table should look like Table 1. You should highlight (in gray) the cells that indicate which algorithm had the highest performance—according to each of the metrics—on each dataset. In the example shown in Table 1, for instance, we can see that, when analyzing Dataset 1, Algorithm 1 had the highest accuracy but Algorithm 2 had the highest F1 score.

	Dataset 1		Dataset 2		Dataset 3		Dataset 4	
	Accuracy	F1-Score	Accuracy	F1-Score	Accuracy	F1-Score	Accuracy	F1-Score
Algorithm 1								
Algorithm 2								
Algorithm 3								
...								

Table 1: Example of how the table summarizing your final results look like.

To facilitate the task of creating this table on LaTeX, please visit this website. We have created a template for generating tables similar to the one shown above. This template can be found in the *table_results.tgn* file, provided to you as part of this assignment's zip file. You can load this template on the website mentioned above (File → Load Table) and complete the table with your results.

[1] Evaluating more algorithms will give you extra credit. See Section 5.

4.1 The Hand-Written Digits Recognition Dataset

The goal, here, is to analyze 8x8 pictures of hand-written digits (see, e.g., Fig. 1) and classify them as belonging to one of 10 possible classes. Each class is associated with one particular digit: $0, 1, \ldots, 9$. In this dataset, each instance is composed of $8 \times 8 = 64$ numerical attributes, each of which corresponding to the grayscale value of one pixel in the image being classified. This dataset is composed of 1797 instances.

SOLUTION : You should discuss which algorithms you decided to test on each dataset and why.

Algorithm 1: Decision Tree

1. The input features in this dataset are not directly interpretable as each column contains that image instance's pixels. However, we believe, that decision trees can still provide insights into the classification process. How? Each decision node in the tree corresponds to a pixel, and the path from the root to a leaf node represents a sequence of decisions based on pixel values. This seemed highly interpretability to us and like it would be a good choice to choose this particular algorithm for the dataset.
2. Another reason why Decision tree seemed like a great idea was because decision trees automatically select the most informative features (pixels in this case) during training, effectively filtering out irrelevant features (columns with all zeros except for one) and focusing on those that contribute most to the classification task.
3. Decision trees can serve as the base learners in ensemble methods like random forests, which can further improve classification accuracy by aggregating multiple decision trees. Which brings us to our next best algorithm choice:

Algorithm 2: Random Forest

1. Since each decision tree in the Random Forest is trained on a bootstrapped subset of the data plus a random subset of features, this approach tends to produce models with better generalization performance compared to individual decision trees. Which our code output proves. This enhanced generalization capability is particularly beneficial for the Handwritten Digits Recognition Dataset, where maintaining a balance between model complexity and generalization is critical because of its relatively small size of the dataset and the potential noise in handwritten digit images.
2. In our dataset, making the handwritten digit's boundaries clear is the most important. A slight deviation in pixel intensity, particularly if a wrong pixel carries significant magnitude, could distort the image, rendering the numbers indecipherable. Random Forests are good at capturing intricate decision boundaries and understanding the nuanced relationships among features and classes makes them ideally suited for this task, ensuring accurate classification even amidst potential image imperfections.

Algorithm 3: Neural Networks

1. Observing the dataset makes it obvious that we dont have any features as such, they are simply pixel magnitudes. Unlike traditional machine learning algorithms that often require proper informative features, neural networks can learn relevant features directly from raw data. This is great for image data, as it eliminates the need for manual feature engineering and allows the model to adapt to different writing styles and variations in digit appearance.
2. Neural networks excel at capturing non-linear relationships between input features and target classes. Handwritten digits can exhibit complex and non-linear patterns, and neural networks are adept at modeling such relationships through their multiple layers of neurons and activation functions.

Algorithms that were not chosen and why.

1. As mentioned earlier, this dataset comprises only 16 pixel magnitudes. KNN heavily relies on distance metrics to classify data points. However, for high-dimensional data like images, where each pixel represents a dimension, defining distance becomes challenging. We believe, KNN's effectiveness diminishes in such scenarios, as it might struggle to decipher patterns that actually result in the number.

2. Multinomial Naive Bayes, on the other hand, assumes independence between features, which is simply not true for pixel values in image data.

4.2 The Titanic Dataset

On April 15, 1912, the largest passenger liner ever made collided with an iceberg during her maiden voyage. When the Titanic sank, it killed 1502 out of 2224 passengers and crew. One of the reasons the shipwreck resulted in such loss of life was that there were not enough lifeboats for the passengers and crew. Although luck was involved in surviving the sinking, some groups of people were more likely to survive than others.

The goal, here, is to predict whether a given person was likely to survive this tragedy. The dataset contains data corresponding to 887 real passengers of the Titanic. Each row represents one person. The columns describe different attributes of the person, including whether they survived, their age, their passenger class, sex, and the fare they paid. The target class (*Survived*) is encoded in the first column of the dataset. The Titanic dataset can be found in this assignment's zip file. Notice that this dataset combines both numerical and categorical attributes.

SOLUTION : You should discuss which algorithms you decided to test on each dataset and why.
Algorithm 1: Decision Tree

1. Decision trees provide a clear, interpretable model that can be easily understood. Thus, it important to give proper insights about the factors (attributes) contributed towards the survival rate on the Titanic, such as passenger class, age, sex, and fare paid.
2. Decision Tree thus, helps in knowing which particular feature contributed towards making the important decision regarding survival of the people. For example, it can tell us that a particular class of people were the ones who survived the most and infact can also tell us why a particular gender of a particular class survived the most. This helps in classifying the unseen data easily and in a traceable way.

Algorithm 2: Random Forest

1. As we know Random forest contributes towards a higher accuracy than a single decision tree as there are multiple decision trees we are working on which converge together to give an average prediction. This helps in reducing the overfitting which might happen due to the decision tree. Thus, random forest is expected to give more accurate prediction of the survivals.
2. Random forest also gives importance to the important features right from the start. For example, in our dataset, it directly computes if a feature, let's say, 'Pclass' acts as an important attribute which separates the dataset and might result into a category which gives us homogeneous result as compared to any other attribute which might also be adding a lot of value. Random forest chooses the attribute which helped it to learn or gain the most information and thereby result in more informed decision making.

Algorithm 3: Neural Networks

1. In the Titanic dataset, there are some interactions between features like age, gender, and passenger class (Pclass) that influence survival. Neural Networks understands these patterns and also finds an important relationship between the input features and target variable, here, Survival.
2. Neural Networks also contributes towards automatic feature learning. There are many features which are easy to understand and don't have a complex relationship whereas, there are some other features which have a complex relationship. This model finds a way to understand the pattern between these attributes and the target class which might be an important information loss if not understood or taken into consideration.

Algorithms that were not chosen and why.

1. The titanic dataset comprises of 8 different attributes. KNN heavily relies on distance metrics to classify data points. KNN considers all the features equally when calculating the distances between the datapoints. There are several features in the titanic dataset which hold no relevant value and are noisy (for e.g., Name, Fare). KNN considers these attributes and ends up getting stuck while computing between these attributes.

2. Naive Bayes assumes that features are independent of each other given the class label. However, features in the Titanic dataset may have dependencies or correlations. For example, the likelihood of survival may depend on a combination of factors such as gender, passenger class, age, etc. If we ignore these correlations, it might not result into accurate outputs due to loss of information.

4.3 The Loan Eligibility Prediction Dataset

Here, the goal is to automatically (and accurately) predict whether a given person should qualify for a loan. Each instance is described by 13 attributes, including the Loan_ID, the gender of the applicant, whether the applicant is married, their income, information about their credit history, etc. The binary target class to be predicted is *Loan_Status*: whether a given applicant's request for a loan was approved or not. Notice that this dataset contains 8 categorical attributes, 4 numerical attributes, and a Loan_ID attribute. There is a total of 480 instances in the dataset. The Loan dataset can be found in this assignment's zip file.

SOLUTION : You should discuss which algorithms you decided to test on each dataset and why.

Algorithm 1: Decision Tree

1. Decision trees offers a simple way to understand the importance of different attributes in predicting loan eligibility. By analyzing the tree structure, we can identify which attributes (such as credit history, income, marital status) are most influential in determining loan approval status.
2. Decision trees are also relatively efficient to train and the Loan Eligibility Prediction dataset having 480 instances acted like a suitable fit. While larger datasets may require more sophisticated algorithms, this dataset was an optimal choice to be run on the decision tree algorithm. Which brings us to our next best algorithm choice:

Algorithm 2: Random Forest

1. By analyzing feature importance scores calculated based on how much each feature contributes to the reduction in impurity (e.g., Gini impurity), we can identify which attributes (such as credit history, income, marital status) are most influential in determining loan approval status.
2. Random Forests can handle categorical attributes naturally without requiring one-hot encoding or other preprocessing steps. With 8 categorical attributes in the Loan Eligibility Prediction dataset, this simplifies the modeling process and reduces the risk of introducing biases or errors during preprocessing.

Algorithm 3: Neural Networks

1. The loan eligibility decision likely depends on intricate relationships between various factors such as income, credit history, marital status, and other attributes. Neural networks can capture these complex, non-linear relationships effectively, allowing for more accurate predictions of loan approval status.
2. If we look at the loan eligibility dataset, there are several attributes which are complex to connect and understand due to it being a categorical attribute. There are several patterns which can be determined, which is understood by the neural networks very efficiently.

Algorithms that were not chosen and why.

1. When you have more attributes (like age, income, credit history) in a dataset, KNN might not work well. For the loan prediction dataset, there are 13 attributes which is quite a lot. KNN calculates distances between data points to make predictions, but with many attributes, it becomes harder to find similar points. This can make KNN less accurate, especially if the dataset isn't very big.
2. Multinomial Naive Bayes works well when dealing with categorical data, like counting the frequency of words in text. However, the Loan Eligibility Prediction dataset has both categorical (like gender or marital status) and numerical (like income or credit history) attributes. Turning numerical data into categories could cause important details to be lost, which might make Multinomial Naive Bayes less accurate for this dataset.

4.4 The Oxford Parkinson's Disease Detection Dataset

This dataset is composed of a range of biomedical voice measurements from 31 people, 23 of whom are patients with Parkinson's disease. Each row in the dataset corresponds to the voice recording from one of these individuals. Each attribute corresponds to the measurement of one specific property of the patient's voice—for example, their average vocal frequency or several measures of frequency variation. The goal, here, is to predict whether a particular person is healthy or whether it is a patient with Parkinson's. The binary target class to be predicted is *Diagnosis*: whether the patient is healthy (class 0) or a patient with Parkinson's (class 1). There are 195 instances in this dataset. All 22 attributes are numerical. The Parkinson's dataset can be found in this assignment's zip file.

SOLUTION : You should discuss which algorithms you decided to test on each dataset and why.

Algorithm 1: Decision Tree

1. Decision trees offer transparency along with interpretability, both vital for medical diagnosis tasks like Parkinson's disease detection. In this dataset, voice measurements are the features. For clinicians to easily interpret the decision rules of a tree, understanding which vocal attributes contribute most to classifying patients as healthy or diagnosed with Parkinson's disease is most important. Random Forest aids to this interpretability. We can understand the relevant biomarkers and understand the diagnostic process.

Algorithm 2: Random Forest

1. Biomedical datasets often contain noise and variability due to factors like measurement errors or individual differences among patients. Random Forests are inherently robust to such challenges. By aggregating predictions from multiple trees, Random Forests can effectively filter out noise and capture the underlying patterns in the data, resulting in more reliable predictions of Parkinson's disease diagnosis. This robustness is particularly valuable for ensuring accurate and consistent classifications in medical applications.
2. our dataset involves numerous features representing different aspects of patient health. Random Forests excel in handling high-dimensional data by automatically selecting relevant features and capturing complex interactions among them. In the case of voice measurements, which can include multiple attributes such as vocal frequency and frequency variation, Random Forests can efficiently analyze these diverse features to discern patterns indicative of Parkinson's disease, thus offering a powerful tool for classification.

Algorithm 3: Neural Networks

1. This dataset comprises various voice measurements, neural networks can leverage this hierarchical representation learning to capture nuanced relationships between different vocal attributes and disease status. Neural networks can automatically learn to recognize patterns indicative of Parkinson's disease across multiple levels of abstraction, from fundamental voice characteristics like pitch and intensity to more complex variations in vocal modulation and articulation.
2. Moreover, neural networks excel at identifying non-linear relationships within the data. Parkinson's disease diagnosis often involves intricate interdependencies among different voice measurements, which may not be adequately captured by linear models or simpler algorithms. Neural networks' capacity to model non-linear relationships allows them to capture the complex interactions between voice features, uncovering hidden correlations that might escape detection by traditional methods.

Algorithms that were not chosen and why.

1. KNN relies heavily on distance metrics to classify data points, making it less suitable for high-dimensional data like the voice measurements in the Parkinson's disease dataset. In high-dimensional spaces, the notion of distance becomes less meaningful, leading to degraded performance. Additionally, KNN's performance can suffer in the presence of noise or variability in the data, which is common in biomedical datasets..

2. Voice measurements are likely correlated, and Naive Bayes may struggle to capture these dependencies effectively, leading to suboptimal performance in classifying patients as healthy or diagnosed with Parkinson's disease..

4.5 The Hand-Written Digits Recognition Dataset

SOLUTION : 2. You should evaluate the performance of each algorithm on a given dataset in terms of its accuracy and F1 score. Use stratified cross-validation with k = 10. 3. To obtain the best performance possible, you should carefully adjust the hyper-parameters of each algorithm when deployed on a dataset. For example, you may have to experiment with different values of k when optimizing the k-NN algorithm; different values of ntree when optimizing a Random Forest; different architectures and regularization parameters when optimizing a neural network; etc

Algorithm 1: Decision Tree

1. HYPERPARAMETERS.

   $n_iterations = 100$

   This hyperparameter determines the number of decision trees to be created in the ensemble. In this case, the decision tree algorithm will generate 100 individual decision trees. Increasing the number of iterations can potentially improve the performance of model by reducing overfitting and increasing robustness.

   $max_depth = 100$

   A decision tree's depth corresponds to the number of levels it can have, with each level representing a decision or split based on a feature. Setting max_depth to 100 means that decision tree has a maximum depth of 100 levels. Deeper trees can capture more complex relationships in the data but may also increase the risk of overfitting. For us, 100 depth proved to be beneficial as anything below that gave lesser accuracy fbecause of underfitting and anything above that overfitted.

   $minsamplessplit = 2$

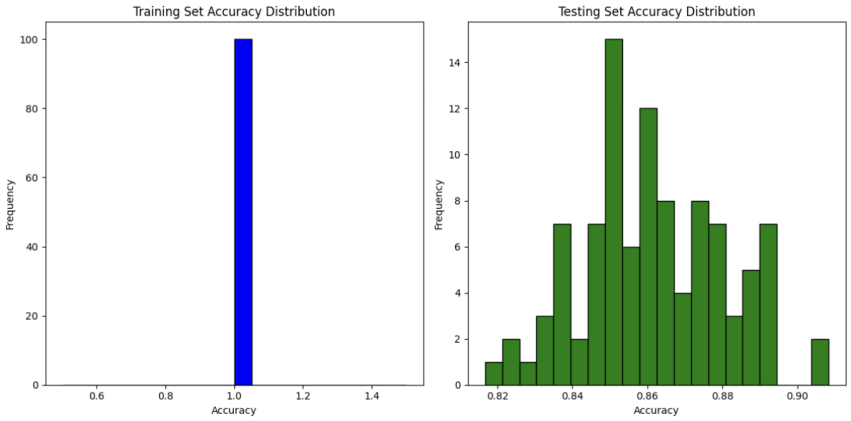
   This hyperparameter specifies the minimum number of samples required to split an internal node in a decision tree within the Random Forest ensemble. Setting $min_samples_split$ to 2 means that a node will only be split if it contains at least 2 samples. A low value for $min_samples_split$ can result in decision trees with finer granularity.

   $nfeatures = None$

   This hyperparameter determines the number of features to consider when looking for the best split at each node in each decision tree within the Random Forest ensemble. By default, $max_features$ is set to None, meaning that all features will be considered for each split. This also happened to give a good accuracy so it was not modified.

Max depth	No. of iterations	Accuracy
100	100	86.19
10	100	72.35
16	100	69.87
100	50	81.18
10	50	50.88
16	50	59.31

- The histogram for training set accuracy shows a single bar at 100%, indicating that the algorithm consistently achieves perfect accuracy on the training data across multiple runs. This suggests that the algorithm has learned to perfectly fit the training data, potentially indicating overfitting.
- The histogram for testing set accuracy shows a distribution of accuracies ranging from approximately 82% to 92%, with an average accuracy of 86.19% and a standard deviation of 0.0191. This



distribution suggests variability in the algorithm's performance on unseen data across different runs.

- The discrepancy between the high training accuracy and the lower testing accuracy suggests that the algorithm may be overfitting to the training data. While the algorithm achieves perfect accuracy on the training set, it fails to generalize well to unseen data, resulting in lower accuracy on the testing set.

Algorithm 2: Random Forest

1. HYPERPARAMETERS.

   $n_trees = 10$ For this dataset, number of trees when set to 10 gave highest accuracy.

   $n_splits = 2$

   $max_depth = 100$

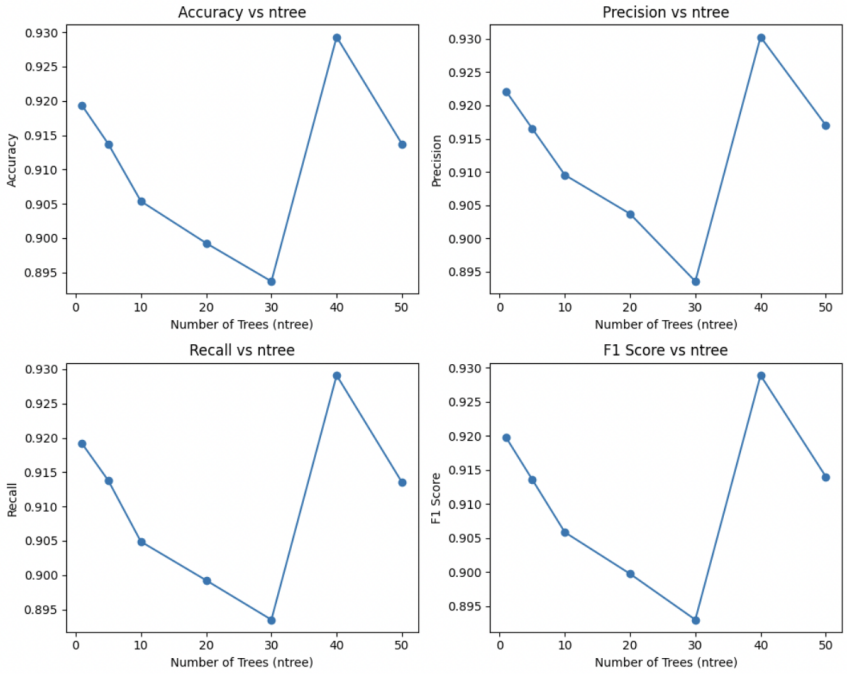
   $min_samples_split = 10$

   This hyperparameter specifies the minimum number of samples required to split an internal node in a decision tree within the Random Forest ensemble. Setting $min_samples_split$ to 10 means that a node will only be split if it contains at least 10 samples.

   $n_features = None$

   This hyperparameter determines the number of features to consider when looking for the best split at each node in each decision tree within the Random Forest ensemble. By default, $max_features$ is set to None, meaning that all features will be considered for each split. This also happened to give a good accuracy so it was not modified.

Number of trees	Accuracy	F1 score
1	90.98	91.04
5	90.37	90.35
10	91.20	91.19
20	91.20	91.21
30	89.87	89.88
40	89.32	89.31
50	90.20	90.33



- The graph illustrating the relationship between the number of trees in the Random Forest ensemble and various performance metrics provides valuable insights into the behavior of the algorithm.
- Across different numbers of trees, there are fluctuations in the performance metrics, indicating that the choice of the number of trees impacts the model's performance.
- Notably, the performance metrics such as accuracy, precision, recall, and F1 score exhibit a trend where the highest values are not consistently associated with the highest number of trees. For instance, while the average accuracy peaks at 10 trees (0.912), it gradually decreases with more trees.

Algorithm 3: Neural Networks

1. HYPERPARAMETERS.

   $architectures = [64, 64, 32, 10]$

   This hyperparameter determines the number of hidden layers in our neural network. In this case, we have chosen 2 hidden layers with 64 and 32 neurons each which gives us the best performance. We came to this conclusion by trying out various architectures where we started with a single hidden layer having (64) neurons. While we got an accuracy of around 84%, it wasn't increasing despite changing the other 2 hyperparameters. Thus, we tried another architecture with [64,64] hoping that increasing the number of layers will contribute towards a better accuracy. This resulted in a decrease in performance. Further decrease and increase in the number of hidden layers also resulted in poor accuracy signifying that the architecture was too complex to solve this problem. Thus we decreased the number of neurons in the 2nd hidden layer which gave us the best architecture as further decrease resulted in a even more poor accuracy.

   $mlearning_rate = 1$

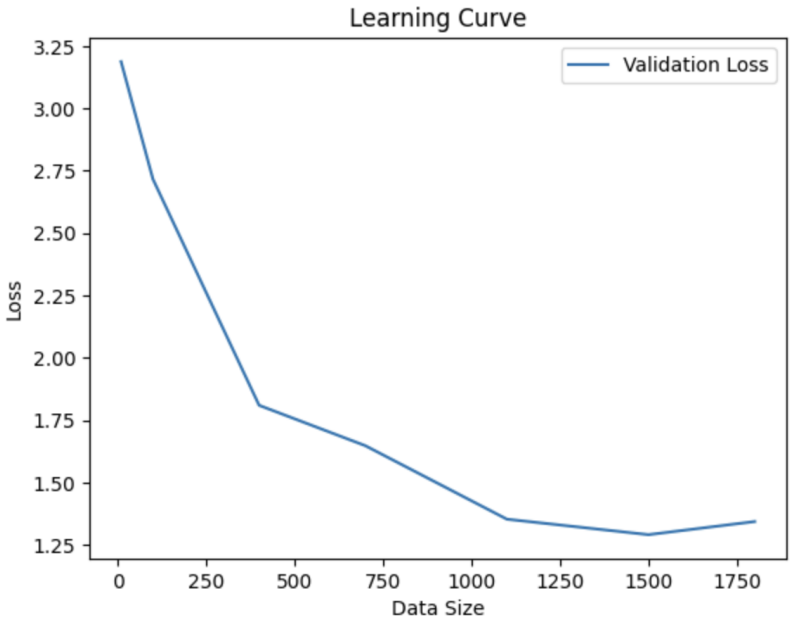
   Learning rate determines how much or how well we want our model to learn from the training examples. Different values were tested for this, which included [0.001,0.1,0.3,0.8,0.9,1,1.05]. However for the different architectures that we studied, there was a lot of learning required by this model. Hence, 1 stood as the winner amongst other learning rates. $lambda = 0.09$

   This hyperparameter helps to reduces the overfitting of the data. Different values of lambda [0.09, 0.1 ,0.15, 0.2] were tested for different learning rates over different architectures. Best accuracy was obtained at a very low regularization of 0.09.

Neural Network Architecture	Learning rate	Lambda (Regularization)	epochs	Accuracy	F-1 score
64,64,10	1	0.09	200	0.8947	0.8812
64,64,32,10 (BEST)	1	0.09	200	0.9234	0.9152
64,32,32,10	1	0.09	200	0.7790	0.7710
64,32,10	0.9	0.09	200	0.8126	0.8102
64,64,64,10	0.9	0.09	200	0.8823	0.8936
64,64,32,10	0.01	0.09	200	0.9034	0.8947

- The graph illustrating the relationship between the data sizes in the Neural Network and loss.
- Across different numbers of data sizes used of training the neural network, there is a pattern observed, indicating that the choice of the number of datapoints used for training impacts the model's loss.
- We can thus say, that more the number of data sizes used for training, less is the loss as the model learns how to predict more accurately.
- Thus, there is a decrease in loss for data sizes more than, 500 and so on towards the end of the datapoints.

- Implementation
- Decision Tree - Mitali Juvekar
- Random Forest - Mitali Juvekar
- Neural Network - Mehak Nargotra



4.6 Titanic Dataset

SOLUTION : 2. You should evaluate the performance of each algorithm on a given dataset in terms of its accuracy and F1 score. Use stratified cross-validation with k = 10. 3. To obtain the best performance possible, you should carefully adjust the hyper-parameters of each algorithm when deployed on a dataset. For example, you may have to experiment with different values of k when optimizing the k-NN algorithm; different values of ntree when optimizing a Random Forest; different architectures and regularization parameters when optimizing a neural network; etc

Algorithm 1: Decision Tree

1. HYPERPARAMETERS.

   $n_iterations = 100$

   This hyperparameter determines the number of decision trees to be created in the ensemble. In this case, the decision tree algorithm will generate 100 individual decision trees. Increasing the number of iterations can potentially improve the performance of model by reducing overfitting and increasing robustness.

   $max_depth = 6$

   With a lower max_depth, decision trees are constrained to simpler decision boundaries. The Titanic dataset, which contains relatively straightforward patterns such as gender, age, ticket class, and whether passengers survived, a simpler decision boundary suffices to capture the main predictive relationships.

   $minsamplessplit = 2$

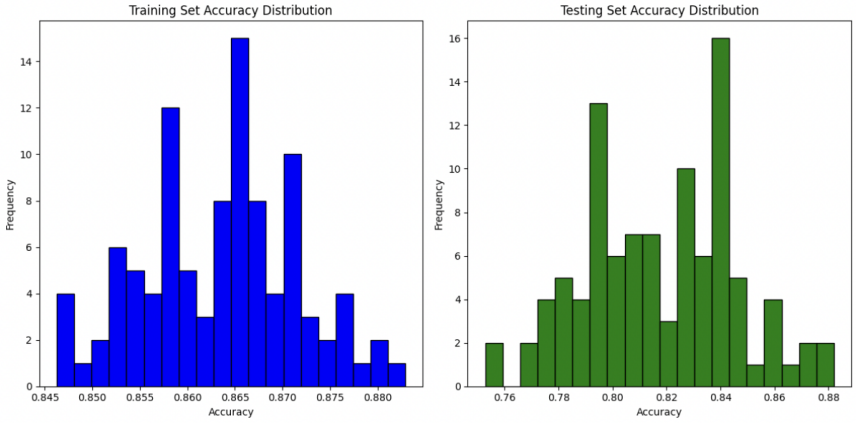
   This hyperparameter specifies the minimum number of samples required to split an internal node in a decision tree within the Random Forest ensemble. Setting $min_samples_split$ to 2 means that a node will only be split if it contains at least 2 samples. A low value for $min_samples_split$ can result in decision trees with finer granularity.

   $nfeatures = None$

   This hyperparameter determines the number of features to consider when looking for the best split at each node in each decision tree within the Random Forest ensemble. By default, $max_features$ is set to

None, meaning that all features will be considered for each split. This also happened to give a good accuracy so it was not modified.

Max depth	No. of iterations	Accuracy
6	100	81.71
10	100	76.35
50	100	76.87
6	50	80.44
10	50	60.43
50	50	40.77



- The histograms illustrate the distribution of accuracy scores for both the training and testing sets, providing insights into the performance of the machine learning algorithm.
- For the training set accuracy distribution, the histogram displays a relatively narrow distribution centered around an average accuracy of 0.8634. The small standard deviation of 0.0081 indicates that the algorithm's performance on the training data is relatively consistent across multiple runs.
- In contrast, the testing set accuracy distribution exhibits a wider spread of accuracy scores, ranging from approximately 0.75 to 0.90, with an average accuracy of 0.8174 and a larger standard deviation of 0.0276. This indicates greater variability in the algorithm's performance on unseen data compared to the training set.
- The discrepancy between the training and testing set accuracy distributions suggests that the algorithm may be experiencing challenges in generalizing well to unseen data. While it performs consistently well on the training data, its performance on new, unseen data is less stable and more variable.

Algorithm 2: Random Forest

1. HYPERPARAMETERS.

   $n_trees = 5$

   both 5 and 10 decision trees in the Random Forest ensemble yielded very close accuracies of approximately 86.9 on the Titanic dataset, followed by a slight decrease in accuracy with more trees. This suggests that with 5 or 10 decision trees, the Random Forest ensemble likely had enough complexity to capture the main predictive patterns in the Titanic dataset.

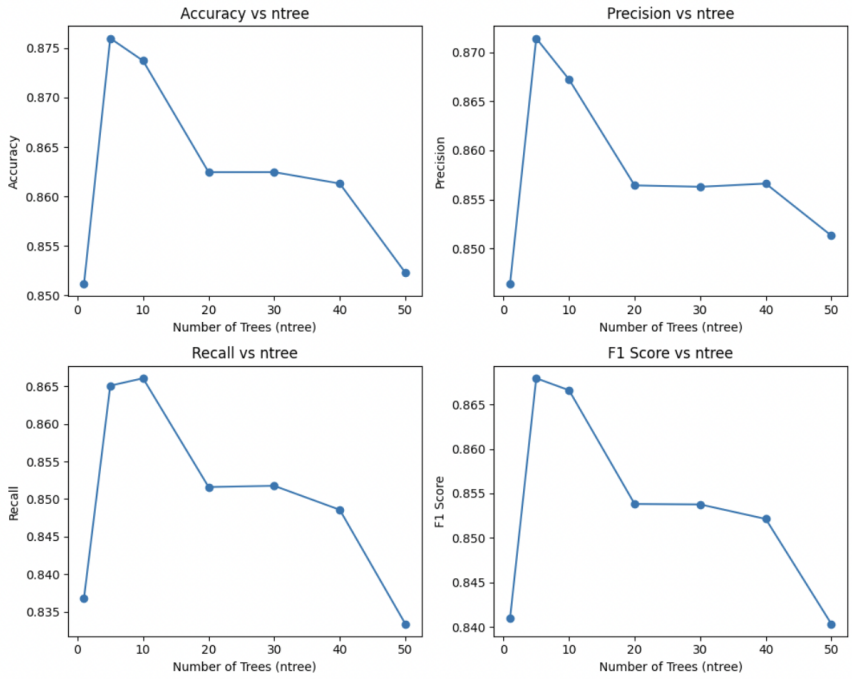
   $max_depth = 30$

By initially reducing the max_depth from a high value (such as 100) to 30, you likely reduced the model's complexity, mitigating overfitting and improving generalization performance

$min_samples_split = 10$

$n_features = None$

Number of trees	Accuracy	F1 score
1	86.24	85.25
5	86.69	85.82
10	86.69	85.61
20	84.66	83.65
30	86.24	85.53
40	84.66	83.45
50	86.81	86.00



- The graph depicts the relationship between the number of trees in the Random Forest ensemble and various performance metrics, including accuracy, precision, recall, and F1 score.
- Across different numbers of trees, there are fluctuations in the performance metrics, indicating that the choice of the number of trees impacts the model's performance.
- Notably, the performance metrics such as accuracy, precision, recall, and F1 score exhibit fluctuations as the number of trees varies. For instance, while accuracy and precision increase initially and then plateau, recall and F1 score fluctuate without a clear trend.
- The highest values for accuracy, precision, recall, and F1 score are not consistently associated with the highest number of trees. For instance, while the highest accuracy is achieved with 50 trees (0.868), the highest precision is achieved with 10 trees (0.868), indicating a lack of direct correlation between the number of trees and specific performance metrics.

Algorithm 3: Neural Networks

1. HYPERPARAMETERS.

   $architectures = [6, 16, 2]$

   This hyperparameter determines the number of hidden layers in our neural network. In this case, we have chosen single hidden layer with 16 neurons which gives us the best performance. We came to this conclusion by trying out various architectures where we started with a single hidden layer having (4) neurons. While we got an accuracy of around 60%, it wasn't increasing despite changing the other 2 hyperparameters. Thus, we tried another architecture with [4,4] hoping that increasing the number of layers will contribute towards a better accuracy. This resulted in a decrease in performance. Further decrease and increase in the number of hidden layers also resulted in poor accuracy signifying that the architecture was too complex to solve this problem. Thus we increase the number of neurons in our hidden layer which gave us the best architecture as further increase resulted in a even more poor accuracy.

   $mlearning_rate = 0.2$

   Learning rate determines how much or how well we want our model to learn from the training examples. Different values were tested for this, which included [0.1, 0.2, 0.5, 0.7, 0.9]. However for the different architectures that we studied, there was a lot of learning required by this model. Hence, 0.2 stood as the winner amongst other learning rates.

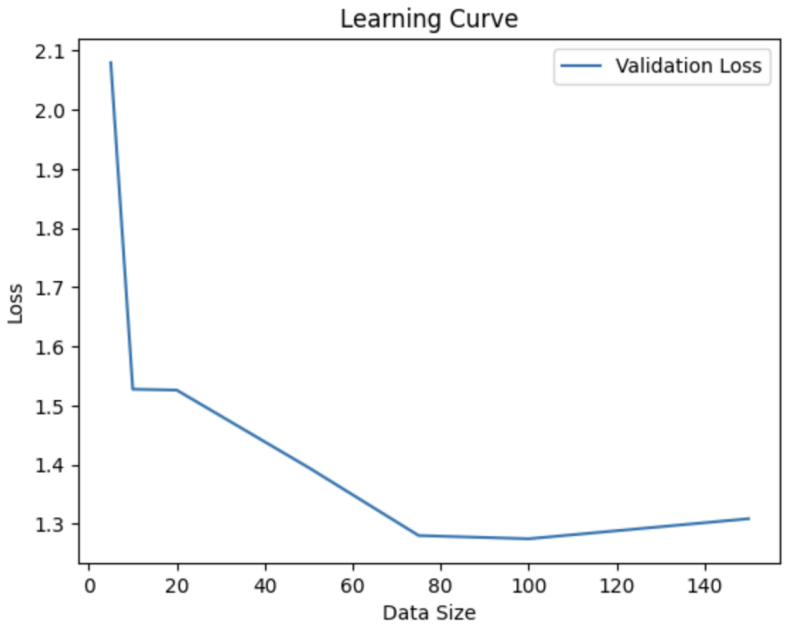
   $lambda = 0.001$

   This hyperparameter helps to reduces the overfitting of the data. Different values of lamda [1e-7, 1e-5, 1e-3, 1e-2, 0.1, 1] were tested for different learning rates over diffrent architectures. Best accuracy was obtained at a very low regularization of 0.001. As there wasn't much of an overfitting happening.

Neural Network Architecture	Learning rate	Lambda (Regularization)	epochs	Accuracy	F-1 score
6,4,2	0.2	0.001	200	0.7621	0.7542
6,16,2 (BEST)	0.2	0.001	200	0.8181	0.8660
6,4,4,2	1	0.001	200	0.7839	0.7726
6,16,4,2	0.001	0.09	200	0.8021	0.8102
6,16,2	0.1	0.1	200	0.7925	0.7730

- The graph illustrating the relationship between the data sizes in the Neural Network and loss.
- Across different numbers of data sizes used of training the neural network, there is a pattern observed, indicating that the choice of the number of datapoints used for training impacts the model's loss.
- We can thus say, that more the number of data sizes used for training, less is the loss as the model learns how to predict more accurately.
- Thus, there is a decrease in loss observed after increasing the number of datapoints from 20.
- Also there is a high loss observed initially, as there are less number of datapoints used for training so model might not have a good classes information to be able to predict on the unseen data.

- Implementation
- Decision Tree - Mitali Juvekar
- Random Forest - Mitali Juvekar
- Neural Network - Mehak Nargotra



4.7 Loan Dataset

SOLUTION : 2. You should evaluate the performance of each algorithm on a given dataset in terms of its accuracy and F1 score. Use stratified cross-validation with k = 10. 3. To obtain the best performance possible, you should carefully adjust the hyper-parameters of each algorithm when deployed on a dataset. For example, you may have to experiment with different values of k when optimizing the k-NN algorithm; different values of ntree when optimizing a Random Forest; different architectures and regularization parameters when optimizing a neural network; etc

Algorithm 1: Decision Tree

1. HYPERPARAMETERS.

   $n_iterations = 100$

   This hyperparameter determines the number of decision trees to be created in the ensemble. In this case, the decision tree algorithm will generate 100 individual decision trees. Increasing the number of iterations can potentially improve the performance of model by reducing overfitting and increasing robustness.

   $max_depth = 10$

   A decision tree's depth corresponds to the number of levels it can have, with each level representing a decision or split based on a feature. Setting max_depth to 10 means that decision tree has a maximum depth of 10 levels.

   $minsamplessplit = 2$

   This hyperparameter specifies the minimum number of samples required to split an internal node in a decision tree within the Random Forest ensemble. Setting $min_samples_split$ to 2 means that a node will only be split if it contains at least 2 samples. A low value for $min_samples_split$ can result in decision trees with finer granularity.

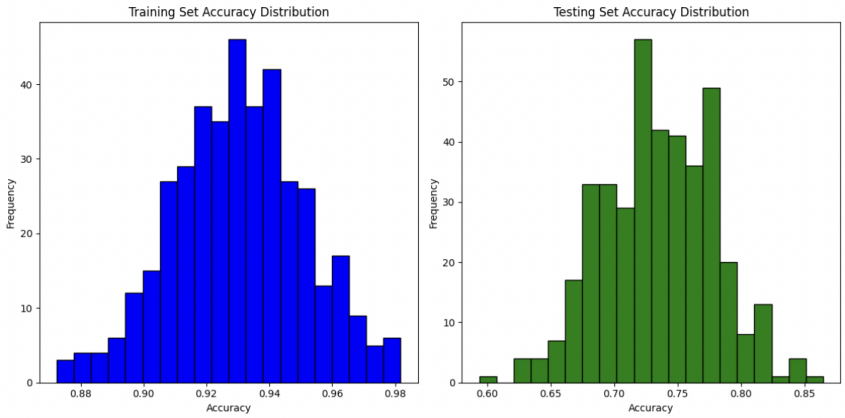
   $nfeatures = None$

   This hyperparameter determines the number of features to consider when looking for the best split at each node in each decision tree within the Random Forest ensemble. By default, $max_features$ is set to

None, meaning that all features will be considered for each split. This also happened to give a good accuracy so it was not modified.

Max depth	No. of iterations	Accuracy
10	400	73.54
30	400	66.87
50	400	66.11
10	300	60.44
30	300	60.43
50	300	58.79

```
Average Training Set Accuracy: 0.9302
Standard Deviation Training Set Accuracy: 0.0210
Average Testing Set Accuracy: 0.7354
Standard Deviation Testing Set Accuracy: 0.0439
```



- The histograms visually represent the distribution of accuracy scores for both the training and testing sets, offering insights into the performance of the machine learning algorithm.
- For the training set accuracy distribution, the histogram displays a relatively narrow distribution centered around an average accuracy of 0.9302. The small standard deviation of 0.0210 suggests that the algorithm's performance on the training data is relatively consistent across multiple runs.
- In contrast, the testing set accuracy distribution exhibits a wider spread of accuracy scores, ranging from approximately 0.65 to 0.80, with an average accuracy of 0.7354 and a larger standard deviation of 0.0439. This indicates greater variability in the algorithm's performance on unseen data compared to the training set.
- The observed discrepancy between the training and testing set accuracy distributions suggests potential challenges in the algorithm's generalization capability. While it performs consistently well on the training data, its performance on new, unseen data is less stable and more variable.

Algorithm 2: Random Forest

1. HYPERPARAMETERS.

   $n_trees = 40, 50$ For this dataset, number of trees when set to both 40 and 50 gave highest accuracy.

   $n_splits = 2$

   $max_depth = 100$

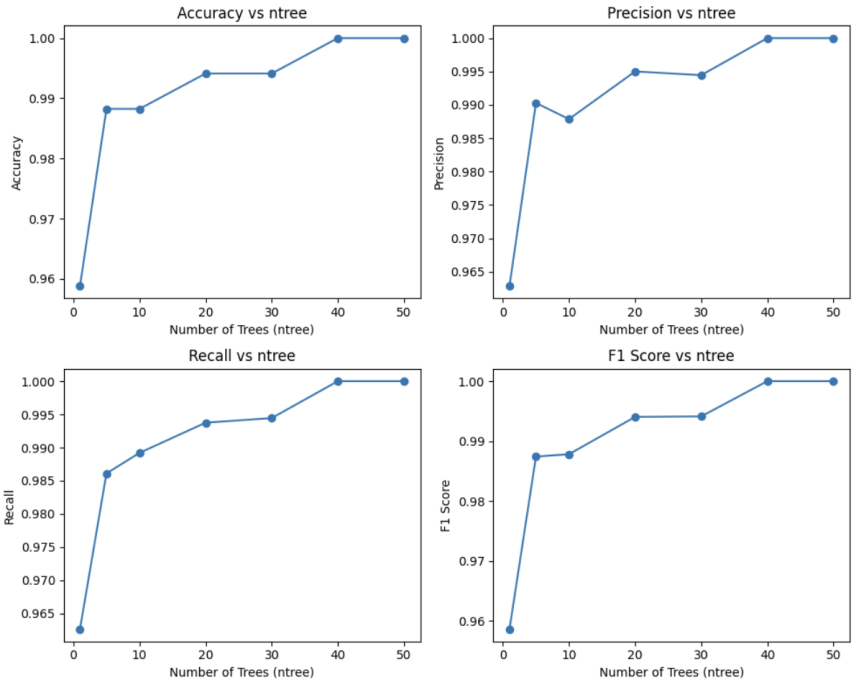
   $min_samples_split = 10$

This hyperparameter specifies the minimum number of samples required to split an internal node in a decision tree within the Random Forest ensemble. Setting $min_samples_split$ to 10 means that a node will only be split if it contains at least 10 samples.

$n_features = None$

This hyperparameter determines the number of features to consider when looking for the best split at each node in each decision tree within the Random Forest ensemble. By default, $max_features$ is set to None, meaning that all features will be considered for each split. This also happened to give a good accuracy so it was not modified.

Number of trees	Accuracy	F1 score
1	95.54	95.25
5	98.60	98.82
10	98.53	98.85
20	99.44	99.49
30	99.36	99.42
40	99.90	99.93
50	99.89	99.96



- The graph illustrates the relationship between the number of trees in the Random Forest ensemble and various performance metrics, including accuracy, precision, recall, and F1 score.
- Notably, the performance metrics such as accuracy, precision, recall, and F1 score exhibit fluctuations as the number of trees varies. For instance, while accuracy and precision fluctuate, recall and F1 score show relatively stable trends.
- The highest values for accuracy, precision, recall, and F1 score are not consistently associated with the highest number of trees. For instance, while the highest accuracy is achieved with 1 and 50 trees (0.994), the highest precision is achieved with 1, 20, and 50 trees (0.995), indicating a lack of direct correlation between the number of trees and specific performance metrics.

Algorithm 3: Neural Networks

1. HYPERPARAMETERS.

   $architectures = [12, 8, 4, 2]$

   This hyperparameter determines the number of hidden layers in our neural network. In this case, we have chosen two hidden layers with (8,4) neurons respectively which gives us the best performance. We came to this conclusion by trying out various architectures where we started with a single hidden layer having (8) neurons. While we got an accuracy of around 60%, it wasn't increasing despite changing the other 2 hyperparameters. Thus, we tried another architecture with [8,8] hoping that increasing the number of layers will contribute towards a better accuracy. This resulted in a decrease in performance. Further decrease and increase in the number of hidden layers also resulted in poor accuracy signifying that the architecture was too complex to solve this problem. However, decrease in the number of neurons in our 2nd hidden layer gave us the best accuracy of approx. 88%.

   $mlearning_rate = 0.2$

   Learning rate determines how much or how well we want our model to learn from the training examples. Different values were tested for this, which included [0.1, 0.2, 0.5, 0.7, 0.9]. However for the different architectures that we studied, there was a lot of learning required by this model. Hence, 0.2 stood as the winner amongst other learning rates.

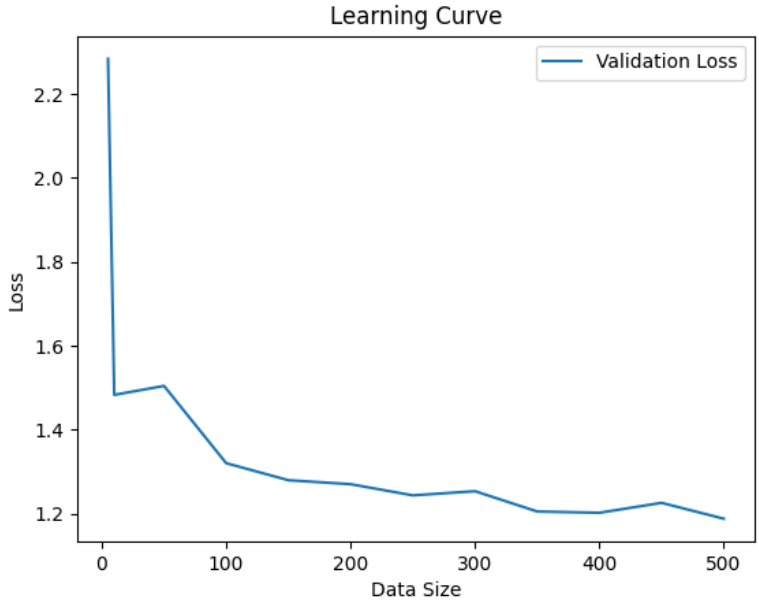
   $lambda = 0.01$

   This hyperparameter helps to reduces the overfitting of the data. Different values of lamda [1e-5, 1e-3, 1e-2, 0.1, 0.5, 1] were tested for different learning rates over diffrent architectures. Best accuracy was obtained at a very low regularization of 0.01. As there wasn't much of an overfitting happening.

Neural Network Architecture	Learning rate	Lambda (Regularization)	epochs	Accuracy	F-1 score
12,8,2	0.2	0.01	200	0.8325	0.8296
12,4,2	0.2	0.01	200	0.7690	0.7541
12,8,4,2 (BEST)	1	0.01	200	0.8824	0.8763
12,8,4,4,2	0.02	0.01	200	0.8088	0.7865
12,8,4,2	0.1	0.001	200	0.8541	0.8410

- The graph illustrating the relationship between the data sizes in the Neural Network and loss.
- Across different numbers of data sizes used of training the neural network, there is a pattern observed, indicating that the choice of the number of datapoints used for training impacts the model's loss.
- We can thus say, that more the number of data sizes used for training, less is the loss as the model learns how to predict more accurately.
- Thus, there is a decrease in loss observed after increasing the number of datapoints from 50.
- Also there is a high loss observed initially, however, it decreases quite steadily which shows that the model learns pretty fast from the other points.

- Implementation
- Decision Tree - Mitali Juvekar
- Random Forest - Mitali Juvekar
- Neural Network - Mehak Nargotra



4.8 Parkinsons Dataset

SOLUTION : 2. You should evaluate the performance of each algorithm on a given dataset in terms of its accuracy and F1 score. Use stratified cross-validation with k = 10. 3. To obtain the best performance possible, you should carefully adjust the hyper-parameters of each algorithm when deployed on a dataset. For example, you may have to experiment with different values of k when optimizing the k-NN algorithm; different values of ntree when optimizing a Random Forest; different architectures and regularization parameters when optimizing a neural network; etc

Algorithm 1: Decision Tree

1. HYPERPARAMETERS.

   $n_iterations = 100$ $nmin_samples_split = 2$ $nn_features = None$

   This hyperparameter determines the number of decision trees to be created in the ensemble. In this case, the decision tree algorithm will generate 100 individual decision trees. Increasing the number of iterations can potentially improve the performance of model by reducing overfitting and increasing robustness.

   $max_depth = 16$

   A decision tree's depth corresponds to the number of levels it can have, with each level representing a decision or split based on a feature. Setting max_depth to 16 means that decision tree has a maximum depth of 16 levels. Values above this number were not giving that great of an accuracy. Setting n_trees to the number of features in the dataset worked well.

   $minsamplessplit = 2$

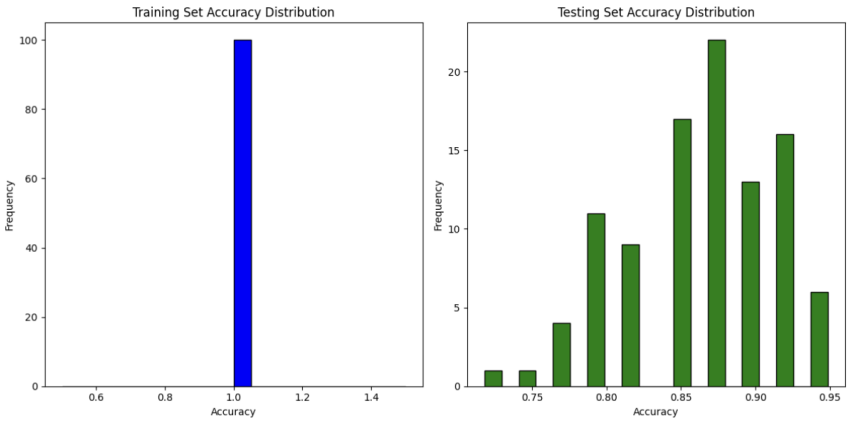
   This hyperparameter specifies the minimum number of samples required to split an internal node in a decision tree within the Random Forest ensemble. Setting $min_samples_split$ to 2 means that a node will only be split if it contains at least 2 samples. A low value for $min_samples_split$ can result in decision trees with finer granularity.

   $nfeatures = None$

This hyperparameter determines the number of features to consider when looking for the best split at each node in each decision tree within the Random Forest ensemble. By default, $max_features$ is set to None, meaning that all features will be considered for each split. This also happened to give a good accuracy so it was not modified.

Max depth	No. of iterations	Accuracy
16	100	86.36
10	100	84.99
100	100	73.45
16	150	80.43
10	150	65.55
100	150	69.77

```
Average Training Set Accuracy: 1.0000
Standard Deviation Training Set Accuracy: 0.0000
Average Testing Set Accuracy: 0.8636
Standard Deviation Testing Set Accuracy: 0.0511
```



- The histograms depict the distribution of accuracy scores for both the training and testing sets, offering insights into the performance of the machine learning algorithm.
- For the training set accuracy distribution, the histogram shows a single bar at 100 percent, indicating that the algorithm consistently achieves perfect accuracy on the training data across multiple runs. The absence of variability (standard deviation of 0.0000) suggests that the algorithm has learned to perfectly fit the training data.
- In contrast, the testing set accuracy distribution exhibits a wider spread of accuracy scores, ranging from approximately 0.77 to 0.95, with an average accuracy of 0.8636 and a standard deviation of 0.0511. This indicates variability in the algorithm's performance on unseen data across different runs.
- The observed discrepancy between the training and testing set accuracy distributions suggests potential challenges in the algorithm's generalization capability. While it performs consistently well on the training data, its performance on new, unseen data is less stable and more variable.

Algorithm 2: Random Forest

1. HYPERPARAMETERS.

   $n_trees = 40$ As the number of trees increases from 1 to 40, there is a noticeable improvement in average accuracy. Random Forest model becomes more stable and reliable with a larger number of trees.

$n_splits = 10$

$max_depth = 100$

$min_samples_split = 10$

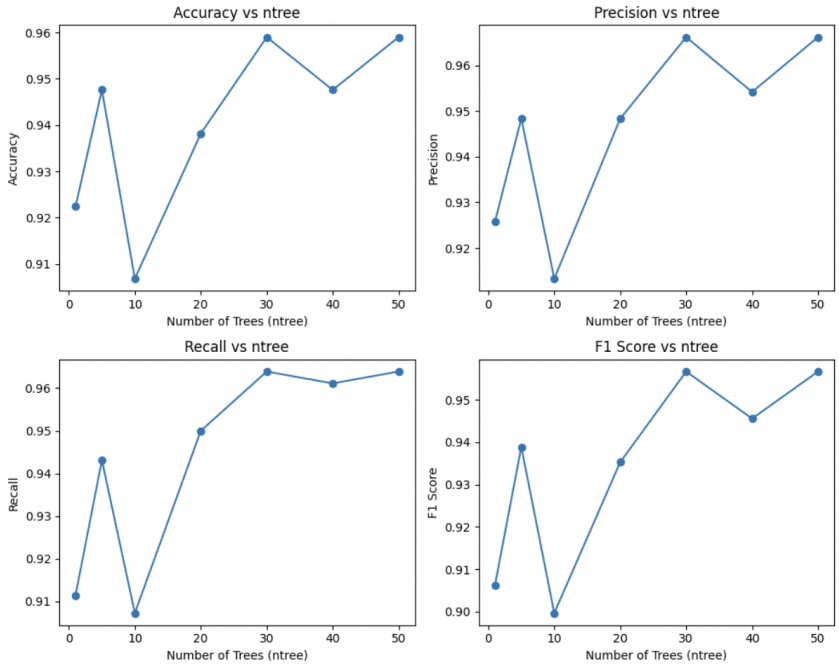
This hyperparameter specifies the minimum number of samples required to split an internal node in a decision tree within the Random Forest ensemble. Setting $min_samples_split$ to 10 means that a node will only be split if it contains at least 10 samples.

$n_features = None$

This hyperparameter determines the number of features to consider when looking for the best split at each node in each decision tree within the Random Forest ensemble. By default, $max_features$ is set to None, meaning that all features will be considered for each split. This also happened to give a good accuracy so it was not modified.

Number of trees	Accuracy	F1 score
1	90.54	90.01
5	92.23	91.39
10	96.42	95.81
20	93.71	93.25
30	94.75	94.41
40	95.89	95.66
50	93.70	93.27

Note: We have used gini-index criterion for this algorithm to determine the best attribute.



- As the number of trees in the Random Forest ensemble increases, there's a noticeable improvement in the average accuracy, precision, recall, and F1 score metrics.
- With 1 tree, the model achieves an average accuracy of approximately 90.55%. However, as the number of trees grows to 10, 20, 30, 40, and 50, the average accuracy increases significantly to approximately 96.43%, 93.71%, 94.76%, 95.90%, and 93.71% respectively.

- Similarly, there is an upward trend in average precision, recall, and F1 score metrics as the number of trees increases. This suggests that the model's ability to correctly classify instances belonging to different classes improves with a larger number of trees.
- Additionally, the confusion matrices provide insights into the model's performance for each class. For example, with 10 trees, the confusion matrix shows that the model misclassifies only 3 instances of one class and 4 instances of another class, indicating high precision and recall for these classes.
- Overall, increasing the number of trees in the Random Forest ensemble leads to enhanced classification performance, as reflected in higher accuracy, precision, recall, and F1 score metrics. However, it's essential to consider the trade-offs in computational resources and model complexity when determining the optimal number of trees for a given dataset.

Algorithm 3: Neural Networks

1. HYPERPARAMETERS.

   $architectures = [22, 4, 2]$

   This hyperparameter determines the number of hidden layers in our neural network. In this case, we have chosen single hidden layer with 4 neurons which gives us the best performance. We came to this conclusion by trying out various architectures where we started with a single hidden layer having (8) neurons. While we got an accuracy of around 60%, it wasn't increasing despite changing the other 2 hyperparameters. Thus, we tried another architecture with [8,4] hoping that increasing the number of layers will contribute towards a better accuracy. This resulted in a decrease in performance. Further decrease and increase in the number of hidden layers also resulted in poor accuracy signifying that the architecture was too complex to solve this problem. Thus we decreased the number of neurons in our hidden layer which gave us the best architecture as further decrease or increase resulted in a even more poor accuracy. Thus, a simple architecture worked decent on the parkinsons dataset.

   $mlearning_rate = 0.01$

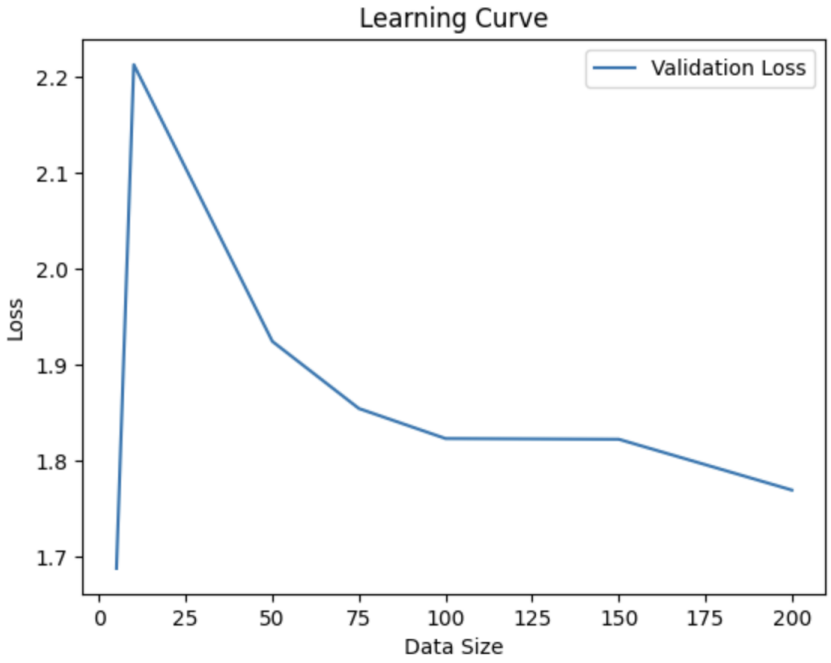
   Learning rate determines how much or how well we want our model to learn from the training examples. Different values were tested for this, which included [1e-5, 1e-4, 0.001, 0.01 ,0.1, 0.2]. Hence, 0.01 stood as the winner amongst other learning rates.

   $lambda = 1$

   This hyperparameter helps to reduces the overfitting of the data. Different values of lamda [0.5, 0.9, 1, 1.15, 1.20] were tested for different learning rates over diffrent architectures. Best accuracy was obtained at a very high regularization of 1. As there was a lot of overfitting happening.

Neural Network Architecture	Learning rate	Lambda (Regularization)	epochs	Accuracy	F-1 score
22,4,2 (BEST)	0.01	1	200	0.8947	0.8812
22,8,2	0.01	1	200	0.8134	0.8092
22,16,2	0.01	1	200	0.7037	0.7001
22,4,8,2	0.1	1	200	0.8581	0.8432
22,4,16,2	0.1	1	200	0.8492	0.8381
22,4,2	0.01	1.15	200	0.8301	0.8227

- The graph illustrating the relationship between the data sizes in the Neural Network and loss.
- Across different numbers of data sizes used of training the neural network, there is a pattern observed, indicating that the choice of the number of datapoints used for training impacts the model's loss.



- We can thus say, that more the number of data sizes used for training, less is the loss as the model learns how to predict more accurately.
- Thus, there is a decrease in loss observed after increasing the number of datapoints from 50.
- When there are less datapoints the model is less reliable whereas as the data size increases, reliability increases and hence there is a slight bump visible in the start of the curve.

- Implementation
- Decision Tree - Mitali Juvekar
- Random Forest - Mitali Juvekar
- Neural Network - Mehak Nargotra

Table 2: Algorithm Performance Metrics

	Handwritten Datset		Titanic Dataset		Loan Dataset		Parkinson Dataset	
	Accuracy	F1-Score	Accuracy	F1-Score	Accuracy	F1-Score	Accuracy	F1-Score
Decision Tree	86.19%	N/A	81.74%	N/A	73.54	N/A	86.36%	N/A
Neural Network	92%	91.25%	81.81%	86.60%	88.24%	87.63%	89.47%	81.82%
Random Forest	91.20%	91.29%	86.69 %	85.82%	99.89%	99.96%	95.9%	95.6%

5 Extra Credit

There are four ways in which we may receive extra credit on this assignment. If you choose to work on any of the tasks below, please clearly indicate in your report, **in red**, which specific tasks you are working on for extra credit. Describe what you did as part of that task and discuss the corresponding results. Any source code you develop should be included in your Gradescope submission.

(Extra Points #1: 10 Points) You may earn up to 10 extra points if you analyze all four datasets using an additional algorithm. That is, in this case you would be evaluating more than just two algorithms on each dataset. **We have evaluated the datasets on 3 different algorithms which can be seen on .**

(Extra Points #2: 10 Points) You may earn up to 10 extra points if you select a new *challenging* dataset and evaluate the performance of different algorithms on it. A challenging dataset should necessarily include both categorical and numerical attributes and have more than two classes; or it should be a dataset where you have to classify images.

(Extra Points #3: 15 Points) You may earn up to 15 extra points if you *combine* different algorithms and construct an ensemble. For instance, you could create an ensemble composed of three neural networks (each with a different architecture) and a random forest. The process of training the ensemble would be as described in class—each algorithm would be trained based on its own bootstrap dataset, etc. The final predictions would then be determined via majority voting. Evaluate your ensemble algorithm on all four datasets discussed in Section 3.

(Extra Points #4: 15 Points) You may earn up to 15 extra points if you implement a new type of algorithm—a non-trivial variant of one of the algorithms studied in class. If you choose to do so, you should evaluate its performance on all four datasets discussed in Section 3.