Certainly! Let's expand each section of the outline into a full 5-page document:

1. Data Preparation
The MNIST dataset is a widely used benchmark dataset in the field of machine learning. It consists of 60,000 training images and 10,000 testing images of handwritten digits, each of size 28x28 pixels. The dataset is split into training and testing sets, with corresponding labels indicating the digit represented by each image.

Normalizing Pixel Values: The pixel values of the images are normalized to a range between 0 and 1. This normalization step helps in stabilizing the training process and improving convergence.

Filtering the Data
For this assignment, we filter the dataset to include only digits 0, 1, and 2. This binary classification problem simplifies the task and allows us to focus on distinguishing between these three classes.

2. Feature Selection
Feature selection plays a crucial role in machine learning, especially when dealing with high-dimensional datasets such as images. In this section, we discuss the process of feature selection and dimensionality reduction using Principal Component Analysis (PCA).

Principal Component Analysis (PCA)
PCA is a popular technique for dimensionality reduction, particularly for datasets with a large number of features. It works by transforming the original feature space into a lower-dimensional space while preserving the maximum variance in the data. This transformation results in a set of orthogonal vectors called principal components, which capture the most important information in the data.

Rationale for PCA
In our approach, we employ PCA to reduce the dimensionality of the feature space while retaining as much information as possible. By selecting a subset of principal components, we aim to reduce the computational complexity of the model while preserving the discriminative power necessary for accurate classification.

Selection of Principal Components
After applying PCA to the filtered dataset, we select a subset of principal components based on their corresponding eigenvalues. Typically, we choose the top-k principal components that capture the majority of the variance in the data. In our case, we select the top 10 principal components to represent the handwritten digits effectively.

Impact of Feature Scaling
Before applying PCA, it is essential to scale or normalize the features to ensure that each feature contributes equally to the analysis. We normalize the pixel values of the images to have a mean of 0 and a standard deviation of 1. This normalization step helps in stabilizing the PCA algorithm and improving the quality of the extracted features.

3. Model Training

With the preprocessed and feature-selected data, we proceed to train machine learning models for digit classification. In this section, we discuss the models used for training, the training process, and the techniques employed for model evaluation.

Models Used

For this assignment, we employ several machine learning models commonly used for classification tasks:

Decision Trees: Decision trees are simple yet powerful models that recursively partition the feature space based on feature values, resulting in a tree-like structure. Each internal node represents a decision based on a feature, while each leaf node represents a class label.

Step 1: Begin with the Root Node

The root node represents the entire dataset initially.

At the root node, we have all the samples from the training dataset.

Step 2: Find the Best Attribute using Gini Impurity

Determine the best attribute to split the dataset. This is done using an Attribute Selection Measure (ASM), often based on metrics like Information Gain, Gini Impurity, or Gain Ratio.

For each attribute, calculate the ASM to evaluate how well it splits the dataset into homogeneous classes.

Choose the attribute that maximizes the ASM as the best attribute for the current node.

Step 3: Divide the Dataset into Subsets

Once the best attribute is determined, split the dataset into subsets based on the possible values of the chosen attribute.

Each subset represents a different branch from the current node, corresponding to one of the possible attribute values.

Step 4: Generate Decision Tree Node

Create a decision tree node corresponding to the best attribute found in Step 2.

Associate each branch of the node with a subset created in Step 3.

Step 5: Recursively Build Sub-Trees

Recursively apply the above steps to each subset created in Step 3.

For each subset, repeat Steps 1 to 4 until one of the following stopping criteria is met:

All samples in the subset belong to the same class, making it a leaf node.

No more attributes are left to split on, or a predefined maximum tree depth is reached.

The subset size falls below a certain threshold, indicating sufficient purity or lack of information gain.

Step 6: Stopping Criteria and Leaf Node Creation

A leaf node is created when one of the stopping criteria is met, indicating that further splitting is unnecessary or not beneficial.

At a leaf node, a decision is made about the class label based on the majority class of samples in that node, or other criteria based on the application context.

Step 8: Tree Evaluation and Validation
Once the tree is constructed, it needs to be validated to assess its performance on unseen data.

Random Forests/Bagging: Random forests are ensemble learning methods that combine multiple decision trees to improve generalization performance. Each tree in the forest is trained on a random subset of the training data, and the final prediction is made by averaging the predictions of all trees.

Training Process
The training process involves fitting the selected models to the training data and optimizing their parameters to achieve the best performance. We use techniques such as grid search and cross-validation to tune the hyperparameters of the models effectively.

Step 1: Bootstrap Sampling
Generate multiple bootstrap samples by randomly sampling from the original dataset with replacement.
Each bootstrap sample has the same size as the original dataset but may contain duplicate instances and miss some original instances.

Step 2: Build Base Models
Train a base model (e.g., decision tree, SVM, etc.) on each bootstrap sample independently.
Each base model is trained on a different subset of the original dataset, resulting in a diverse set of models.

Step 3: Aggregate Predictions
For each instance in the test dataset, collect predictions from all base models.
Aggregate the predictions using a suitable method such as averaging for regression or voting for classification.

Step 4: Final Prediction
Make the final prediction based on the aggregated predictions.
For regression tasks, the final prediction is the average of predictions from all base models.
For classification tasks, the final prediction is the majority class voted by the base models.

Step 5: Evaluate Performance
Evaluate the performance of the bagging ensemble model using appropriate metrics such as accuracy

4. Model Evaluation

```
[ ]  cntr=0
     for i in range(len(pred)):
         if pred[i]==y_t[i]:
             cntr+=1
     print(cntr/len(pred))
```

```
0.8303145853193518
```

```
 966 1067 580 980 1135 1032 1377 1120 650
 class 0 accuracy 98.57142857142858 %
 class 1 accuracy 94.00881057268722 %
 class 2 accuracy 56.201550387596896 %
```

Without Bagging Output

```
x=0
for i in range(len(y_pred)):
    if y_pred[i]==predicted_classes[i]:
        x+=1

print(x/len(y_t))
```

```
0.8706704798220527
```

With Bagging Output