Project 1

Abstract

The project creates 5 machine learning models for digit recognition. We compare all 5 models to determine the complexity of the training. The KNN model compares a validation image to training images and assigns the label of the most similar ones based on pixel distance, the accuracy of my result was between 96-97% for k = 1,3,5. Naive Bayes model assumes each pixel value is independent and estimates probabilities for each pixel to each digit, the accuracy of my result was 83%. Linear Classification was trying to find a single straight hyperplane for each 0-9 data, the accuracy of my result was 84%. Multilayer Perceptron is a connected network with hidden layers and a nonlinear activation function which is ReLU, the accuracy of my result was 90%. CNN uses convolutional layers to process images, the accuracy of my result was 95%.

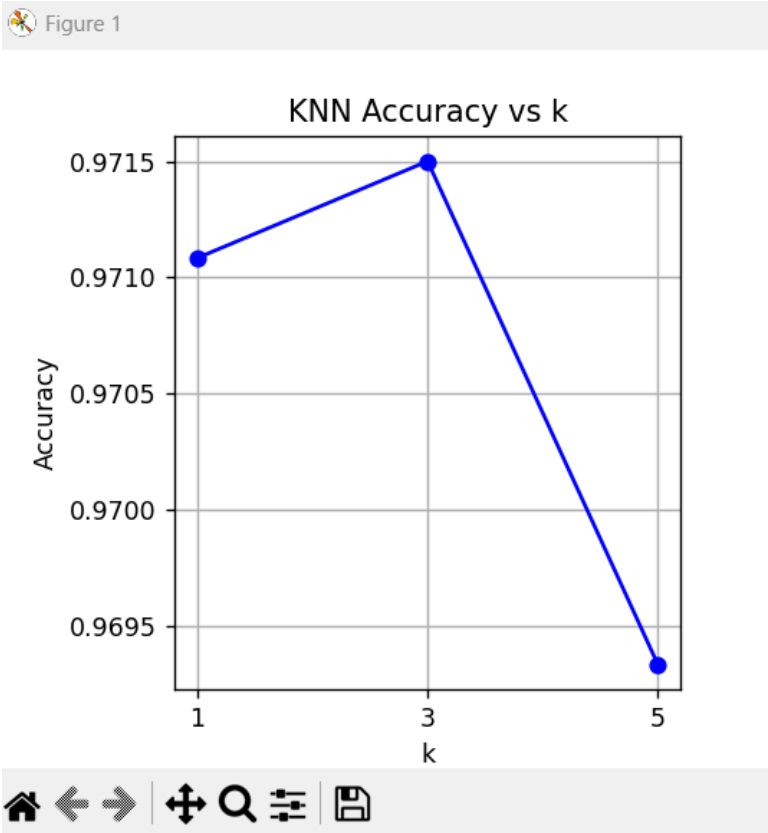
Project Description

The purpose of this project is to create various machine learning models for digit recognition using the MNIST dataset provided. We use 5 different models: K-Nearest Neighbor (KNN), Naïve Bayes, a basic Linear Classifier, a Multilayer Perceptron (MLP), and a Convolutional Neural Network (CNN). Each of these models are different and we are comparing their strength, weakness, and complexity for image classification. The MNIST dataset provided contains ~60,000 images and using this data I had split this into a training dataset and validation dataset. The training dataset contains 80% of the images while the validation dataset contains 20% of the images. A potential application of my project is more complex image recognition.

AI Techniques

K-Nearest Neighbor (KNN)

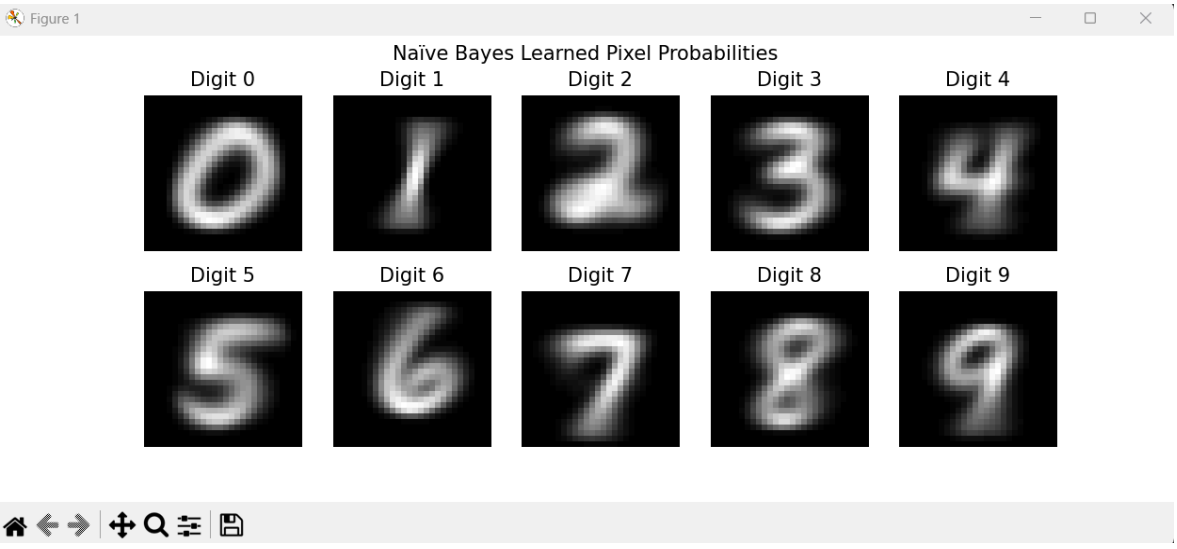
Is a learning algorithm that classifies new data based on the majority class of its closest k neighbor using Euclidean distance. It looks at the closest training examples to decide the label of a new example. Using the KNN mode and when k = 1,3,5 the accuracy I have gotten is:
KNN Accuracy (k=1): 0.9711, Time: 64.5s
KNN Accuracy (k=3): 0.9715, Time: 58.7s
KNN Accuracy (k=5): 0.9693, Time: 55.1s



The following graph shows the accuracy based off of k. Shows how performance will change based on the number of neighbors (k). Based on the code we first train 80% of the dataset by calculating the Euclidean distance between the test image and training image vector. The logic behind this is sorting the distances to find the k nearest neighbor then classifying the image by the majority class amount the neighbor. The algorithm was using NumPy only.

Naive Bayes

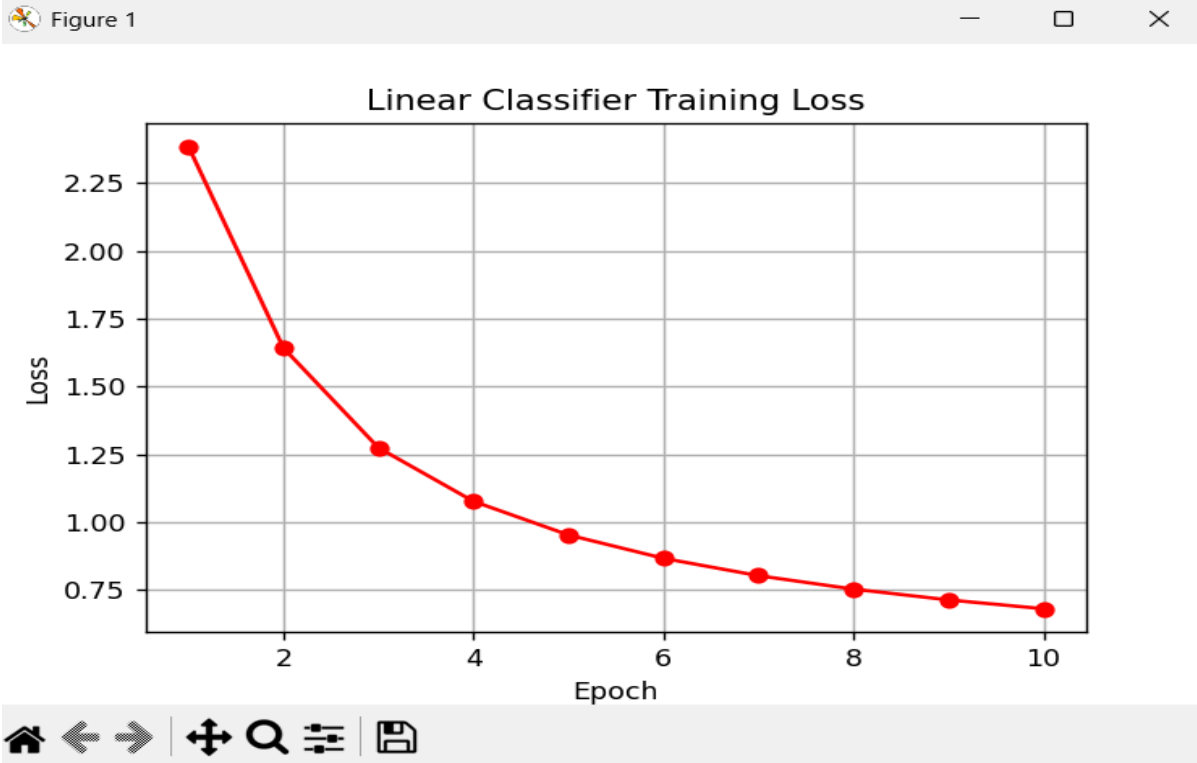
A learning algorithm based on Bayes Theorem that assumes independence between pixels given the label. It calculates the probability of each class given the input features and assigns the class with the highest probability. Using the Naïve Bayes model the accuracy I got is: 0.8323.



The following graph shows a pixel probability heatmaps of naive bayes model. This visually shows what the model learns for each digit. The code binarizes the pixel value by converting them to either 0 or 1. Then I trained and stored conditional probabilities based on each of the 784 binarized pixels being on the value of 1 given each digit from 0-9. The code will apply bayes rule by using the estimated probabilities to calculate the posterior probability of each digit.

Linear Classification

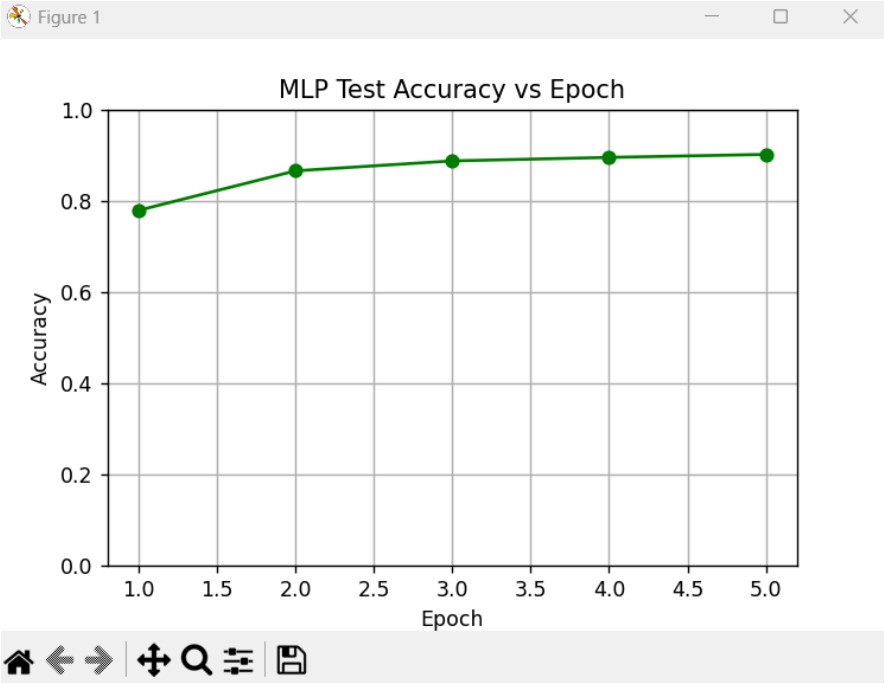
A model that consists of a single linear layer, it is trained using L2 loss and Gradient Descent. It assigns weights to each pixel in the image and combines them to decide which digit the image represents. To learn the weight matrix that linearly separates the classes. Using the Linear Classifier model, the accuracy I got is: 0.8423.



The following graph shows the loss vs epoch. Demonstrate the learning convergence of the linear model. A neural network aiming to find a linear boundary between classes. The operation is a matrix multiplication, y = Wx, where W is the weight matrix and x is the flattened image vector. We use epoch to allow the machine to implement a learning loop while calculating the forward pass, calculation of L2 loss, calculation of the backward pass, and update step using gradient descent. I also used a built-in nn.Linear module and used the SGD optimizer and let the Pytorch handle the gradient calculation.

Multilayer Perceptron (MLP)

A neural network featuring one or more hidden layers with ReLU which is a non-linear activation function. It processes input data through several layers, applying weights and activation functions to learn complex patterns. Using the MLP model, the accuracy I got is:
Epoch 1, Test Accuracy: 0.7562
Epoch 2, Test Accuracy: 0.8618
Epoch 3, Test Accuracy: 0.8850
Epoch 4, Test Accuracy: 0.8963
Epoch 5, Test Accuracy: 0.9032

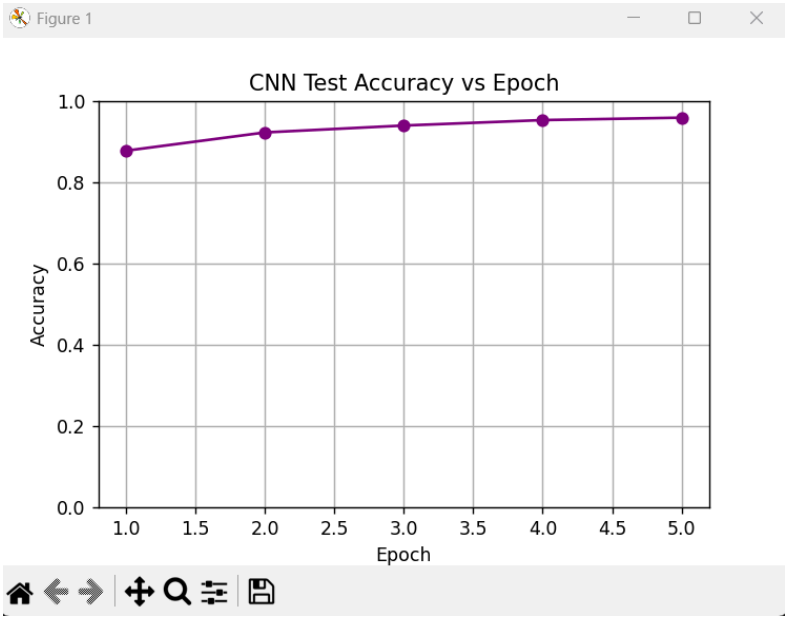


The following graph shows the test accuracy vs the epoch. This shows the improvement over training and generalization performance. A deep learning network using one or more non-linear hidden layers which allow it to learn complex non-linear patterns. First define a sequence of layers using PyTorch's nn.Linear and nn.ReLU while the input layer will take the flattened vector. Then I will train the model using SGD along with a cross entropy loss to train the network on the MNIST data.

Convolutional Neural Network (CNN)

A model that uses convolutional layers and pooling layers to automatically learn features and patterns like edges and curves from the image. It uses filters to scan over the image and capture important patterns like edges, shapes, and textures that help in recognizing digits. This is trained using cross-entrophy loss. Using the CNN model, the accuracy I got is:
Epoch 1, Test Accuracy: 0.8831
Epoch 2, Test Accuracy: 0.9057
Epoch 3, Test Accuracy: 0.9354
Epoch 4, Test Accuracy: 0.9507
Epoch 5, Test Accuracy: 0.9568

This graph shows accuracy vs epoch as well. This highlights CNN's strong learning performance and generalization. CNN is a deep learning network specifically designed for image data. It utilizes convolutional layers to automatically learn features. The code will use an nn.Conv2d and nn.MaxPool2D layers as well as a connect FC layer to produce the digit class output. The imputed image must also be 2D shaped (28 x 28) for the convolutional layers. This model is trained using cross entropy loss to learn the filter that best recognize the digit features



Datasets

The project provided a dataset called MNIST database, which is a standard dataset for training and testing machine learning modules in pattern recognition. The dataset consists of ~60,000 images of digits from 0 to 9. Each image is a 28 x 28 pixel. For each machine learning model there was a random split of the dataset to train the model and to validate using the trained model. The split was 80% for training and 20% for validating. All image pixel values were normalized and flattened into a 784 dimensional vector.

Implementation Tools

Category	**Tools/Package Used**	**Purpose**
Platform	Python 3.13.5	The core language for implementation
Math	NumPy	Used for all mathematical operation and arrays
Deep Learning	PyTorch	The deep learning framework used for this project.
Data Handling	NumPy, PyTorch	Loading and preprocessing the MNIST data.

Visualization	Matplotlib	Used to generate graphs of data to visualize the model components.

References and AI Assistance

The AI assistance I just to help me through the project is ChatGPT and Gemini. I have used these two AI assistants to help me understand what each model is doing and how I should implement it. I also use them to help debug my code. Having these two assistance was a great resource to review and understand the logic of this project.