Final Report

Abstraction

The goal of our project, led by Team MENJ (Mahmud Hasan, John Royal, Nur Mazumder, Erlin Paredes), was to create a system capable of recognizing text. We aimed to achieve this by using machine learning models to identify handwritten digits. By using multiple models - K-Nearest Neighbors (KNN), Neutral Networks, Convolutional Neural Networks (CNN), Naive Bayes, and Logistic Regression - using the MNIST dataset. Our methodology involved applying each model to the dataset comparing their performance based on accuracy. The fundamental goal of this project was to understand how different machine learning models classify handwritten images to grasp a deeper understanding of these complex models.

Introduction

Our project utilized the MNIST dataset, which contains 60,000 training images and 10,000 testing images of handwritten digits. These images are converted into a black and white, 28x28 pixel format, providing a standardized dataset for our models. The models being used include K-Nearest Neighbors (KNN), Neutral Networks, Convolutional Neural Networks (CNN), Naive Bayes, and Logistic Regression. The primary objective of this project, is to compare the accuracy of these models in classifying handwritten digits. For some models we will create our simple model from scratch and compare against prebuilt implementation. Further, we will analyze why certain models have performed better than others. This comparison helps us understand the strengths and weaknesses of each algorithm in handling image data but also provides valuable insights for both educational and practical applications in the field of AI

Pre-processing

The pre-processing stage involved preparing the MNIST dataset for our models. This included converting the images into a format that our models could interpret and use for training. In the case of the neural network, the image's pixel data was converted from its black and white 0 to 255 values into a 28x28 matrix and then a 784x1 matrix to be compatible with the model.

For CNN, logistic regression, and Naive bayes we loaded our dataset from tensorflow and sklearn, the MNIST dataset is somewhat already pre-processed but we still need to further pre-process the images to be used into the convolutional Neural Network, include normalization and reshaping. For normalization, we have to normalize the piel values of the images in the training set. The MNIST images are grayscale, with pixel values ranging from 0 to 255. Normalization scales these values to a range of 0 to 1. This scaling is done to aid in the convergence of the model during training, as smaller, standardized values are generally easier for the model to process.The normalization is done across axis=1, which typically means the axis representing rows in a 2D array (in this case, each row in an image). Next, for reshaping, we set the image size to 28, so it is 28x28 pixels. In our CNN model, The reshape(-1, IMG_SIZE, IMG_SIZE, 1) function changes the shape of each image in the dataset. The -1 ensures that the total number of images remains the same, IMG_SIZE (28) represents the height and width of the images, and 1 is the number of color channels (grayscale images have 1 channel).This step is crucial for CNNs, as they require input to be in a specific shape. The additional dimension (1) indicates the number of color channels. CNNs expect data to be in the format of [samples, height, width, channels], which is what this reshaping achieves.

Modeling

We used several models in our project:

- K-means
- Neural Network
- K-nearest-neighbors
- Logistic regression

In our approach, we implemented a logistic regression model from scratch, specifically designed to handle multi-class classification using the One-vs-Rest (OvR) strategy, a common technique for extending binary classifiers to multi-class problems. Logistic regression fundamentally operates on the principle of modeling the probability of binary outcomes. This is achieved using the sigmoid function,$\sigma(z) = \frac{1}{1+e^{-z}}$, where z represents the linear combination of input features and model weights $z = X\theta$. The sigmoid function transforms these linear outputs into probabilities, facilitating binary classification.

The model employs binary cross-entropy as the loss function, which quantifies the difference between the predicted probabilities and the actual labels. The loss for each observation is calculated as (-[y log(h) + (1 - y) log(1 - h)]), where (h) is the predicted probability and (y) is the true label. The model is optimized using gradient descent, where the weights are iteratively adjusted in the direction that minimizes the average loss across all training samples.

- Naive Bayes

Mathematically, the model is based on Bayes' theorem, which relates the class prior probability, (P(y = c)), the likelihood of the data given the class, (P(X = x | y = c)), and the posterior probability, (P(y = c | X = x)). In this context, the likelihood is modeled using a Gaussian distribution for each feature, defined by its mean μ and variance σ^2

The probability density function (PDF) of the Gaussian distribution is given by ($P(x_i \mid y = c) = \frac{1}{\sqrt{2\pi\sigma^2_{c,i}}} \exp\left(-\frac{(x_i-\mu_{c,i})^2}{2\sigma^2_{c,i}}\right)$. During training (in the \`fit\` method), the classifier calculates and stores the mean and variance for each feature in each class, which characterizes the Gaussian distribution for each feature-class pair.

For predictions, the \`predict\` method computes the posterior probability for each class for a given data point. It uses the log probabilities for numerical stability: (log(P(y = c | X = x)) = log(P(X = x | y = c)) + log(P(y = c))). The classifier then chooses the class with the highest posterior probability as the prediction.

- Convolutional Neural Networks (CNN)

For our CNN, we constructed it using Keras. The Sequential model in Keras is a linear stack of layers. It allows you to create models layer-by-layer in a step-by-step fashion. In a general understanding of CNNs, they utilize convolution layers, where learnable filters slide over the input image to perform element-wise multiplication and extract key features like edges and textures. This convolution process is followed by the application of an activation function, commonly the Rectified Linear Unit (ReLU), which introduces non-linearity, enabling the network to capture complex patterns. The CNN architecture also typically incorporates pooling layers, which serve to reduce the spatial dimensions of the output from the convolution layers, thus decreasing the computational load and the number of parameters. The pooling process, typically either max or average pooling, helps in making the detection of features somewhat invariant to scale and orientation changes. As the network progresses deeper, the convolution and pooling layers hierarchically extract increasingly complex and abstract features. The final stages of a CNN often consist of fully connected layers, where the high-level reasoning based on the extracted features takes place. The output of these layers is then used for tasks like image classification, where the final layer often employs a softmax activation function to generate a probability distribution over different classes. This entire process allows CNNs to effectively learn and make predictions or classifications based on visual input.

Each model was used to train on the MNIST dataset and test its ability to accurately recognize the handwritten digits.

Analysis

Our analysis focused on comparing the performance of each model.

Neural Networks (Mahmud Hasan)

Mahmud Hasan was responsible for the Neural Network model, which was built from scratch using mostly numpy. The neural network model built using PyTorch was created similarly to the one built from scratch. It has around 85% accuracy, but can be improved with addition of more nodes and hidden layers. It does not seem to be overfitting as it has similar accuracy when testing images it had not seen before. An autoencoder network was also built to attempt to recreate images into the MNIST database, which may help train a neural network that can run on less data. Additionally, there is a generator which attempts to use the neural network model in reverse and create new handwritten digits of a similar type and style as the MNIST database, however it is still very inaccurate.

Logistic Regression, Naive Bayes, and CNN (Nur Mazumder)

In our Logistic regression model, the accuracy was 0.87 and the prebuilt model from Scikit-learn was 0.908. In our Naive Bayes, the accuracy was 0.722 and the prebuilt accuracy was 0.55. The reason for such a low accuracy is because Naive Bayes classifiers assume that all features (in this case, pixels of an image) are independent of each other given the class label. This assumption rarely holds true in image data, where adjacent pixels are often highly correlated. The spatial relationship between pixels is crucial in understanding and recognizing images, something Naive Bayes does not account for. Unlike Convolutional Neural Networks (CNNs), Naive Bayes doesn't capture the spatial relationships between pixels. CNNs, through their convolutional layers, can identify patterns like edges, corners, and other complex textures in specific regions of an image, which are essential for effective image classification. For this reason in our CNN model the accuracy is 0.98. We also tested our own handwritten digits and tested against our models. As expected logistic regression and Naive Bayes, performed horribly in predicting the digit, why CNN was able to accurately identify most numbers. However, some limitations I found in the CNN model was that when writing a

6, if we were to enrage the circle curve when handwriting the 6, CNN has trouble identifying whether it is a 6 or 0. While for humans, we can make the interpretation that it is intended to be a 6, the model lacks to understand that and interprets the enlarged curve to be a 0.

KNN (Erlin Paredes)

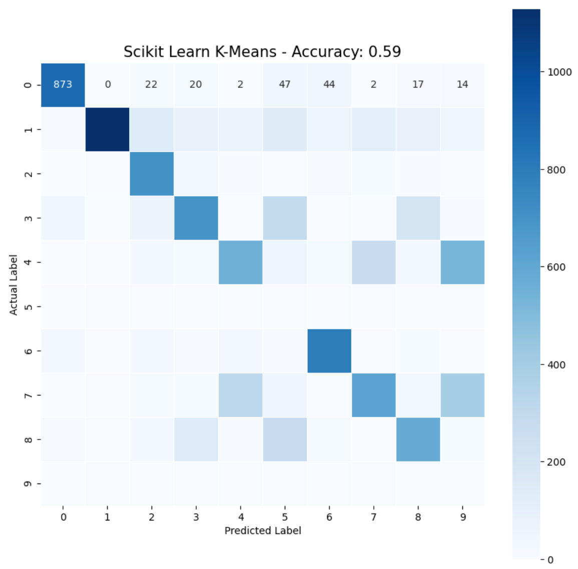
I was able to develop the code that is going to recognize handwritten digits. In order to accomplish this, I used some libraries used in python and the general methodology of the KNN algorithm. The code reads two images, "digits.png" and "test_digits.png"; in grayscale mode, they are both described by the file path. It splits the digits image into 50 rows, and then it further splits each row into 50 cells. The cells are stored in the list cells. It creates labels for each cell. There are 10 labels (0 to 9), and each label is repeated 250 times (assuming 250 cells in each row).It splits the test_digits image into 50 rows and flattens each row.

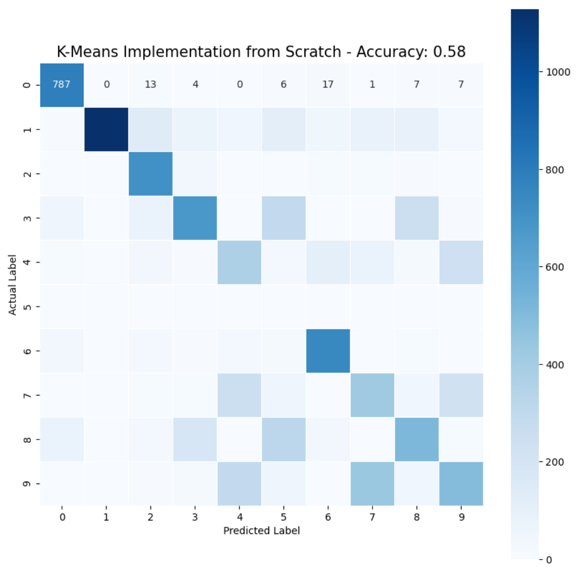
K-Means (John Royal)

For the K-Means project, I began by preprocessing the MNIST dataset, which is crucial since K-Means is sensitive to the scale of the data. The images were normalized to a 0 to 1 range, and to add more diversity, I introduced augmentation by slightly rotating the images. This process helped in simulating a more realistic scenario where data variation is natural.

In implementing K-Means, I developed a class from the ground up to iterate through the core steps of the algorithm: selecting initial centroids randomly, assigning data points to the nearest centroids to form clusters, recalculating centroids as the mean of the points in each cluster, and repeating these steps until the centroids no longer moved significantly. To ensure robustness, I ran this iterative process multiple times, choosing the iteration with the lowest inertia — sum of squared distances within clusters. My analysis involved a detailed look at the confusion matrix, which compares predicted clusters against true labels, to evaluate the accuracy of the clustering. Each cluster's predominant digit was identified, and the accuracy of these assignments was calculated to understand how often the model correctly identified each handwritten digit.

Finally, I compared my implementation to that of the K-Means function in Scikit-Learn. Although the custom implementation took significantly longer (15 minutes for the from-scratch implementation versus 1 minute for the Scikit-Learn one), the accuracy of the two models is comparable.





Project Organization

The final scripts, libraries, and key notebooks used in our project include:

- Libraries: Pandas, NumPy, Matplotlib, Scikit-learn
- Models: K-means, K-nearest-neighbors, Logistic regression, Neural Network
- Key Notebooks:
  - Neural-Network_Mahmud-Hasan/neuralnet.py (contains the code for the Neural Network model)
  - K-Means_John-Royal/main.ipynb (contains the tests and outputs for the K-Means model)

Summary

Our project successfully created a system capable of recognizing text using various machine learning models. While there were challenges along the way, such as the inaccuracy of the generator, we learned valuable lessons about the application of machine learning in text recognition. Furthermore, we were able to get a deeper understanding of different machine learning models such as CNN, logistic regression, Naive Bayes, and why certain models perform better at classifying images than others.

Team Contributions

Quantitative contributions to the project are as follows:

- Mahmud Hasan: Developed the Neural Network model, including the autoencoder network and the generator.
- John Royal: Implemented and tested the K-Means model.
- Nur Mazumder: logistic regression, Naive Bayes, CNN
- Erlin Paredes: Developed the KNN model

https://github.com/QuodFinis/Final-Project/tree/main