REPORT.

Problem statement

The challenge is correctly identifying digits from a dataset of tens of thousands of handwritten images in the MNIST ("Modified National Institute of Standards and Technology") dataset, which has been a reliable resource for benchmarking classification algorithms since its release in 1999. The goal is to gain first-hand knowledge of what works well and how various techniques compare to each other.

Project Objective

Handwritten digit recognition is a well-researched subarea within the field of artificial intelligence that presents significant challenges due to the variation and distortion of handwritten characters. Despite the advances in machine learning methods over the last decade, the variation and distortion of handwritten characters present a significant challenge in this field.

Literature Review

RESEARCH PAPER-

https://www.researchgate.net/publication/326408524_Handwritten_Digit_Recognition_Using_Machine_Learning_Algorithms

SUMMARY-

The Multi-Layer Perceptron (MLP) is a neural network used to classify handwritten digits. It consists of an input layer, hidden layer, and output layer, with nodes connected to the next layer. The number of nodes in the input layer depends on the number of attributes in the dataset, and the number of nodes in the output layer depends on the number of classes. The optimal number of hidden layers and nodes is determined experimentally. MLP uses a supervised learning technique called Backpropagation algorithm to adjust the weights between nodes during training.

Support Vector Machine (SVM) is a supervised machine learning technique used for classification of data points. SVM maps the examples of separate

classes in a high-dimensional space and maximizes the margin between these classes. New examples are mapped into the same space and classified based on which side of the gap they fall on. SVM is developed through a training phase, where training data is used to develop an algorithm capable of discriminating between groups previously defined by the operator. In the testing phase, the algorithm is used to predict the group to which a new perception belongs. SVM provides accurate classification performance over training data and produces enough search space for accurate classification of future data parameters. It is recommended to always scale the data in SVM as it can significantly improve the results. However, caution should be exercised with large datasets as it may lead to increased training time.

J48 is a decision tree algorithm that extends C4.5 and provides tree pruning options to improve accuracy and reduce overfitting. It recursively classifies data until each leaf is pruned, producing more accurate results with excessive rules. J48 applies two pruning methods: subtree replacement and subtree rising, which are useful in balancing accuracy with flexibility. Error rates are needed to make actual conclusions about which parts of the tree to rise or replace. Reduced-error pruning reserves a portion of the training data as test data to reduce overfitting, but it also reduces the volume of data available for training. J48 is useful in developing decision trees for classification tasks and producing generic results. However, it may produce excessive rules, and pruning may cause less accuracy of a model on training data. It is recommended to avoid using reduced error pruning for small datasets.

The random forest algorithm is an ensemble of un-pruned regression or classification trees that use random feature selection and bootstrap samples of the training data. It predicts by accumulating the predictions of the ensemble through majority voting for classification. However, it may suffer from imbalanced training data, focusing more on the majority class and resulting in poor accuracy for the minority class.

Naive Bayes is a simple probabilistic classifier that relies on Bayes theorem with two important simplifying assumptions: that predictive attributes are conditionally independent given the class, and that no hidden attributes

influence the prediction method. Despite its oversimplified assumptions, Naive Bayes performs well in many real-world problems such as email sorting, spam detection, sentiment detection, and document categorization. While it may be outperformed by other approaches, Naive Bayes is computationally less intensive and requires less training data, making it a potent classifier with a considerably smaller training time than alternative approaches such as boosted trees, Max Entropy, Support Vector Machines, and random forests.

Bayesian networks are a graphical model that encodes probabilistic relationships among variables of interest. It can be used for classification by learning the conditional probability of each attribute given the class label from training data. Bayesian networks have several advantages for data analysis, including the ability to handle missing data entries and learn causal relationships. They can be used to gain understanding about a problem domain and predict the consequences of intervention.

Random Tree is an ensemble algorithm that deals with regression and classification problems. It consists of multiple tree predictors called forest, where each tree is trained with the same parameters but on different training sets created through the bootstrap procedure. The algorithm takes an input feature vector and outputs the class label with the most votes for classification or the average of responses over all trees for regression. Random trees do not require accuracy estimation techniques and estimate the error internally during training.

Methodology

Model

To make our model we import Keras which is an open-source software library, a high-level, deep-learning API that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. Then we import our databas MNIST(Modified National Institute of Standards and Technology database). Further we import layers module of keras which provides a set of pre-built layers that can be used to construct a neural network.

Loading the MNIST dataset into our syatem and setting up input shape based on the backend. Since our modelis being developed using TensorFLow as the backend, the input shape needs to be specified in terms of the number of input features and the batch size. We reshape our data to fit the CNN input shape because convolutional neural networks (CNNs) require input data to be in a specific format in order to operate effectively. The next step is to convert pixel values to float float and normalize them between 0 and 1 for two main reasons which are numerical stability and improved performance. We convert labels to one hot encoded vectors in order to represent the categorical or nominal data in a format that can be used as input to machine learning models, particularly for multi-class classification tasks. One hot encoding is a process of converting categorical variables into a numerical representation that can be used as input to machine learning algorithms. The next step is to build the CNN model nd complie it with cross-entropy loss and adam optimizer. We now train our model with 10 epochs(a complete iteration over the entire dataset during the training phase of a machine learning model) and batch size of 128. The last is to evaluate the model on the testing data.

Whiteboard

The code imports the tkinter module and defines a class called Whiteboard. The Whiteboard class has an initialization method called "init" that takes a "master" parameter, which is a reference to the root window of the GUI application.

The initialization method sets up the GUI window, including its title and size, and creates a canvas on which to draw. It also sets up mouse event handlers for drawing lines on the canvas when the user clicks and drags the mouse. Additionally, it creates a button for clearing the canvas and binds it to a method called "clear_canvas".

The "start_draw" method sets the starting coordinates of the line when the user first clicks the mouse on the canvas. The "draw" method draws a line from the last coordinates to the current coordinates of the mouse as the user drags the mouse. The "end_draw" method is currently empty, but could be used to perform some action when the user stops drawing.

The "clear_canvas" method deletes all the objects on the canvas, effectively erasing the drawing.

Finally, the code creates a Tkinter root window, and an instance of the Whiteboard class with the root window as its master. This starts the GUI application and keeps the window open until the user closes it.

Model and tkinter integration

Our code code consists of two classes, "DigitRecognizer" and "Whiteboard". The "DigitRecognizer" class contains a method that uses a trained CNN model to recognize digits in an image. The "Whiteboard" class contains methods for setting up a graphical user interface (GUI) with a canvas on which the user can draw digits using the mouse. When the user finishes drawing a digit, the "Whiteboard" class uses the "DigitRecognizer" class to recognize the digit and print the result.

1. The required libraries, including tkinter, numpy, and PIL, are imported.
2. The "DigitRecognizer" class is defined with an "**init**" method that loads a pre-trained CNN model from a file using the Keras library. A "recognize_digit" method is also defined that takes an image as input, resizes it to 28x28 pixels, converts it to grayscale, inverts the colors (to make the background black and the digit white), and normalizes the pixel values to be between 0 and 1. The method then uses the loaded CNN model to predict which digit is represented by the image and returns the result.
3. The "Whiteboard" class is defined with an "**init**" method that creates a GUI window using the tkinter library. The window has a canvas on which the user can draw using the mouse. The canvas is set up with mouse event bindings to track when the user starts, continues, and finishes drawing. A "clear_canvas" method is also defined that erases all of the contents of the canvas when called. Finally, an instance of the "DigitRecognizer" class is created to be used for recognizing digits.
4. When the user starts drawing on the canvas, the "start_draw" method is called and stores the coordinates of the mouse cursor.
5. As the user continues to draw on the canvas, the "draw" method is called repeatedly and creates lines between the current mouse position and the previous one.
6. When the user finishes drawing on the canvas and releases the mouse button, the "end_draw" method is called. This method uses the PIL

library to create an image of the current contents of the canvas. It then calls the "recognize_digit" method of the "DigitRecognizer" class to determine which digit is represented by the drawn image. Finally, it prints the result to the console.

7. If the user clicks on the "Clear" button, the "clear_canvas" method is called, which erases all of the contents of the canvas.
8. The window is kept open using the tkinter "mainloop" method, which waits for user input until the window is closed.