Optical Digit Recognition

Aritifical Intelligence Project

Namish Arora

Table of Contents

Introduction

The purpose of this final project was to explore the concepts of Artificial Intelligence and come up with a problem which can be solved using the concepts of AI. The project we chose was of

Optical Character Recognition. The purpose of this project is to explore a program that can take a visual representation of a digit, from 0-9, and try to identify which digit it is. It is a specific case of an optical character recognition project. This sort of project has lots of important real life use cases, such as

- automatically reading documents like cheques, passports, invoices, etc.
- parsing the information for data entry
- automatic vehicle plate recognition
- traffic sign recognition for self-driving vehicles
- automatic scanning of credit cards.

There are lots of various scenarios where time can be saved by allowing automatic text/image recognition and classification.
Our project will focus on just identifying one digit from a csv format file representing a digit. Each cell will represent a 'pixel' in the picture of the number. It will process thousands of example images to train it to recognize the probability of a certain digit being present given the pixels in the image. With a large enough training sample, it can reach very high levels of accuracy.

Methods

While character recognition can be done with several concepts

For example: -

Using Naïve Bayes theorem

Using Fuzzy logic

Using Neural Networks

The concept we used for our project is Naïve Bayes Theorem along with feature extraction to recognize digits from 0-9.

The dataset we used is called "mnist_train.csv".

The dataset we used had a total of 60,000 rows. Out of all the rows we used the first 50,000 rows to train the data and used the next 10,000 to validate the data we trained.

What is Naïve Bayes Theorem?

Naïve bayes is a classifier that applies Bayes Theorem with strong assumptions of independence between features. Naïve Bayes classifiers are highly scalable, requiring several parameters linear in the number of variables (features/predictors) in a learning problem.

What is Feature Extraction?

Feature extraction is the process of reducing the number of resources required to describe a large set of data. In AI/ML is starts from an initialized set of data to derived values that are informative and are used to organize the data.

Program

The program works under 2 parts. The first part is training the program with the dataset to get all the required data for recognizing a digit and the second part is to recognize a digit based on the data that we have calculated or extracted from part 1.

Part 1 (Training)

The dataset we used has a total of 785 columns for each row, the leftmost column of every row is the digit that the row makes or identifies by the next 784 columns are either 0 or not 0. The dataset is in form of a .CSV file.

These 784 columns are information about each pixel of a 28 * 28-pixel matrix, 28 * 28 = 784, hence 784 columns are used for each row, the number in each of this pixel tells us if the number made in this 2D array is occupying that pixel or not.

The image for reference of data set:

	A	B	C	D		ADC	ADD	ADE
1	3	0	0	0		0	0	0
2	4	0	0	0		0	0	0
3	9	0	0	0		0	0	0
4	6	0	0	0		0	0	0
5	6	0	0	0		0	0	0
6	5	0	0	0	.....	0	0	0

Leftmost column
States the number

784 columns of a 2D
matrix in 1D form

Our dataset has 60,000 rows of this type, and we are training our program with 50,000 rows when we read the dataset.

The feature we have used in our program is different from what we did in Assignment 2 of this course. In that assignment the features we used were the ratio of black pixels in the top of the matrix, in the left side of the matrix and overall black pixels. But here we approached the problem in a different way and considered each pixel to be a feature for

us. For each digit that we are training, we calculate the probability of that digit occupying a particular pixel in all its occurrence.

For example, if number 5 occurs 10 times in our dataset, we calculate the probability of 5 occupying a particular pixel using the 10 occurrences we have.

Hence,

$$P(5\ occupying\ a\ pixel) = \frac{Number\ of\ times\ 5\ occupies\ that\ pixel}{occurrences\ of\ 5}$$

Another set of values we require is the prior probability of each number from the dataset. When we read the dataset, we start calculating occurrences for both the feature of a digit as well as the digit itself. To calculate the prior probability of a digit we need to find out its occurrences in the dataset and then we can calculate the probability of the digit in a finite dataset

For any number X, the prior probability for our program is as follows.

$$P(X) = \frac{Number\ of\ occurrences\ of\ X}{Total\ rows\ of\ training\ in\ dataset}$$

After calculating the occurrences, we normalize the data. Normalizing the data basically means to convert given numbers into probabilities, using the formulas mentioned above we normalize all the data from our data set and store it into variables.

At this point we have all the data to predict a given digit from the test file.

Part 2 (Prediction)

As we have collected all the data, we needed to predict a digit from the training methods. We now test our program by sending in some digits and their 1D matrix in the same format as the ones we used in our training dataset. We take the data for 1 digit and predict it using Naïve Bayes Theorem.

Naïve Bayes Theorem for a single digit (X) with features from f_1 to f_{784}

$$P(X|f) = P(f_1|X) * P(f_2|X) * \ldots\ldots\ldots P(f_{784}) * P(X)$$

Where P(X) is the prior probability

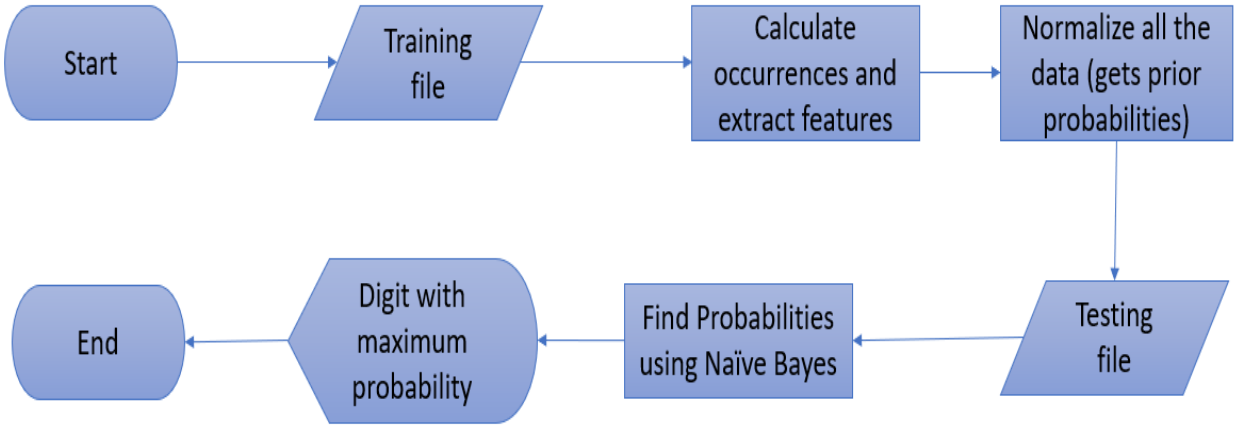
This is how the program computes the probability of every digit.

After calculating the probabilities, the program outputs its prediction which is the highest probability among all the digits.

$$Output = argmax_x(P(X) \prod_{i=1}^{i=784} P(f_i|X))$$

'Output' is the prediction from the program. To verify it we equate it to the left most column of the input file, which outputs if the prediction was right or wrong.

Flow of program



This is a basic flow of program on how our program runs, from start to getting trained and then to be able to predict digits

Important Functions

```
def train():
    # returns a pixel_prob
    prior_prob = defaultdict(int)
    counter = {i: {x: [0, 0] for x in range(10)} for i in range(784)}
    #print(counter)
    for data in train_data:
        data = convert_data(data)
        label = data[0]
        pixels = data[1:]
        prior_prob[label] += 1   ##increasing the occurences of each r
        #print(prior_prob)
        for i in range(len(pixels)):
            counter[i][label][pixels[i]] += 1
```

This function is responsible for handling the data that we have provided for training. It keeps track of all the feature extraction and probabilities.

```
def normalize(probs):
    sum_probs = sum(probs)
    #print(sum_probs)
    if sum_probs == 0:
        return normalize([1 for each in probs])
    eachProb = []
    for each in probs:
        eachProb.append(each/sum_probs)
```

This function is responsible for creating all the number values and data acquired from training to their respective probabilities

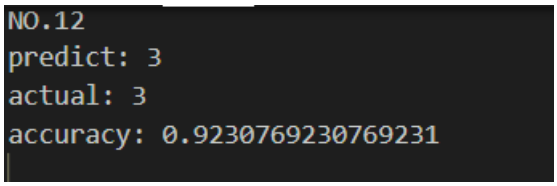
```
def predict(data):

    global pixel_prob, prior_prob
    #print(pixel_prob)
    probs = [prior_prob[x] for x in range(10)]
    data = convert_data(data)
    for i in range(len(data)):
        for x in range(10):
            probs[x] *= pixel_prob[i][x][data[i]]
            #print(probs[x])
        probs = normalize(probs)
    print(probs)
    return argmax(probs)
```

This function takes the input data from the testing file and used to in the Naïve Bayes theorem along with the previous calculated vales to give out the prediction.

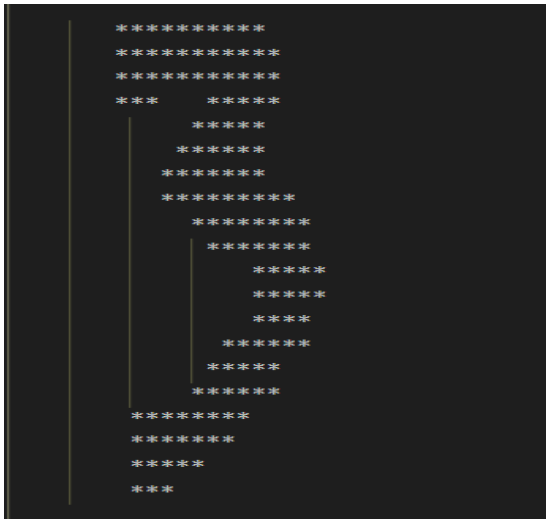
Other Functions

All the other functions in the program are either helper functions or printing functions to make the program easier and to print the outputs in a fancy way.

Results

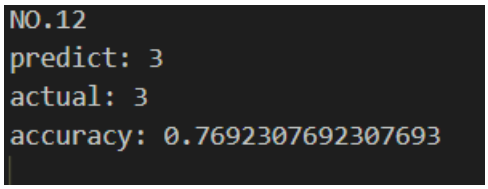


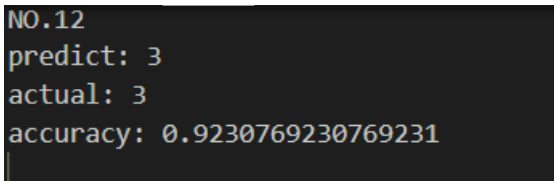
This is how the result of 1 test input looks like You can see what the program predicted and see what the actual number was.



The output also includes an image of the number provided to give an idea to the user about how the testing matrix looks like.

Running the program multiple times with different testing files did not give us different results, and the accuracy remained very close to each other. But changing the number of rows used for training the program somewhat effected the accuracy.





The result on the left image is obtained when the program was trained with just 250 rows of data while the result on the right image is obtained when the program was trained with 50,000 rows of data.

The prediction in these 2 test runs was performed on the same set of testing data but show difference in accuracy due to the difference in training data.

As we got a much higher accuracy while training with 50,000 rows, for all the next iterations of the program we used 50,000 rows of training data and we kept on getting accuracy >90.

(NOTE: - the predict and actual in image is only for the last row of testing data while accuracy if for whole set of testing data)

Discussion

Impacts of project

A reliable method to identify and classify digits from a picture can have a lot of implications when you think of how this technology can be expanded into identifying longer numbers such as credit cards when you're buying something online. Even beyond that you can expand it to identify alphabetical characters, and from there identify words and longer documents. Imagine instead of feeding it pictures, we can allow computers to be able to read using their 'eyes', or in other words, identify words and numbers from a live video.

Limitations

Currently the project is limited to only identifying single digit numbers from the set {0,1,2,3,4,5,6,7,8,9}. It is also limited to reading CSV files as input, with each value in the file representing whether a pixel is on or not. An argument can be made that the accuracy of this method isn't high enough to be relied upon in real world uses yet.

Future work & improvements

Another method that can be explored to achieve the same result with higher accuracy is one that uses neural networks. In the future neural networks can be studied and then implemented, to compare real world differences between the two methods. One specific type of neural network that would work best for this type of project is a convolutional neural network (CNN). A convolutional neural network takes an input image, assigns importance based on learnable weights and biases, and uses that to differentiate the images from one another. In a research study done at the Science and Information Organization comparing the accuracy of convolutional neural networks and naïve bayes classifiers found that the CNN classifier model produces an accuracy of 88%, while the naïve bayes classifier produces an accuracy of 78%. That's a big

enough difference in the real world to make it worthwhile to implement the more accurate method.

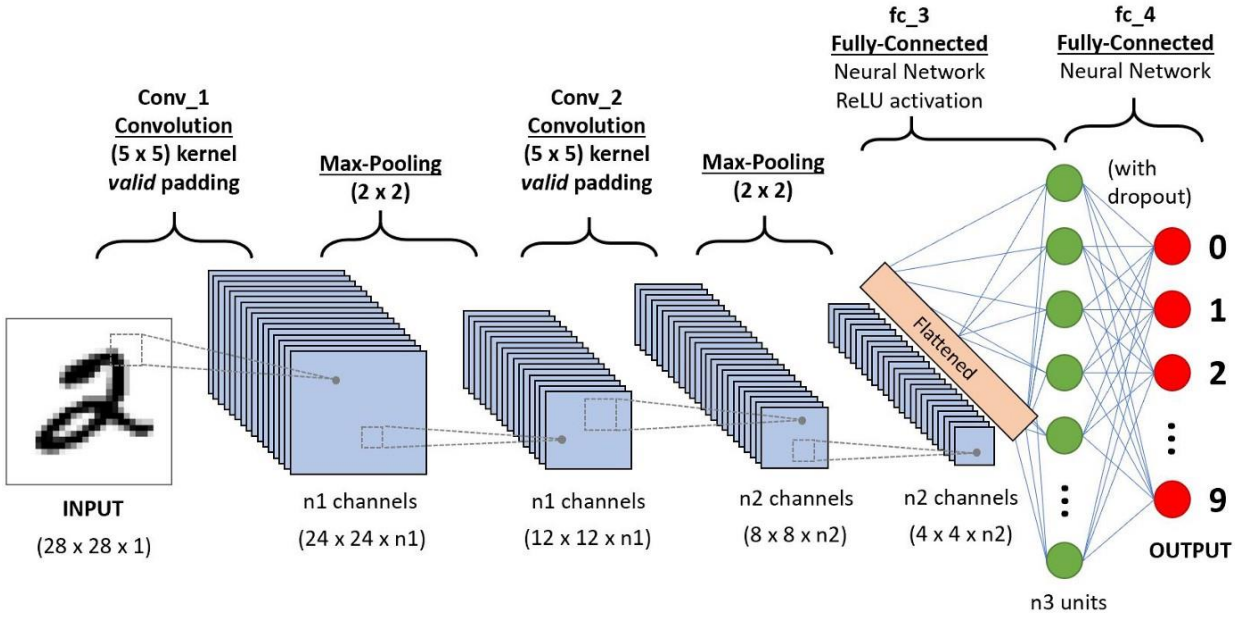


Figure: Convolutional Neural Network sequence to classify handwritten digits

When we can implement this method which is more accurate, we can further extend our program to start reading number with more than 1 digit or read a number till the next space. We would also be able to differentiate between characters and digits.

References

P.O. Abas Sunarya, Rina Refianti, Achmad Benny Mutiara and Wiranti Octaviani, "Comparison of Accuracy between Convolutional Neural Networks and Naïve Bayes Classifiers in Sentiment Analysis on Twitter" International Journal of Advanced Computer Science and Applications (IJACSA), 10(5), 2019. http://dx.doi.org/10.14569/IJACSA.2019.0100511

Naïve Bayes Classifier, Youtube Video ,Krish Naik

https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/