Abstract

Computers excel at performing quantitative and repetitive tasks quickly, but they struggle with qualitative tasks, such as handwriting recognition. This is because it is effectively impossible to program recognition of millions of different handwriting styles into one algorithm. However, with the advent of machine learning, computers can be trained to recognize unknown objects and characters by analyzing a large labelled database. A practical use for machine learning is recognition of handwritten zip codes, which can be performed by a computer and image scanner much faster than a human can, in a setting such as a post office. In this project, the machine learning was performed within MATLAB using support vectors to define boundaries in multidimensional space between handwritten digits. The boundaries were based on a labelled dataset from MNIST (Mixed National Institute of Standards and Technology) containing 60,000 training images. Once the classification boundaries were formed, test images from MNIST were used to check accuracy, and once high enough those learned boundaries were used to determine the number in an image from a webcam with reasonable accuracy in a controlled environment.

1 Introduction

Computers are able to quickly and efficiently perform complex computations; however, computers are not able to distinguish between handwritten characters as easily as humans are. The wide range of variation in the handwriting of humans makes it difficult for computers to distinguish characters and letters from each other with complete accuracy.

Automatic zip code readers have been developed and used by the postal service, but the technology has not been perfected. Handwriting recognition most notably adds efficiency to the postal service and allows for writing on tablet computers. This project explores a way of programming a computer to recognize a zip code written by hand using machine learning algorithms instead of programming for ever type of writing. The computer program will take an image of a handwritten zip code and recognize what zip code it is and which geographic location it correlates to.
2 Background

Within the last decade, many new developments have been made concerning artificial intelligence and classification. Specifically, “Kernel functions” and computational learning theory have been developed allowing to classify and separate data in nonlinear ways. These functions can be implemented using MATLAB, a mathematics software used mainly for matrix and technical computing. In this project, machine learning was used to classify different types of handwritten digits. It uses statistical methods for multiple types of linear classification [1].

2.1 Support Vector Machines

Most often, SVMs are used for nonlinear classification since many problems are too complex to be solved linearly. Often times, SVMs take a long time to run because the program needs time to calculate the coordinates for each specific point. For this reason, many people also use Kernel Functions alongside SVM classifiers to speed up the running time making the program more efficient especially when working with a large amount of data in multidimensional space, a strategy known as the “Kernel Trick” [2]. The 4 types of Kernel Functions [3] are linear, polynomial, radial base function, and sigmoid. The formulas for these categories are shown in Figure 1.

\[
K(X_i, X_j) = \begin{cases} 
  X_i \cdot X_j & \text{Linear} \\
  (\gamma X_i \cdot X_j + C) \quad \text{Polynomial} \\
  \exp(-\gamma |X_i - X_j|^2) \quad \text{RBF} \\
  \tanh(\gamma X_i \cdot X_j + C) \quad \text{Sigmoid} 
\end{cases}
\]

Figure 1: Equations of multiple Kernel Functions

Since digits qualify as separable data, there is no overlap between different classes therefore they can be easily separated by a hyperplane. Because of these factors, a Linear Support Vector Machine was the most appropriate way of classifying the data. Specifically the Binary Classifier, also known as the SVM classification type 1, was used to differentiate between one digit and all other digits. A support vector machine works by differentiating and classifying two different groups by plotting them as vector coordinates in n-dimensional space and finding a hyperplane boundary that could cleanly separate the two groups, a procedure known as optimization. There are many possible boundaries that could divide the vectors, but the SVM finds the one with the maximum margin between it and the nearest vector points. The larger this margin is, the more accurate the SVM will be as shown in the figure below.

Figure 2: Diagram of a Support Vector Machine

When new data needs to be classified, the program converts it into a vector and classifies it based on which side of the boundary it resides on. The program decides which half of the boundary the vectors are on by using the inner product to find an angle between the boundary and
the vector images. The inner product between two vectors is defined as the length of the two vectors multiplied by the cosine of the angle between them. By computing a dot product with each image vector and the vector that defines the boundary, the SVM can figure out the degree of the angle between them[4]. The sign of the angle, positive or negative, defines which side of the boundary each vector is on as shown in the figure below.

\[ \text{Figure 3: Diagram showing how the angle of a boundary is calculated} \]

2.2 One-vs-All Classification

A large component to digit classification is forcing the computer to distinguish between more than two classes. A binary classifier, as discussed in the last section, creates a boundary between two classes. In one-vs-all classification a digit (such as zero) is classified against a group of all the other images of digits. This approach necessitates 10 classifiers, whereby all the digits are distinguished from the other digits, enabled to computer to learn the characteristics of all the digits. See the diagram below for a simplified explanation of one-vs-all classification.

Thus, one-vs-all classification allows the computer to distinguish between all 10 digits, despite using a binary or two class classifier. This is a significantly more efficient approach than classifying each digit against each individual digit. Such a classification strategy, referred to as one-vs-one, would require 45 classifiers and would take nearly an entire day to execute the code.

3 Procedure

3.1 Loading the Training Data

Before the program can differentiate between handwritten digits, it must be taught through examples. Using MATLAB software, 60,000 images of handwritten digits from the Mixed National Institute of Standards and Technology (MNIST) database can be used to train the computer to differentiate between digits. The provided data set contained all the images in a one-dimensional array of roughly 47 million pixels. The file was parsed into groups of 784 after it was inputted into MATLAB to separate all 60,000 of the images. To confirm that the imaging code was correct, the one dimensional arrays of 784 pixels were transformed into a 28 by 28 image array and printed out to the screen.
The result was a series of visible images of the digits. For the images to be inputted into the SVM, the images had to be formatted into rows of 784 pixels. A file that contains corresponding labels for the digits also had to be inputted into MATLAB. If the image is of a handwritten number five, then the corresponding label would be the digit five. Thus, the computer begins to associate corresponding labels and images. In order to check the accuracy of the imaging, the label array and the images were matched together and outputted to the screen. At this point, all the input images and labels were properly matched, and ready to be used in the creation of support vector machines.

3.2 Classifying the Data

Ten SVMs were created to train the computer to recognize all ten digits. Each SVM created a hyperplane between each digit and the all the others. By creating ten of these, the digits could be classified with a high degree of accuracy. After the computer was trained, the testing images were classified in order to determine the accuracy of the SVM models. Using vector mathematics, the program decided which side of the boundary the new image vectors were on using inner products. The inner product revealed an angle from the boundary that revealed which side of the boundary the image vectors were on.

In order to classify each test image, it was first run through the classifier for zeros against the rest of the digits. If the program decided the image belonged to the ‘0’ class, it was determined to be a zero. Otherwise, the test image would then be run through the next classifier (this one for distinguishing ones from the other digits). The process was repeated until the program could conclusively determine which class the test image belonged in.

3.3 Implementing Multi-class Classification

After developing the boundary between two classes, the next step was teaching the computer to learn the difference between all 10 classes. For this challenge, one-vs-all classification was implemented. Essentially, 10 binary classifiers were established such that the zeros image vectors were separated from all the other digits and the ones image vectors were separated from all the other digits and so on.

3.4 Formulating the GUI

When the accuracy from the test data was sufficient, the classifiers could be used to read digits written by the user. In order to read handwritten digits, a webcam was connected to the computer, and a graphical user interface was created, containing a panel with the webcam feed and a button to interact with the program. The button was programmed so that when it was clicked, the image feed from the webcam was halted to display the captured image, and then after a second click the image was converted to the necessary format, displayed in the new format, and passed to the prediction program for classification. The webcam, a Logitech HD Pro C920, outputs a 1920px by 1080px image in full color. However, in order for the image to be classified, it must be of the same size and type as the images used to train the program, which were 28 by 28 grayscale images. In order to make the images in the right format, the image was reduced in size to 28 by 28px with imresize, and then it was converted into grayscale using the rgb2gray function. Most importantly,
since the images in the training data contained white digits on a black background and the test data will be black marker on a white paper, the image had to be inverted using the incomplement function and then darkened to increase readability. In order to be classified by the program, the modified image file had to be converted from a 2 dimensional 28 by 28px array into a 784 column wide row vector. Once in the proper format, the row vector was fed into the classification method, and then the label was displayed in a popup window, allowing the user to easily see the result.

4 Results and Discussion

4.1 SVM Classifiers

The Support Vector Machine (SVM) models were built successfully using the dataset of 60,000 training images. These training images were paired with their corresponding labels via the command “fitcsvm.” This command allowed the program to differentiate between one particular digit and the other 9 digits. After testing the SVM model built for the digit zero, the accuracy was determined to be 96.11 percent. This indicates that the program “learned” the boundary between images of zeros and images of all other digits. Furthermore, the input images of zeros were not the same images that the classifier was built using. This confirms that the machine learning technique worked reasonably well and that hyperplane in between each class and the rest of the digits was created accurately. After the initial classifier was built and tested, the other 9 classifiers were built and tested. The classification program was tested with 10,000 testing images the computer had never seen before. The program correctly classified 92.02 percent of the test images. This accuracy rate further affirmed the legitimacy of the classifiers and the theory of machine learning. The percentage also makes the classifier viable for commercial use.

4.2 GUI

The Graphic User Interface (GUI) also functioned successfully. It consistently and accurately took a grayscale picture of a 28x28px digit placed in front of the webcam. The program also functioned to successfully turn this 28x28 array of pixels into a row vector with 784 pixels. This success meant it was possible to give users the capability of scanning digits. Once each image is converted into a row vector, the information can then be processed with the classification script (see Appendix). See Figure 5 for a screenshot of the user interface for scanning an image:

![Figure 5: Picture of the GUI program in use](image)

While this GUI does not allow the user to scan more than one digit at a time, it still precisely vectorizes the input image. In addition, an individual image can be classified and displayed almost instantaneously and therefore the time required
to scan 5 digits would still be short enough to warrant using this methodology in commercial use.

4.3 Limitations/Issues

Although the classifiers were successful, the webcam was afflicted by light interference. This issue led to wildly inconsistent data in the incoming image pixels, leading the computer to correctly classify handwritten digits only 40 percent of the time. The margin of error in the webcam’s interpretation of data prevents the current form of this technology from being commercially viable.

5 Conclusion

5.1 Summary

Ultimately, machine learning is the future of large scale data scanning. A computer can process and sort handwritten zip codes significantly more efficiently than a human employee could. Yet, the nature of handwriting may always pose a challenge to machines due to its inherent variability and inconsistency. Thus, a large dataset of handwritten digits from the MNIST database was loaded into a Support Vector Machine Classifier, allowing the program to decipher handwritten digits with some degree of error. From here, these scanned digits can be compiled into a zip code and searched on a map. Using this algorithm, the user will be able to efficiently scan through large amounts of handwritten digits effectively.

5.2 Future Improvements

The main purpose for this research is to make handwriting recognition technology more efficient and more accurate. To start, the ability to read 5 digits simultaneously would help immensely in making the user experience quick and easy. This would, however, necessitate a program capable of parsing the incoming data from the webcam into 5 distinct vectors prior to entering the data into the classification algorithm. In addition, a larger initial training data set would raise the accuracy of the SVM classifiers. The more diverse the training set of the handwritten digits is, the better the program will be at deciphering between different digits. The only downfall with this improvement is the immense amount of time required to input such a large amount of data. With just 60,000 training images, the SVM classifiers still took between 4 and 5 hours to build. Thus, with more time and access to a larger data set of images, the classifier could conceivably be near-perfect at differentiating between different digits.

5.3 Applications

In addition to zip code scanning, this technology could be applied to any type of handwriting recognition platform. This method of loading many training samples into a classifier could be easily used for handwritten letters. Granted that letters are more intricate and numerous than digits are, alphabetical recognition would require an enormous training sample size in order to achieve reasonable accuracy. Nonetheless, the applications of this technology demonstrate the utility of machine learning.

6 Acknowledgments

The authors gratefully acknowledge Professor Waheed Bajwa of the Rutgers
Department of Electrical and Computer Engineering for guidance throughout this process, as well as Keon Kim, Dean Jean Patrick Antoine, Dean Ilene Rosen, and the NJ Governor’s School sponsors: Southern Jersey Industries, Lockheed Martin, Printrbot, Rutgers, the State University of New Jersey, and Rutgers School of Engineering.

7 References


Appendices

% This script builds the one-vs-all classifiers for each digit.

ImageID = fopen('/Users/bengiugliano/Downloads/train-images.idx3-ubyte')
% Read image file into 1 long column vector, 'LongImgVector'
longImageVector= fread(ImageID);
% Open Label file
labelID = fopen('/Users/bengiugliano/Downloads/train-labels-2.idx1-ubyte');
% Read Label file into column vector, 'LabelVector'
LabelVector = fread(labelID);
% Find all places in the label array where there is a 0 or 1, and place in
% an array.
check = -1;
runlength = 5421;
s = 'SVMModel';
% Delete header for labels
for h = 1:8
    LabelVector(1) = [];
end
% Duplicate LabelVector for later
LabelVectorCopy = LabelVector;
% Delete header for images
for k = 1:16
    longImageVector(1) = [];
end
% Create row vectors for all images
imageVector = zeros(784, 60000);
for i=1: 60000
    start=1+((i-1)*784);
    stop=1+(i*784);
    for t=start:(stop-1)
        imageVector((t+1-start),i)=longImageVector(t);
    end
    perc = (i./60000)*100
end
imageVector = transpose(imageVector);
start(:,:,)=[];
stop(:,:,)=[];
for count = 0: 9
    LabelVectorNew = num2cell(LabelVector);
    % SVM stuff
    check = check + 1
    currentname = strcat(s, num2str(check))
    classnamecurrent = num2str(check)
    allIndeces = find(LabelVector ~= count);
    currentClassIndeces = find(LabelVector == count);
    for p = 1:length(allIndeces)
        LabelVectorNew{allIndeces(p)} = 'all';
    end
    for v = 1:length(currentClassIndeces)
        LabelVectorNew{currentClassIndeces(v)} = classnamecurrent;
    end
    classifier = fitcsvm(imageVector, LabelVectorNew, 'KernelFunction','linear', 'Standardize',true,'ClassNames',{[all',classnamecurrent']});
    eval([currentname,'=',classifier;'']);
    % Restore Label Vector back to original state
    for h = 0:9
        LabelVectorNew{LabelVectorCopy == h} = h;
    end
end
disp('classifiers done')

% This script classifies test data based on the SVM Models

ImageID = fopen('/Users/bengiugliano/Downloads/t10k-images.idx3ubyte')
% Read image file into 1 long column vector, 'LongImgVector'
longImageVectorTest = fread(ImageID);
% Open Label file
labelID = fopen('/Users/bengiugliano/Downloads/t10k-labels-3.idx1-ubyte')
% Read Label file into column vector, 'LabelVector'
labelVectorTest = fread(labelID);
% Delete header for labels
for s = 1:8
    labelVectorTest(1) = [];
end
%% Delete header for images
for b = 1:16
    longImageVectorTest(1) = [];
end
predictImage = zeros(1, 784);
runlength = 100;
numberCorrect = 0;
for y = 1:runlength
    start = (1 + ((y - 1) * 784));
    stop = (1 + (y * 784));
    for f = start: (stop - 1)
        predictImage(1, (f + 1 - start)) = longImageVectorTest(f);
    end
    [label0, score0] = predict(SVMModel0, predictImage);
    [label1, score1] = predict(SVMModel1, predictImage);
    [label2, score2] = predict(SVMModel2, predictImage);
    [label3, score3] = predict(SVMModel3, predictImage);
    [label4, score4] = predict(SVMModel4, predictImage);
    [label5, score5] = predict(SVMModel5, predictImage);
    [label6, score6] = predict(SVMModel6, predictImage);
    [label7, score7] = predict(SVMModel7, predictImage);
    [label8, score8] = predict(SVMModel8, predictImage);
    [label9, score9] = predict(SVMModel9, predictImage);
    combinedScores(1) = score0(1, 2);
    combinedScores(2) = score1(1, 2);
    combinedScores(3) = score2(1, 2);
    combinedScores(4) = score3(1, 2);
    combinedScores(5) = score4(1, 2);
    combinedScores(6) = score5(1, 2);
    combinedScores(7) = score6(1, 2);
    combinedScores(8) = score7(1, 2);
    combinedScores(9) = score8(1, 2);
    combinedScores(10) = score9(1, 2);
    combinedScores = combinedScores * (-1);
    highProb = max(combinedScores);
    correctLabel = find(combinedScores == highProb) - 1;
    realLabel = labelVectorTest(y);
    if realLabel == correctLabel
        numberCorrect = numberCorrect + 1;
    end
    predictImage(:) = [];
end
perc = (y ./ runlength) * 100
accuracy = (numberCorrect ./ runlength) * 100
function varargout = gui(varargin)

% This script generates the GUI and classifies the handwritten digit

% GUI MATLAB code for gui.fig

% GUI, by itself, creates a new GUI or raises the existing singleton.*.

% H = GUI returns the handle to a new GUI or the handle to the existing singleton.*.

% GUI('CALLBACK', hObject, eventData, handles,...) calls the local
% function named CALLBACK in GUI.M with the given input arguments.

% GUI('Property', 'Value', ...) creates a new GUI or raises the existing singleton.*. Starting from the left, property value pairs are applied to the GUI before gui_OpeningFcn gets called. An unrecognized property name or invalid value makes property application stop. All inputs are passed to gui_OpeningFcn via varargin.

% *See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one instance to run (singleton)".

% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help gui

% Last Modified by GUIDE v2.5 16-Jul-2016 14:36:01

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1
gui_State = struct('gui_Name', mfilename, ...
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1})
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:})
else
    gui_mainfcn(gui_State, varargin{:})
end

% End initialization code - DO NOT EDIT

% --- Executes just before gui is made visible.
function gui_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
% varargin command line arguments to gui (see VARARGIN)

% Choose default command line output for gui
handles.output = hObject

% Update handles structure
global picture
global pause
guidata(hObject, handles)
axes(handles.axes1);
clear cam;
cam=webcam('Logitech');
pause=false
while pause==false
    picture=snapshot(cam);
    img = picture;
    image(picture)
end
clear('cam')
% UIWAIT makes gui wait for user response (see UIRESUME)
% uiwait(handles.figure1)

% --- Outputs from this function are returned to the command line.
function varargout = gui_OutputFcn(hObject, eventdata, handles)
% varargout cell array for returning output args (see VARARGOUT)
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton1 (see GCBO)
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)
global picture
global pause
pause = true
img = imresize(picture, [28 28]); % shrinks image to be
img = rgb2gray(img); % Makes image grayscale;
imgflipped = imcomplement(img); % flip the image so black is white, and vice versa.
imgflipped = imgflipped-130; % make it darker
% imgflipped=im2double(imgflipped);
colormap gray;
image(imgflipped);
colormap gray
predictImage = zeros(1,784);
currentIndex = 0;
for i=1:28 % rows 1-28
    for t = 1:28
        currentIndex = ((i-1)*28) + t;
        predictImage(1,currentIndex) = imgflipped(i,t);
    end
end
predictImage = double(predictImage)
assignin('base', 'predictImage', predictImage)

[label0, score0] = predict(SVMModel0, predictImage);
[label1, score1] = predict(SVMModel1, predictImage);
[label2, score2] = predict(SVMModel2, predictImage);
[label3, score3] = predict(SVMModel3, predictImage);
[label4, score4] = predict(SVMModel4, predictImage);
[label5, score5] = predict(SVMModel5, predictImage);
[label6, score6] = predict(SVMModel6, predictImage);
[label7, score7] = predict(SVMModel7, predictImage);
[label8, score8] = predict(SVMModel8, predictImage);
[label9, score9] = predict(SVMModel9, predictImage);
combinedScores(1) = score0(1,2);
combinedScores(2) = score1(1,2);
combinedScores(3) = score2(1,2);
combinedScores(4) = score3(1,2);
combinedScores(5) = score4(1,2);
combinedScores(6) = score5(1,2);
combinedScores(7) = score6(1,2);
combinedScores(8) = score7(1,2);
combinedScores(9) = score8(1,2);
combinedScores(10) = score9(1,2);
highProb = max(combinedScores);
correctLabel = find(combinedScores == highProb) -1;
disp(correctLabel)
result = int2str(correctLabel)
statement = ['I think it is a ' result ' .']
msgbox(statement, 'Result')