Sign Language Interpretation for Communication

Mahdieh Nejati Javaremi
Northwestern University
Master of Science in Robotics
m.nejati@u.northwestern.edu

Sun Yue
Northwestern University
Master of Science in Robotics
sunyue@u.northwestern.edu

Abstract

We use a combination of image analysis techniques and a nearest neighbor algorithm to design a sign language interpreter. In this paper, we apply Decision Tree, k-Nearest Neighbor, Naïve Bayes, and Random Forest machine learning techniques to correctly classify signed gestures belonging to the American Sign Language (ASL). Our analysis shows that 3-Nearest Neighbor is able to produce a model with a testing accuracy of 97%. Using 3-NN, we were able to develop a program that correctly classifies ASL signs and returns the correct letter from the English alphabet in under 10 seconds.

1. Introduction

The purpose of this project is to identify machine learning algorithms and classification methods that can accurately understand signed hand gestures in real time. To achieve this task, the input to our algorithm is one static image of a sign and the output is the identity of the sign.

Hand gesture recognition for human computer interaction is an area of active research in machine learning and computer vision. The applications of gesture recognition can be extended to several useful fields, especially in robotics and human-computer interaction. The primary goal of our gesture recognition project was to create a system which can identify American Sign Language (ASL) gestures and translate them to the written English alphabet.

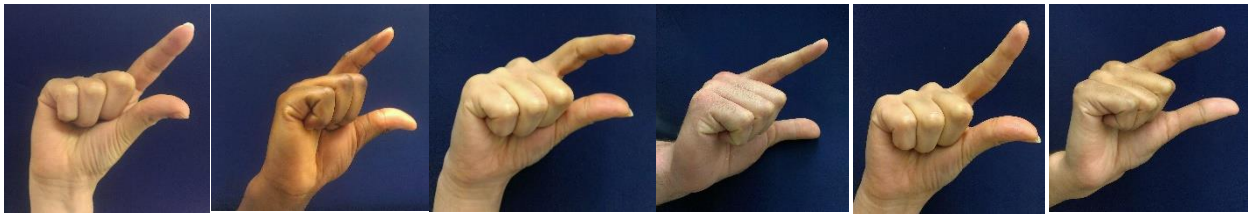
An estimated 360 million people worldwide suffer from hearing loss. Because the majority of hearing individuals do not understand sign language, communication between the hearing and the deaf can be challenging. A program such as what we have proposed in this paper is extremely important in the interest of breaking down this barrier.

Our goal was to create a learner that can recognize static signs with high accuracy. We aimed to optimize the speed with which the learner can understand the sign gesture. Thus, our measure of success throughout our investigation was both a measure of percent accuracy of detection as well as the time taken to identify signs.

2. Data set

For the purposes of our task, we did not find any suitable dataset online. The UCI repository contained some data for Australian Sign Language. This data was collected using sensing gloves, and contained information about the joint angles for each joint in the hand. Since our goal is to use a camera to input sign poses to the computer and classify them in real time, this data was not useful for our task. Thus, we collected our own data set. Pictures were taken from different individuals with different genders, hand sizes and skin colors. All pictures were taken with participant consent at Northwestern University by the authors.

Table 1: Eight instances of ASL for the letter G.



We have generated three different datasets, each comprising of their own training set and test set.

- Dataset 1:
  - 10 letters { B, E, F, H, J, L, O, V, W, Y }
  - 392 instances per letter
  - Total instances in data set = 3920
- Dataset 2:
  - 26 letters
  - 168 instances per letter
  - Total instances in data set = 4,368
- Dataset 3:
  - 26 letters
  - 392 instances per letter
  - Total instances in data set = 10,192

Each row in the dataset files constitute an example/instance, while each column is a feature/attribute. The final column corresponds to the classification. Our first dataset has 10 classes for each of the 10 letters in the dataset. The letters in the first dataset were chosen such that the ten signs were the most distinct from each other in a way that made them easiest for a human to differentiate between visually. The second and third datasets each have 26 classes corresponding to the 26 letters constituting the entire English alphabet.

Link to our datasets:
https://github.com/hereissunyue/machine_learning_project/tree/master/Dataset/Train%25Test

2.1 Data Partitioning

For the purpose of training, we use 80% of the total instances inside each of the three datasets. The remaining 20% is used for model testing. None of the testing instances are present in the training set. We later use new images captured by camera for the prediction step, described in section 4 of this report.

2.2 Data Preprocessing

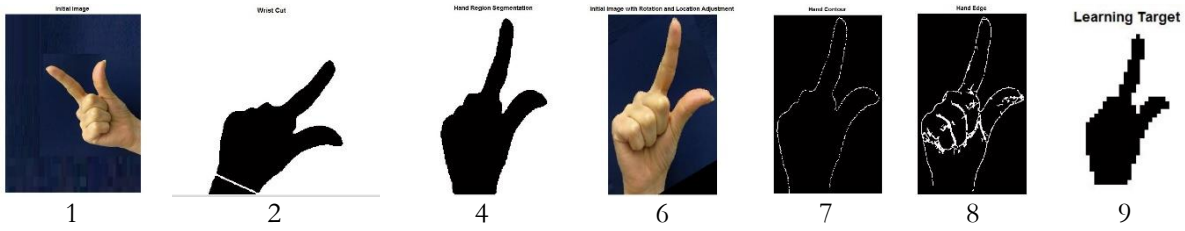
Data preprocessing, including hand segmentation and feature extraction, is the crucial first step for gesture recognition. The pre-processing stage prepares the input images and extracts useful features used later with our classification algorithms. The general approach used for the problem in hands consists of identifying the pixels in the image that constitute the hand, extract features from those identified pixels in order to classify the hand, and use those features to recognize the occurrence of specific pose sequences as gestures [1].

Our data preprocessing algorithm comprises of the following image processing methods:

1. Initial image is converted to a binary black and white image.
2. Wrist is detected and cut.
3. Hand region is segmented.

4. Hand segment is rotated and oriented properly.
5. Image is cropped to frame. If image is too small or too large, program will resize to predefined window of 38x23 pixels.
6. Image is converted back to RGB format after rotation and local adjustment.
7. Hand contour is detected.
8. Hand edges are detected.
9. Binary representation of the sign is developed.
10. Hand sign is represented as an array containing all grey image pixel values (1 or 0).

Table 2: Different steps during data pre-processing. The number below the images refers to the pre-processing step listed above.



Each image in the dataset is represented as an array, which is the output of the algorithm described above. For each image, the array is used as an instance; each value in the 38x23 array is a feature used for learning the model.

2.3 Data Processing

To expand our examples in our feature space and introduce error in our dataset, we used computer vision techniques to manipulate the raw images. Two methods were used:

- Erosion: Changes the morphology of the pose to mimic differing sizes. Erosion levels are set from 1 to 7 in increments of 1.
- Rotation: Rotate the gesture through different angles from –6 to 6 degrees of rotation in 2 degree increments.

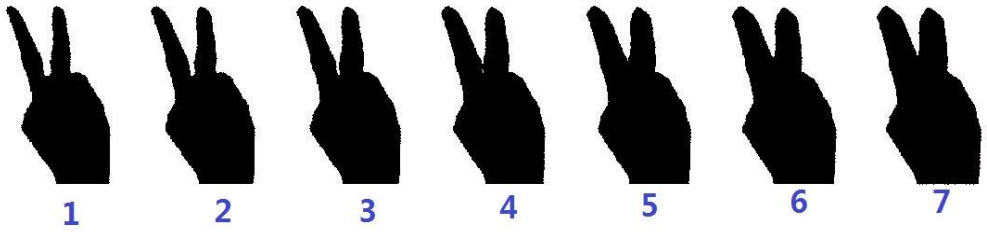


Figure 1 Instance Generation Method: Erosion Level

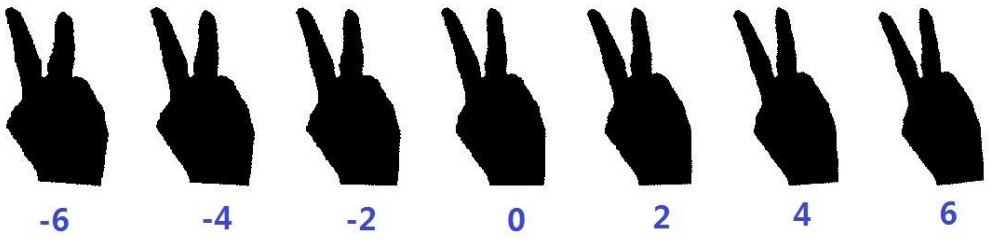


Figure 2 Instance Generation Method: Rotation / degrees

Link to the data-preprocessing code: https://github.com/hereissunyue/machine_learning_project

2.4 Attributes

Before algorithms can be applied to the task of learning from hand signs, proper feature selection is extremely important. We experimented with different features before settling on the current feature set. Please refer to Appendix A for a list of all previous features explored.

Our feature set is the final output of our image processing algorithm described above. The feature set is an array consisting of the pixel value in a predefined image frame of fixed size 38 x 23. This gives us an 874 dimensional features space. We originally had a larger image frame (380 x 230), however this greatly increased computational complexity without clear evidence of increased accuracy, as we could not run most classification algorithms on this feature space, even after increasing the Weka heap memory size.

The final feature space consists of 874 discrete binary attributes corresponding to the pixel value (0, 1).

3. Experimental Results

3.1 Learning techniques

The experimental results were achieved using Weka 3.6 on an Intel Core i5, 2.50 GHz, Windows 8, 64 bit operating system with 4.00 GB RAM. The results of the following five learning algorithms are listed in the tables that follow:

- Decision tree (J48)
- Random Forest
- 1 Nearest Neighbor
- 3 Nearest Neighbor
- Naïve Bayes

We also tried using Neural Networks and Support Vector Machines. With our feature space, the training times for these two algorithms were unreasonably high, taking more than 24 hours to compute. We were able to achieve a thorough analysis without the use of Neural Networks and SVMs, so we abandoned them during our analysis.

The five algorithms were applied to all three datasets. The training times, test times, and accuracies were measured and compared for the different learning methods. The training experiments were all performed with 10-fold cross validation. The 10-fold cross validation accuracy of the training data is then compared with test data accuracy.

3.2 Results

Dataset 1:

Table 3 Results of Training and Testing Classification Algorithms on Dataset 1

	Classifier	Decision Tree	Random forest	1-NN	3-NN	Naïve Bayes
Training set	Accuracy (%)	97.68	99.95	100	100	86.51
	Train time (s)	3.7	4.18	0.03	0.02	1.07
Test set	Accuracy (%)	85.618	97.55	98.16	98.06	81.43
	Test time (s)	0.01	0.01	14.3	9.78	2.87

Dataset 2:

Table 4 Results of Training and Testing Classification Algorithms on Dataset 2

	Classifier	Decision Tree	Random forest	1-NN	3-NN	Naïve Bayes
Training set	Accuracy (%)	95.65	100	100	100	72.57
	Train Time(s)	2.95	3.01	0.03	0.02	0.49
Test set	Accuracy (%)	62.70	87.30	95.24	95.60	60.71
	Test time (s)	0.01	0.02	14.83	10.1	5.01

Dataset 3:

Table 5 Results of Training and Testing Classification Algorithms on Dataset 3

	Classifier	Decision Tree	Random forest	1-NN	3-NN	Naïve Bayes
Training set	Accuracy (%)	96.78	99.98	100	99.98	72.06
	Train time (s)	17.12	12.62	0.09	0.02	1.16
Test set	Accuracy (%)	74.13	93.92	96.46	96.78	65.30
	Test time (s)	0.01	0.02	15.3	10.3	11.53

3.2 Discussion

The results using the first data set are better than the results from the second and third dataset for all five learning algorithms. This is because the first dataset consist of ten signs that are quite distinguishable from each other such that chances of misclassifying a sign are reduced. We had designed the dataset with this prediction in mind, and were happy to see the results matched our predictions. The second and third dataset contain the full alphabet, where the following set of signs are very similar to each other with only small positional variations that are even difficult for an untrained human to distinguish. The following three set of letters have very similar signs, and the individual classification accuracy of these are small with higher rates of error, which reduces the overall accuracy of the second and third dataset.

1. {a, c, e, s, t}
2. {m, n}
3. {r, u, z}

To explore this further, we can compare the confusion matrix using 3-NN on the test set for both Dataset 1 and Dataset 3. As shown in Figure 3, we can see that for 8 out of the 10 letters in Dataset 1, the percentage of correct classification is 98%. The sign "A" is misclassified as the letter "B" for 7% of the test data, and the sign "D" is misclassified as the letter "C" for 12% of the test data. Nevertheless, all letters have a percentage of correct classification *greater than* 90%. However, misclassified instances are greater for Dataset 3, as shown in Figure 4. We can see that the signs "C, N, R, and Z" have a value of *less than* 90% in the confusion matrix shown in Figure 4. The sign "C" is misclassified as the letter "E" for 2% of the data and as "S" for 16% of the data; the sign "N" is misclassified as the letter "M" for 12% of the test data; the sign for "R" is misclassified as the letter "U" for 2% and the letter "D" for 13% of the test data; the sign "Z" is misclassified as the letter "U" for 10% of the test data.

```
=== Confusion Matrix ===

  a  b  c  d  e  f  g  h  i  j   <-- classified as
 91  7  0  0  0  0  0  0  0  0 |  a = B
  0 98  0  0  0  0  0  0  0  0 |  b = E
  0  0 98  0  0  0  0  0  0  0 |  c = F
  0  0 12 86  0  0  0  0  0  0 |  d = H
  0  0  0  0 98  0  0  0  0  0 |  e = J
  0  0  0  0  0 98  0  0  0  0 |  f = L
  0  0  0  0  0  0 98  0  0  0 |  g = O
  0  0  0  0  0  0  0 98  0  0 |  h = V
  0  0  0  0  0  0  0  0 98  0 |  i = W
  0  0  0  0  0  0  0  0  0 98 |  j = Y
```

Figure 3 Figure 3 Confusion Matrix from testing 3-NN trained model on Dataset 1

```
=== Confusion Matrix ===

  a  b  c  d  e  f  g  h  i  j  k  l  m  n  o  p  q  r  s  t  u  v  w  x  y  z   <-- classified as
 98  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |  a = A
  0 98  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |  b = B
  0  0 80  0  2  0  0  0  0  0  0  0  0  0  0  0  0 16  0  0  0  0  0  0  0  0 |  c = C
  0  0  0 95  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  3 |  d = D
  0  0  0  0 98  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |  e = E
  0  0  0  0  0 98  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |  f = F
  0  0  0  0  0  0 94  0  0  0  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |  g = G
  0  0  0  0  0  0  0 98  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |  h = H
  0  0  0  0  0  0  0  0 98  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |  i = I
  1  0  0  0  0  0  0  0  0 97  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |  j = J
  0  0  0  0  0  0  0  0  0  0 91  7  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |  k = K
  0  0  0  0  0  0  1  0  0  0  0 97  0  0  0  0  0  0  0  0  0  0  0  0  0  0 |  l = L
  0  0  0  0  0  0  0  0  0  0  0  0 98  0  0  0  0  0  0  0  0  0  0  0  0  0 |  m = M
  0  0  0  0  0  0  0  0  0  0  0  0 12 86  0  0  0  0  0  0  0  0  0  0  0  0 |  n = N
  0  0  0  0  0  0  0  0  0  0  0  0  0  0 98  0  0  0  0  0  0  0  0  0  0  0 |  o = O
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 98  0  0  0  0  0  0  0  0  0  0 |  p = P
  0  0  0  0  0  3  0  0  0  0  0  0  0  0  0  0 95  0  0  0  0  0  0  0  0  0 |  q = Q
  0  0  0 13  0  0  0  0  0  0  0  0  0  0  0  0  0 83  0  0  2  0  0  0  0  0 |  r = R
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 98  0  0  0  0  0  0  0 |  s = S
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 98  0  0  0  0  0  0 |  t = T
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 98  0  0  0  0  0 |  u = U
  0  0  0  0  0  0  0  0  0  8  0  0  0  0  0  0  0  0  0  0  0 90  0  0  0  0 |  v = V
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 98  0  0  0 |  w = W
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 98  0  0 |  x = X
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 98  0 |  y = Y
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 10  0  0  0  0 88 |  z = Z
```

Figure 4 Confusion Matrix from testing 3-NN trained model on Dataset 3

Data set 3 has more instances than data set 2 (392 per letter vs. 168 per letter). The classification accuracy for dataset 3 is higher for all five classification methods compared to dataset 2, which agrees with our expectation that accuracy increases with an increased number of training instances.

From the five experimented classification algorithms, Nearest Neighbor with k=3 is the most accurate with reasonable computation time, while 1-Nearest Neighbor is the runner up. With k-NN algorithm, model training time is really fast. However, it is also the longest of all algorithms during test time; averaging at around 10 seconds for 3-NN. In terms of accuracy, Random Forest is comes in third, while taking a longer time to train, but shorter time to test compared to k-NN.

In order to get 1-NN and 3-NN to work with our largest dataset, we had to change the heap size of the Weka memory. Otherwise, we would get a low memory exception.

From our analysis we concluded that we have a suitable feature set, since we can learn reasonably accurate models with greater than 97% accuracy on test data. Therefore, we used our learned model to predict hand gestures unknown to the computer.

4. Prediction Step

For the prediction step the 3-NN classifier was chosen because for the purposes of our demonstration we wanted high accuracy while a processing time of 10 seconds was still acceptable.

In the demonstration video we use a webcam to take a snapshot of a signed gesture as input. The input was taken from an individual whom we had not used in any of our dataset before. The image is processed using our data-preprocessing algorithm written in Matlab as described in section 2.2 of this report. The program outputs the ASL sign in the form of a 38x23 array of pixel bits (the feature array). Given the array of bits, the 3-NN classifier searches through the training set to find the three observations with the closest feature array to the presented sign. The program prints to screen the letter in the English alphabet corresponding to the 3 closest neighbors.

In the demonstrative video we show our program was able to successfully classify all the letters in the word "hello" correctly.

A video of the working model is available on our project website: www.mahdiehn.wix.com/aslinterpreter

5. Conclusion

We were able to successfully collect data of ASL gestures and represent the images by their most important feature array. We found that the most important feature set to be an array of bit values after nine image processing steps. We were also able to test different supervised classification machine learning methods on our dataset and compare their accuracies. By doing so, we were able to choose the most accurate classifier for our prediction step. This was the 3 Nearest Neighbor classifier with a classification accuracy of 96.78% on test data and a run-time of about 10 seconds. Finally, we successfully implemented our data preprocessing code along with our chosen high accuracy classifier into a program that can correctly classify signed gestures from individuals outside of the training and test dataset space and return a letter, all in real time.

6. Future Work

There are several things that can help extend this project further:

- The training dataset contains signs from individuals in their late teens, 20s and 30s. A good addition would be including signs form younger and older individuals.

- Implementing an algorithm that can detect a sequence of signs, not just static images. Many ASL phrases are dynamic hand gestures. Currently, our code can only classify static signs.
- Trying to extract out the most informative features from our feature array to reduce dimensionality with the hopes of increasing the prediction speed of our algorithm.

7. References:

[1] "A Comparison of Machine Learning Algorithms Applied to Hand Gesture Recognition." A Comparison of Machine Learning Algorithms Applied to Hand Gesture Recognition. N.p., n.d. Web. 28 May 2015. Retrieved from: https://www.academia.edu/4859499/A_comparison_of_machine_learning_algorithms_applied_to_hand_gesture_recognition

[2] E. Alpaydin. Introduction to Machine Learning. Second Edition. The MIT Press. 2010. Ch 10. pg 230.

[3] "Beginner's Guide to Understand Fingertips Counting Using Convexity Defects in OpenCV." - CodeProject. N.p., n.d. Web. 28 May 2015.http://www.codeproject.com/Articles/782602/Beginners-guide-to-understand-Fingertips-counting

[4] "A Dynamic Gesture Recognition System Based on CIPBR Algorithm." A Dynamic Gesture Recognition System Based on CIPBR Algorithm. N.p., n.d. Web. 28 May 2015.

[5] "Thomas Royal." A High Level Description of Two Fingertip Tracking Techniques: K-curvature and Convexity Defects. N.p., n.d. Web. 28 May 2015.

[6] Dhawan, Amiraj. Implementation of Hand Detection Based Techniques for Human Computer Interaction (n.d.): n. pag. Web.

Appendix A

Explored Features

We started by identifying the following set of attributes:

1. Mean and variance of gray pixel values: (not very useful, if hand is changed a little, the difference is huge)
2. Hough line angles: Finding the straight lines, and comparing the angles of prominent lines for each gesture
3. Blob area → informative feature
4. Blob perimeter → informative feature
    - Detect corner points and use the coordinates of those points to classify gestures
5. Blob bounding box:
6. Blob number of convexity defects:
    - Convexity Defects is a feature that finds the defects between a convex hull and a contour; those defects are useful to find features in a hand, as for example the number of extended fingers.
7. Directional gradient of image
8. Orientation histogram (histogram of orientated gradients HOG)
9. Radial signature:
    - distance from the hand centroid to the edges of the hand along a number of equally spaced radials
    - Find 100 equally spaced radials:
        a. count the number of pixels along a given radial (eliminating those that are not on the hand)
        b. All the radial measurements can be scaled so that the longest radial has a constant length. With this measure, we can have a radial length signature that is invariant to hand distance from the camera.
10. Edge detection with erosion and dilation → informative feature
11. Binary representation of image (matrix of binary pixels) → most informative feature

Appendix B

American Sign Language Gestures