American Sign Language Character Classification

Anthony Allen, Parth Patel, Caijun Qin
ML_Lab 4/20/20

***Abstract*— This report follows the experimental processes implemented in the creation of a machine learning American Sign Language Character Classification (ASL) algorithm. Students in a Special Topics Machine Learning class at the University of Florida created a dataset of 1844 images of ASL letters from A-I. The dataset was analyzed machine learning classifiers such as Linear Discriminant Analysis, k-Nearest Neighbors (kNN), Decision Tree, Random Forest, Support Vector Machine and Multi-Layer Perceptron. The goal was to create and algorithm that could achieve an accuracy score of 90% or greater when tested against an unseen dataset. The kNN classifier producing the best accuracy score of 90.5% on a local dataset segmented into training and test datasets. It was further bombarded with evaluative experiments in attempts to increase this score. The model was tested against various preprocessing algorithms as well as feature generating functions. Ultimately, the best results were achieved with a simple pixel normalization technique and no feature generation.**

I. Introduction

THIS report delineates an experimental design of a machine learning algorithm capable of reading letters in American Sign Language (ASL). The model presented here will use an image dataset created by the authors of this report combined with data supplied by a Special Topics Machine Learning class, EEL4930, at the University of Florida. This dataset provides 20 images from each group (with approximately 10 groups) of each ASL letter from A – I as defined by the American Sign Language University (ASLU) [1].Vectorized and pixel normalized data with a HOG feature extraction and kNN classifier was experimentally determined to be the best combination of processing and classification techniques. The K-Nearest Neighbors Classifier compares a test point to a set amount (k) of nearest training data points and then estimates an output value based on the actual output values of the k number of nearest training points, as discussed in Lecture 12 [2]. In order to find the k nearest neighbors in the training data, there needs to be a measure for similarity or dissimilarity, and in the case of the K-Nearest Neighbors Classifier, these measures are distance-based, such as Euclidean and Minkowski Distance metrics. Once the k nearest neighbors to the test point in the training data are found, the class label of the test point can be determined using a majority vote scheme. The experimentation of this model in its entirety can be organized into three categories including pre-processing, feature extraction, and model selection and evaluation.

II. Implementation

The model designed by the group can be categorized into 3 steps: pre-processing steps, model training, and model evaluation. The dataset provided for the model was formatted as RGB images that altogether was represented as a four-dimensional array. The group experimented with many methods to pre-process the data in order to streamline performance and ultimately improve model accuracy. The original dataset had several issues and required modifications in order to have each image on the same scale in relationship to each other. The pre-processing steps included converting each image to grayscale to focus only on pixel intensity instead of RGB values, flattening each image to be a 2-dimensional array in order to reduce dimensionality and to be compatible with the model pipeline. Each pixel was then normalized to reduce variance in the dataset and have the pixel values be on the same scale of between 0 to 1. Immediately below is the pseudocode for each step of preprocessing.

```
from skimage.color import rgb2gray
#image greyscale
greyscaled_images = rgb2gray(image for each image in
dataset)
```

```
import numpy as np
#image vectorization
vectorized_images = np.asarray(image.flatten() for each
image in greyscaled_images)
```

```
#pixel normalization
normalized_images = vectorized_images/255
```

The actual classification model used was a k-Nearest Neighbors Classifier. Based on comparisons between different models in terms of performance and accuracy, the k-Nearest Neighbors Classifier gave the best accuracy measure on the preprocessed dataset. By testing for the best parameters, the group found that using a single neighbor with Euclidean Distance for the power parameter for the Minkowski Metric

offered the best accuracy. We split the dataset into a training and testing set, and trained the model based on the training set, then evaluated its performance using the score function associated with the model with the testing dataset. Below is the pseudocode describing the model training and evaluation steps.

```
#training and testing sets
x_train, x_test, y_train, y_test = train_test_split(original
dataset, class labels, test_size=0.2)

#Model
from sklearn.neighbors import KNeighborsClassifier
KNN_Model = KNeighborsClassifier(n_neighbors=1,
weights='uniform', metric='minkowski', p=2))

#run pre-processing steps before model training and scoring

KNN_Model.fit(x_train, y_train)
KNN_Model.score(x_test, y_test)
```

III. Experiments

The project was implemented using Jupiter Notebook through anaconda. The idea was to test out several different machine learning algorithms on the provided dataset to see which ones performed the best. Before any of this could happen however, the ASL images needed to be pre-processed.

A. *Pre-Processing*

Various Pre-Processing methods were analyzed to determine the best representation of the dataset. This was an important initial step because 1) it provided familiarity of the dataset and 2) it reduced the data to the same scale allowing equal analysis of all components without large variables overshadowing smaller ones. This was a necessary step as the training set supplied by the class had some fundamental issues. Many of the images were taken at an angle, offering a nice skewing variable to be accounted for. Most of the supplied images were cropped. However, in a small percentage of them, the entire arm of the person signing was present. Many students collected images against a background of a similar color to their skin which may make segmentation difficult. Additionally, many of the students collected images atop a marble background which introduces a "salt and pepper" issue. Luckily, it appears that all images were taken with the right hand. The normalization techniques considered for this dataset were as follows.

1) *RGB Color channels normalization:*

This technique implemented the following equation

$$New\ image\ = \frac{old_{image}}{S}$$

Where

$$S = redchanel + greenchanel + bluechanel$$

The goal of this approach was to offer a sense of color constancy where objects are assigned a relatively constant color regardless of illumination.

2) *Pixel normalization*

RGB channels were transformed into grayscale and their intensity was reduced by a factor of 255. This provided intensity values between 0 and 1 and condensed the data to a single channel. The goal of this approach was to offer a more manageable dataset that had a significantly smaller variance in pixel intensity as compared to the original dataset. This would help make any optimization processes numerically stable.

3) *Min-Max scaling*

$$Xtrain\ = \frac{Xtrain - Xtrain.min(\ )}{Xtrain.max(\ ) - Xtrain.min(\ )}$$

Min-max scaling offers a similar numeric reduction as pixel normalization with increased sparsity. Instead of limiting the 0 to 255-pixel intensity ranges to values between 0 and 1, the range is reduced and spread across a range between -1 and 1. This gives a mean offset of 0. Again, the goal was to make the data more numerically manageable and reduce pixel intensity variance, as well as to see how it compares to other normalization techniques. However, this technique is most useful when normalized values have a gaussian distribution.

4) *Standard Scaler approach*

This approach implements the same technique as the min-max scaling normalizer. However, it ensures that the mean value is in fact 0. This is particularly useful when the pixel values do not have a gaussian distribution.

5) *Decision*

Pixel normalization was chosen as the starting pre-processing method because 1) it was the easiest to implement and 2) it produced the best results when analyzed against the finished pipeline (Table III).

B) *Feature Generation*

The next step of analyzing the class dataset involves generating features from the training set and assessing their quality. This is done by computing a set of cluster validity-type metrics. The feature generators analyzed were Principal Component Analysis (PCA) and Histogram of Oriented Gradients (HOG). Data that had been pixel normalized was used in the analysis of these features for simplicity. The feature generators were compared to each other using the following indexes:

1) Calinski-Harabasz Index:
   The index is the ratio of the sum of between-clusters dispersion and of inter-cluster dispersion for all clusters (where dispersion is defined as the sum of distances squared)[2].
2) Davies-Bouldin Index:
   This index computes the average similarity between clusters, where the similarity is a measure that compares the distance between clusters with the size of the clusters themselves[2].
3) Silhouette Index:
   The silhouette index measures how well class group cluster together it estimates the average distance between clusters[2].

TABLE I
INDEX EVALUATION OUTPUTS

Index	Output
Calinski-Harabasz Index (HOG)	10.3139
Davies-Bouldin Index (HOG)	8.9198
Silhouette Index (HOG)	0.0510
Calinski-Harabasz Index (PCA)	3.5592
Davies-Bouldin Index (PCA)	13.1797
Silhouette Index (PCA)	-0.0102

Calinski helps choose the proper clustering. From table I, we see that the index for the HOG method is much higher than that of the PCA method. Due to this fact we see that the HOG method supplies a better solution in this case. Furthermore, for the Davies-bouldin index, we see that the value for HOG is lower which again results in a relatively better clustering. Lastly, the Silhouette index from HOG has the closest value to 1. Therefore, the HOG approach is best in this case. Overall, we see that that HOG feature extraction produces the best cluster over all forms of analysis.

B. Model Selection

The machine learning models analyzed against this data were Linear Discriminant Analysis (LDA), k-Nearest Neighbors (kNN), Decision Tree, Random Forests, Support Vector Machines(SVM), and Multi-Layer Perceptron (MLP). The goal was to find a classifier that produced the greatest accuracy in testing. All models were analyzed without feature extraction and with pixel normalized data initially. The best training and testing accuracies for each model as well as mean absolute error are listed in table II.

1) LDA

Six components were deemed ideal for the LDA classifier as this quantity is the smallest value that explains at least 90% of the variance. Upon visual inspection of this variance however, the clusters mean did not appear to be very spare. Consequently, LDA did not produce the most effective model for this data.

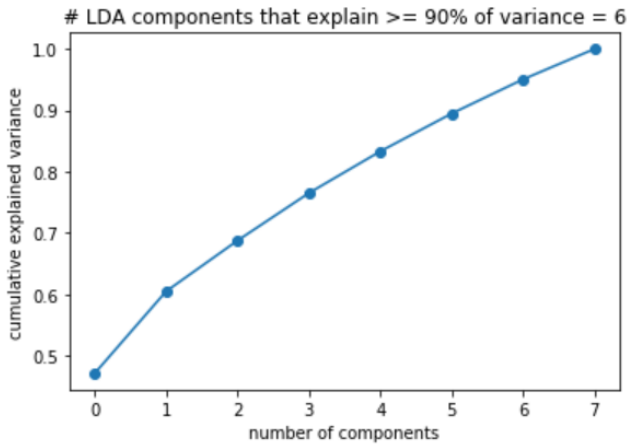

Figure 1 Image of the LDA number of components vs cumulative explained variance

2) kNN

The accuracy of the kNN classifier was tested against multiple parameters: number of nearest neighbors, weight, and metric. Uniform and distance weights as well as Minkowski and Euclidean metrics were used to analyze the data. The four parameters were tested in all configurations. The optimal kNN classifier for this project is uniform weights, Euclidean and n_neighbors = 1. Euclidean distance metric was analyzed because it normalizes the distances in a consistent way. The Makowski method was also chosen where p = 1 because it represents the distance with an absolute sum. This creates a sense of sparsity. The hope was that this sparsity would increase class detectability. However, this was not the case. Figure 3 depicts a plot of the optimally parameterized kNN classifier at different k neighbors sizes. This classifier was ultimately chosen to be used in the machine learning pipeline because it produced the highest accuracy score as well as the lowest mean absolute error as seen in table II.

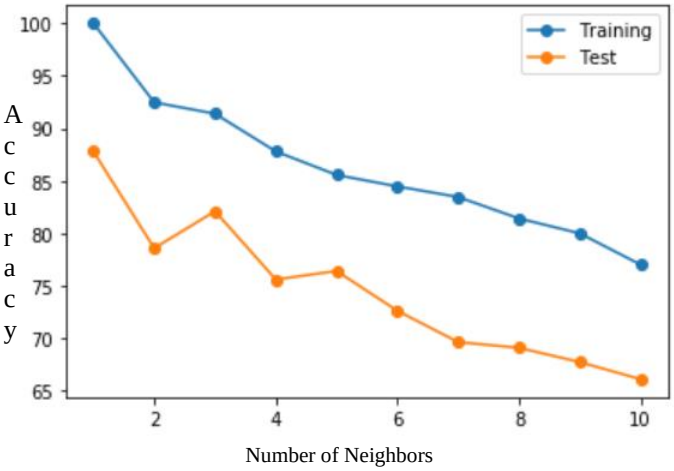

Figure 2 This plot depicts an accuracy (y-axis) vs number of neighbors (x-axis) of the a kNN classifier with weights = uniform and metric = Euclidean.

3) Decision Tree

Image data was processed against a Decision Tree classifier with variable max depth and min leaf size parameters. These two components were analyzed in a grid search to produce optimal values of max_depth = 10 and min_leaf = 4. Upon viewing the initial tree, the testing accuracy of the model was barely over 50%. The model had many stray branches near the bottom. Intuitively, the best way to eliminate these branches is typically by reducing the depth or min leaf size. Hence why these parameters were chosen. Unfortunately, the optimal model produces not much better accuracy than the original model. Decision Tree likely performed poorly because of overfitting. This is apparent because the training accuracy is substantially higher than the testing accuracy.

4) Random Forest

The accuracy of a random forest applied to the data was analyzed against a varying bootstraps number parameter. The graph created indicates that the best bootstrape size is 20 as it produces the highest testing accuracy. The number of bootstraps was chosen as the parameter to analyze because it directly correlates to the number of decision tree models used in the forest. Random forest performed okay relative to the others. However, it still indicated signs of overfitting with its high training data accuracy and low testing data accuracy. Its

deficiency is likely a result of the dataset and the buckets of misclassified probabilities from the decision tree.

5) SVM

Support Vector Machine algorithm using a Radial Basis Function was analyzed using a grid search to find the optimal values of C and gamma. Where C is the regularization coefficient and controls the margin sizes and gamma controls how far the influence of each training value reaches. These parameters were chosen because the model is very sensitive to gamma and again C is a regularization parameter. SVM is not generative. This classifier fine tunes a kernel and analyzes the data resulting from it. Thus, it is able to find linear relations from nonlinear data. There is likely a non-linear correlation in the data that only SVM is seeing.

6) MLP

ASL data was processed against a multilayer perceptron as a function of batch size. The accuracy for the best-case batch size of 72 is listed in table II. MLP didn't do well because of the size and quality of the dataset. It was also extremely computationally expensive as the growing parameters built more layers and more computation.

TABLE II
BEST ACCURACY SCORES FOR EACH MODEL

Classifier	Parameters	Training Accuracy	Testing Accuracy	Mean Average Error
LDA	N_components = 6	93.22	60.43	2.08
kNN	N_neighbors = 3	96.80	90.50	0.35
Decision Tree	depth=12 leaf = 3	85.67	53.46	1.52
Random forest	N_Bins = 50	100	79.40	1.52
SVM	C = 20 Gamma = 1	96.45	83.97	0.45
MLP	Batch = 72	73.21	64.58	2.85

C. Improving kNN

Further experiments were conducted to improve the accuracy of kNN. The first analysis of this model was conducted without using a feature extraction algorithm. Since kNN typically does not work well with large dimensions, and since HOG performed the best in prior analyses, HOG was added to the pipeline for testing. Despite using a grid search to find the best parameters, the accuracy of the model was reduced to 60%. Additional tests were conducted with PCA just for good measure. However, the accuracy was again reduced to 43%.

Further preprocessing approaches were analyzed as well since kNN is typically sensitive to noisy data. Edge detection, masking and image smoothing were used with and without the HOG feature extractor. The goal of masking was to remove the background in all the images by setting pixel values to zero and object values to 1. This helps isolate the signing hand. The idea was to reduce the influence of the background pixels on the classification algorithm. Edge detection serves the same purpose as masking in the sense that it identifies the signer's hand and makes it more pronounced. Lastly, image smoothing acts to blur the images, thus reducing the effect of noise on the dataset. The effects of these experiments on the accuracy of the model are listed in Table III.

TABLE III
ACCURACY SCORE OF PIPELINE NORMALIZATION VS FEATURE EXTRACTION

	NONE	HOG	PCA
Pixel Normalization	90.50	57.99	53.12
Min-Max	84.23	52.85	51.28
Standard scalar	86.44	55.55	52.85
Robust Scalar	86.17	88.08	51.49
Smoothing	88.90	88.89	58.26
Sobel	65.32	70.73	38.48
Masking	65.32	62.30	13.45

* scores represent percentages

IV. CONCLUSION

Overall, the kNN classifier achieved the desired goal of a 90% accuracy score. Surprisingly, this was done without feature generation. As seen in table III, using a HOG feature generation with a robust scalar produced an accuracy just 2 percentiles shy of the accuracy goal. This implies that feature generation can be useful in identifying the output images. Further analysis on preprocessing techniques may be worth looking into for future testing. With accuracy scores fluctuating all over the place because of normalization techniques, the greatest take away, was just how important normalizing data actually is.

REFERENCES

[1] American Sign Language University, "ASLU," [Online]. Available: https://www.lifeprint.com/. [Accessed 10 February 2020].

[2] C. Silvia, "MachineLearning-S20/Lectures," GitHub, 16-Feb-2020. [Online]. Available: https://github.com/MachineLearning-S20/Lectures/tree/master/Lecture 12 - Decision Trees & Random Forests. [Accessed: 20-Apr-2020].