American Sign Language Recognition using Machine Learning Algorithms

First Assignment of the Curricular Unit TAA.

João Pedro Saraiva Borges
DETI
Universidade de Aveiro
Nº Mecanográfico 98155

Daniela Filipa Pinto Dias
DETI
Universidade de Aveiro
Nº Mecanográfico 98039

***Abstract*—Hand gestures and sign language are the most commonly used methods by deaf and non-speaking people to communicate among themselves or with speech-able people. However, it takes a lot of effort to learn and to be able to communicate with this group of people. For this reason, building a system that recognizes hand gestures and sign language can be very useful to facilitate the communication gap between speech-able and speech impaired people.**

I. Introduction

Sign language builds a great bridge of communication between impaired people themselves and speech-able ones; however, it's a shame that we still have a significant communication gap because of the effort that needs to be done to learn this language. Hence, trained sign language interpreters are necessary tools to close this gap, with a seen increasing demand over the years in many areas, such as health, law and education.

This report will give a walkthrough of the used algorithms and the conclusions taken with a data set containing 27,455 training cases and 7,172 testing cases. We have taken multiple approaches to recognize gestures in sign language, including One-vs-All Classification (Logistic Regression Classifiers with or without Stochastic Gradient Descent), K-Nearest-Neighbour, Convolutional Neural Networks and Support Vector Machine (Linear or with One-vs-All decision function).

Rest of this research is organized as follows: Section 2 gives a analytic overview of the chosen dataset. Section 3 goes into detail over all the algorithms and methodologies implemented for this study, along with the results obtained. Finally, Section 4 finishes this study with an overall view of all our conclusions and compares the algorithms according to their performance.

II. Sign Language Data Set

A. The Data Set

The Data Set used for analysis in this project was obtained from the website Kaggle, with the name of *Sign Language MNIST - Drop-In Replacement for MNIST for Hand Gesture Recognition Tasks* [1]. It contains 28x28 images of all the hand signs according to the American Sign Language, except for the J and Z which need motion in their execution.

This dataset has undergone multiple transformations, including cropping to hands-only, gray-scaling, resizing, and then creating at least 50+ variations to enlarge the quantity of training data.

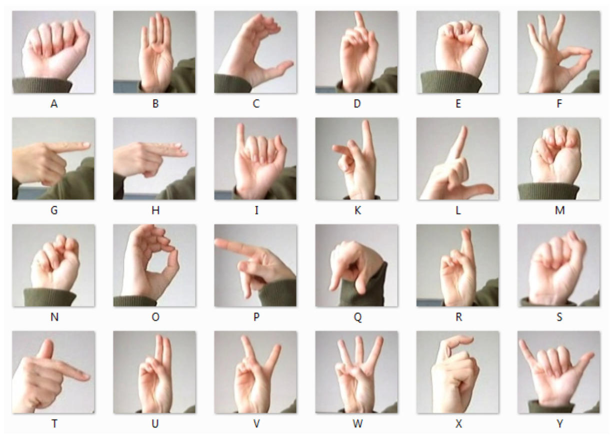


Fig. 1. American Sign Language Alphabet

It takes as reference the classic MNIST image dataset, which is a dataset containing only handwritten numbers, very popular for testing image-based machine learning methods. The Sign Language MNIST follows the same CSV format with labels and pixel values in single rows, representing a multi-class problem with 25 classes of letters.

B. The Composition of the Dataset

In this dataset, we have 2 CSV files which contain the test dataset (7,172 cases) and the train dataset (27,455 cases). The CSV files are structured in the following way:

- Headers: The first line contains the column headers (label and pixels).
- Labels: In the first column, the labels identify the letter in a one-to-one map for each alphabetic letter from A to Z (0-25).
- Pixels: Each row has 785 columns, being the first one the label, and the rest of them the (28x28) image pixels - each pixel is identified in a scale of grays from 0 to 255.

Fig. 2. An example of an image in the dataset

C. Statistical Analysis

The number of instances of each letter in the training and testing data sets is not uniform. The next figure shows, for example, that there is a larger amount of letters with the labels ranging from around 17 to 19.

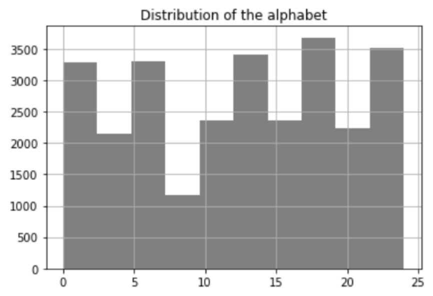


Fig. 3. Distribution of the alphabet histogram

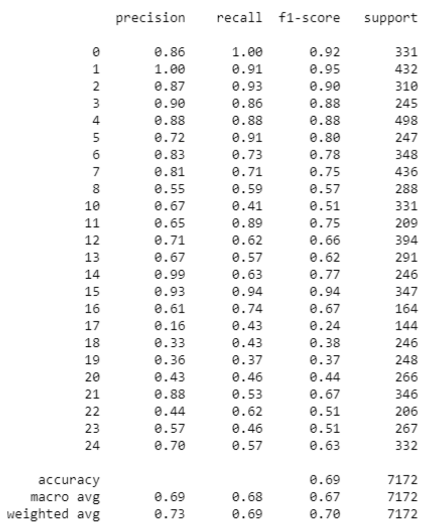
III. Methodologies

A. One-vs-All Classification

We first implement the One-vs-All Classification by training Regularized Logistic Regression Classifiers for each of the letters in the MNIST dataset, with:

- Penalty set to ”l2”, in other words, the regularization is made possible with Ridge Regression
- Solver set to ”lbfgs”, standing for Limited-memory Broyden–Fletcher–Goldfarb–Shanno Algorithm (an optimization algorithm that uses a limited amount of computer memory)
- Max iteration set to 10000
- Other parameters set to default

With this algorithm, we achieved 69% of accuracy performance.



```
              precision    recall  f1-score   support

           0       0.86      1.00      0.92       331
           1       1.00      0.91      0.95       432
           2       0.87      0.93      0.90       310
           3       0.90      0.86      0.88       245
           4       0.88      0.88      0.88       498
           5       0.72      0.91      0.80       247
           6       0.83      0.73      0.78       348
           7       0.81      0.71      0.75       436
           8       0.55      0.59      0.57       288
          10       0.67      0.41      0.51       331
          11       0.65      0.89      0.75       209
          12       0.71      0.62      0.66       394
          13       0.67      0.57      0.62       291
          14       0.99      0.63      0.77       246
          15       0.93      0.94      0.94       347
          16       0.61      0.74      0.67       164
          17       0.16      0.43      0.24       144
          18       0.33      0.43      0.38       246
          19       0.36      0.37      0.37       248
          20       0.43      0.46      0.44       266
          21       0.88      0.53      0.67       346
          22       0.44      0.62      0.51       206
          23       0.57      0.46      0.51       267
          24       0.70      0.57      0.63       332

    accuracy                           0.69      7172
   macro avg       0.69      0.68      0.67      7172
weighted avg       0.73      0.69      0.70      7172
```

Fig. 4. Table of Data Retrieved

A confusion matrix was made, in which the rows symbolize the actual label, and the columns the label predicted by the algorithm; this way, we can see which were predicted correctly (we can easily see by accompanying the diagonal traced), and the amount of incorrect predictions made with another specific letter.

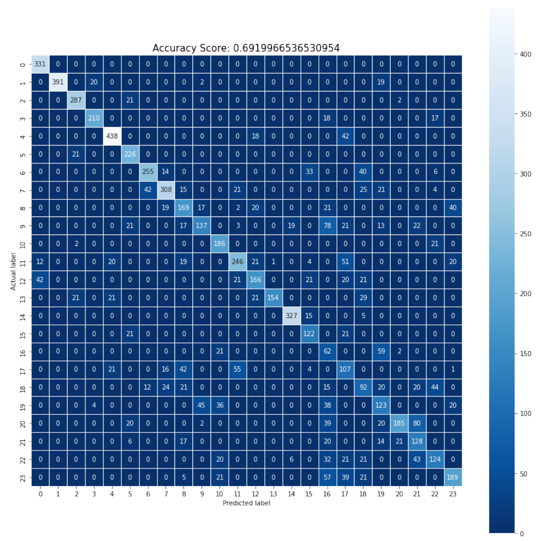


Fig. 5. Matplotlib Confusion Matrix

For example, it is possible to notice that the letter labeled 9 has common issues of being evaluated as the letter 16, which makes the accuracy for this specific letter lower.

B. One-vs-All Classification with Stochastic Gradient Descent

This algorithm follows the same scheme of the One-vs-All classification, but takes into account a learning rate which determines how often the model is updated along the way. The gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka. learning rate).

We used SGDClassifier with the following parameters:

- Loss set to ”log”, standing for Logistic Regression
- Penalty set to ”l2” - once again, the regularization is made possible with Ridge Regression
- Other parameters set to default

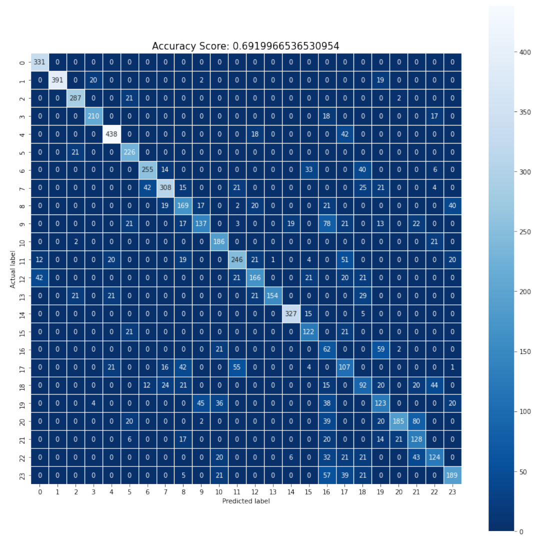


Fig. 6. Table of Data Retrieved

The accuracy with this approach is of 64%, which still falls back to the previous algorithm.

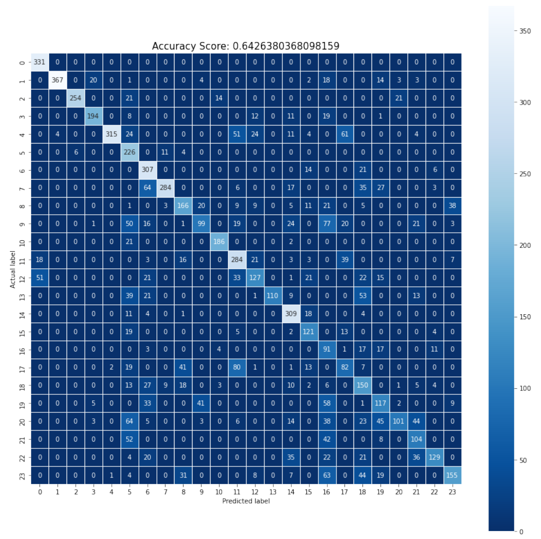


Fig. 7. Matplotlib Confusion Matrix

C. K-Nearest Neighbours

The k-nearest neighbours (KNN) algorithm is a simple supervised machine learning algorithm that can be used to solve classification problems and it does not require as much time as other classification methods to be fitted and to predict. Being a non-parametric method, it is often successful in classification situations where the decision boundary is very irregular.

The principle behind nearest neighbor methods is to find a predefined number of training samples closest in distance to the new point, and predict the label from these. The distance can, in general, be any metric measure: standard Euclidean distance is the most common choice.

The number of samples can be a user-defined constant (k-nearest neighbor learning), or vary based on the local density of points (radius-based neighbor learning). In this case, we chose k = 165, since that is the closest odd number to the squareroot of the training dataset size. For this choice, we took inspiration from multiple works, in particular another implementation of this algorithm within the same context [8].

For this algorithm, the model has obtained an overall accuracy of 60,79%.

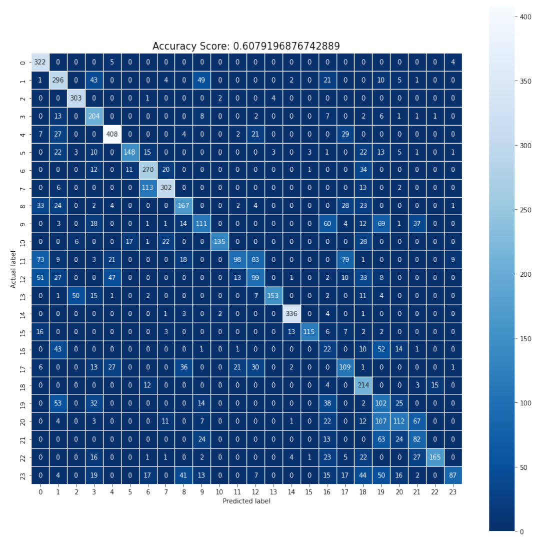


Fig. 8. Confusion Matrix of k-nearest neighbours

Despite being a very simple algorithm, it’s quite successful in a large number of classification and regression problems, as proven here.

D. Convolutional Neural Networks

CNNs are multi-layer neural networks inspired from biological processes and their structure has a notable resemblance of the visual cortex present in an animal.

They are largely applied in the domain of computer vision and has been highly successful in achieving state of the art performance on various test cases. They recognize visual patterns directly from pixel images with minimal pre-processing and, therefore, in our context of image recognition, we have found its implementation relevant.

A CNN architecture is developed by a stack of different layers that convert the input volume into an output volume through a differentiable function. A few different types of layers are commonly used, such as:

- Input layer
- Convolutional layers
- Pooling layer
- Fully connected layer

We also added dropout layers in between layers: dropout randomly switches off some neurons in the network which forces the data to find new paths. Therefore, this reduces overfitting.

In terms of design, we want out CNN to accept only four-dimensional vectors, so we had to reshape our data accordingly. We also had to reshape your labels into binary class matrices. For example, Y = 2 is transformed into Y = [0,0,1,0,0,0,0,0,0,0].

For our two attempts at implementing a CNN, we varied a couple of convolution layers and the optimizer used to compile the model. In both we used convolutional layers, a max pooling layer, a flatten layer and multiple dense layers.

We also set the value of the following variables:

- Batch size equal to 32
- Epochs equal to 20

In our first attempt, we compiled the model with a categorical cross entropy loss function, Adadelta optimizer and an accuracy metric. The Adadelta optimizer, instead of summing up all the past squared gradients from 1 to "t" time steps, it restricts the window size. For example, it computes the squared gradient of the past 10 gradients and average out, solving the radically diminishing learning rates of other optimizers.

This attempt, after fitting the training dataset to the model, scored a total accuracy of 5.86% for training data and 9.3% for testing data.

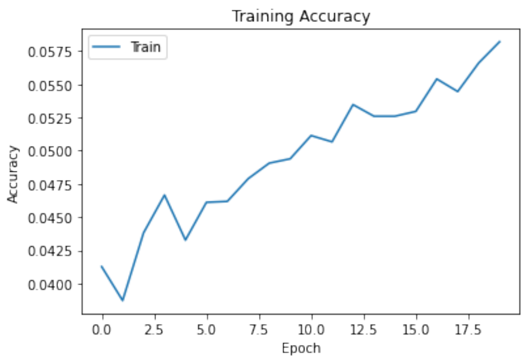


Fig. 9. First Attempt Accuracy

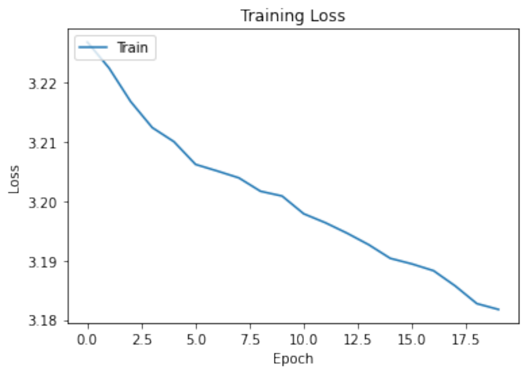


Fig. 10. First Attempt Loss

In a second attempt, with the goal of improving the performance of our CNN, we used the optimizer SGD with momentum instead of Adadelta optimizer. The problem with SGD is that while it tries to reach minima, because of the high oscillation, we can't increase the learning rate, taking time to converge. With momentum, we use Exponentially Weighted Averages to compute Gradient and used this Gradient to update parameter.

We compiled our model with the following parameters:

- Learning rate equal to 1e-3
- Learning rate decay equal to 1e-6 (to modulate how the learning rate of our optimizer changes over time)
- Momentum equal to 0.9
- Other parameters set to default

With this attempt, the model scored 99.13% accuracy for the training data and 90.55% accuracy for the testing data.

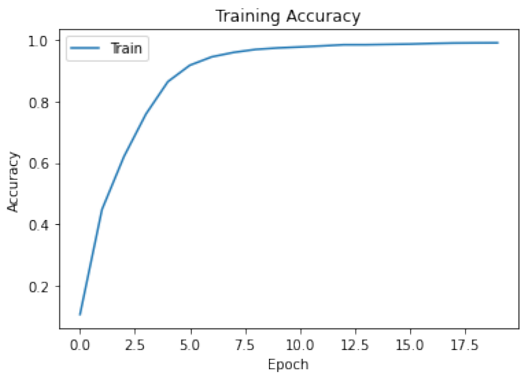


Fig. 11. Second Attempt Accuracy

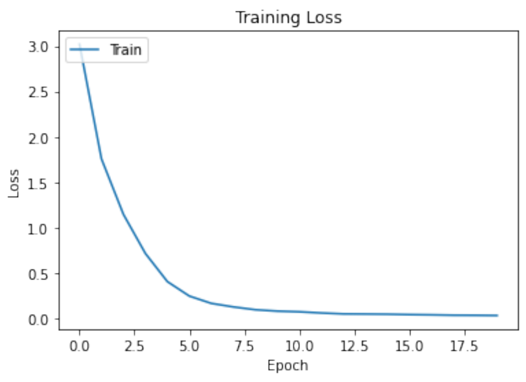


Fig. 12. Second Attempt Loss

Although the layers involved in both attempts aren't exactly the same, we have found that this huge improvement was due to the change of optimizer.

E. Support Vector Machine

Support vector machine is another simple algorithm highly preferred by many as it produces significant accuracy with less computation power. Support Vector Machine, abbreviated as SVM, can be used for both regression and classification tasks. The idea behind it is simple: The algorithm creates a line or a hyperplane which separates the data into classes.

We first tried a Linear approach with One-vs-One scheme. One-vs-One, also known as OvO, is another heuristic method for using binary classification algorithms for multi-class classification. Unlike One-vs-All that splits a multi-class classification dataset into one binary dataset for each class, the One-vs-One approach splits the dataset into one dataset for each class versus every other class.

Classically, this approach is suggested for support vector machines (SVM) and related kernel-based algorithms, as the performance of kernel methods does not scale in proportion to the size of the training dataset and using subsets of the training data may counter this effect.

Here, our algorithm has scored 78.61% accuracy with C = 1.

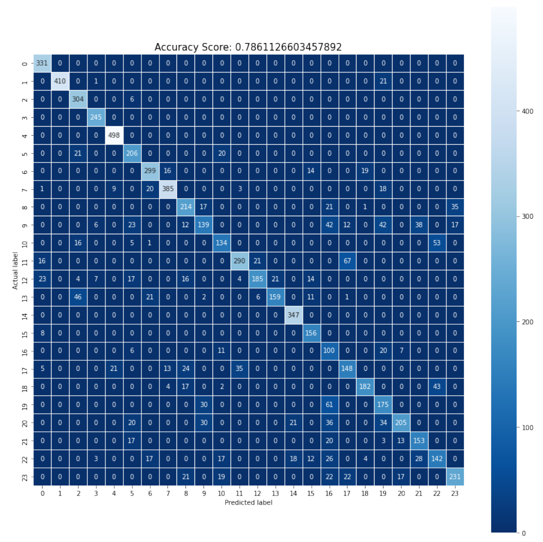


Fig. 13. Confusion Matrix of k-nearest neighbours

And it scored 78.16% accuracy with C = 10. The C parameter tells the SVM optimization how much we want to avoid miss classifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane. On the other hand, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane miss classifies more points.

In our case, a higher C slightly decreases the accuracy of the model.

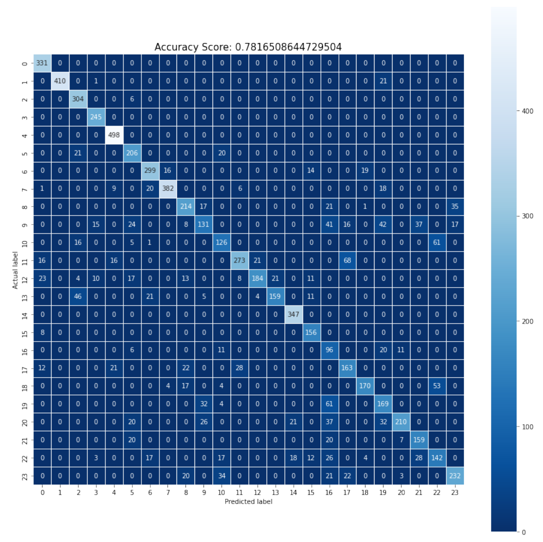


Fig. 14. Confusion Matrix of k-nearest neighbours

Finally, we implemented a SVM with One-vs-All Decision Function. We all of our training data to estimate the parameters for our SVM model and afterwards validate it with the test dataset. This method was been evaluated with an accuracy of 84.18%, revealing a better performance in comparison to the previous implementations of Support Vector Machines.

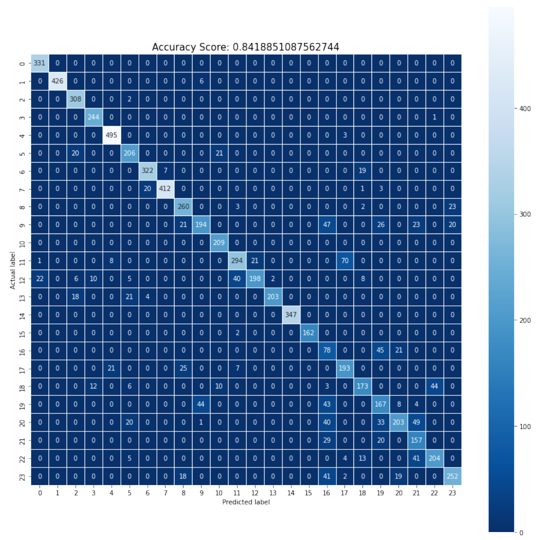


Fig. 15. Confusion Matrix of k-nearest neighbours

IV. Conclusions

In conclusion, the CNN algorithm, with the correct configuration, can achieve the best accuracy possible for this kind of problem (with image recognition).

Yet, we learned that certain strategies could be applied to improve the other algorithms, like data augmentation. Also, it should be taken into account that, depending on the amount of data in the dataset, the CNN algorithm can be compromised because of the amount of time taken to process everything.

References

[1] Sign Language MNIST DataSet By Tecperson. URL: https://www.kaggle.com/datasets/datamunge/sign-language-mnist, Sign Language MNIST, Kaggle, 2017

[2] scikit-learn 1.1.0 Documentation. URL: https://scikit-learn.org

[3] Davide Giordano, 7 tips to choose the best optimizer. URL: https://towardsdatascience.com/7-tips-to-choose-the-best-optimizer-47bb9c1219e

[4] Onel Harrison, Everything You Ever Wanted To Know About Computer Vision. URL: https://towardsdatascience.com/everything-you-ever-wanted-to-know-about-computer-vision-heres-a-look-why-it-s-so-awesome-e8a58dfb641e: :text=Computer

[5] Onel Harrison, Machine Learning Basics with the K-Nearest Neighbors Algorithm. URL: https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761

[6] Rohith Gandhi, Support Vector Machine — Introduction to Machine Learning Algorithms. URL: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[7] Towards AI Editorial Team, Convolutional Neural Networks (CNNs). URL: https://pub.towardsai.net/convolutional-neural-networks-cnns-tutorial-with-python-417c29f0403f

[8] Ledioz, SVM vs KNN — Sign Language Recognition. URL: https://www.kaggle.com/code/ledioz/svm-vs-knn-sign-language-recognition