kaggle Competition Project

Team Name : AAGTeam
Ahmed Amine Daou - Alejandra Jimenez - Gabriel Lagubeau

December 1, 2018

1 Introduction

In this document, we'll walk through a machine learning classification problem in the context of a Kaggle competition.The purpose is to design an algorithm that can automatically identify in which class a noised hand drawn image belongs among 31 classes . The dataset is divided into a training (10 k) and a testing (10 k) sets.

We started with exploring the data, processing it to make the future work easier, the next step was applying a simple baseline and other ML algorithms such as Logistic regression , CNN, SVM... and Bagging decision Trees which gave us the best result.

2 Feature Design

After loading the data from its $.npy$ (training & testing features) and $.csv$ (training labels) files into numpy arrays, we applied pre-processing methods to make sure our chosen machine learning algorithms work in a more efficient way.

Looking for missing & infinite values :

- Make sure there's no invalid or corrupt values in our datasets.
- Verify if all input values are finite to avoid this kind of $sklearn$ errors:

  `sklearn error ValueError: Input contains NaN, infinity for a value too large for dtype('float64')`

Reshaping inputs:

Reshaping training and testing images from (10000 , 2) to (10000 , 10000) . In our case, an image is a 100×100 matrix, transforming it to a vector makes things go easier.

Denoising, Centering & Resizing Images:

Training and testing sets images are extremely noised, and that will easily mislead any learning algorithm by making him learn wrong information. Our denoising algorithm conserves only the largest number of pixels connected to each other. It will wipe off Isolated connected pixels (1, 2 or 3 isolated pixels).

The next step is image "centering", by searching the Center of mass, we translate the image by x and y axis to make the images superimposed which leads to more powerful learning (always changing almost the same parameters).

Finally we perform resizing and the returned result is a 40×40 image instead of 100×100, a little image containing only relevant information and faster to process.

Data augmentation:

We also tried data augmentation by adding 1000 randomly rotated(-25°,+25°) images to our training set, It did not lead to any improvements in our case.

Data sample:

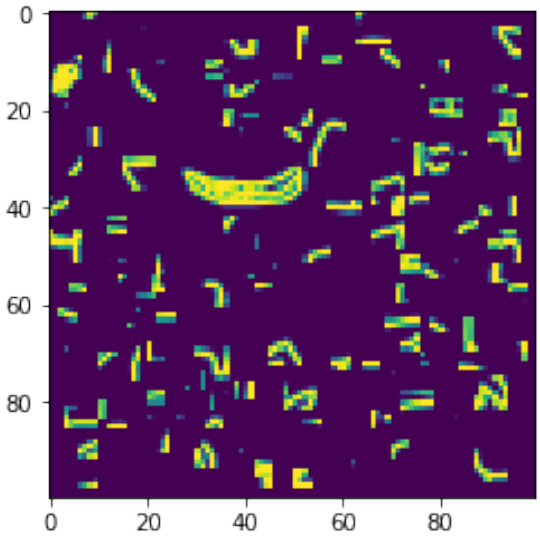

Figure 1: Original sample Image

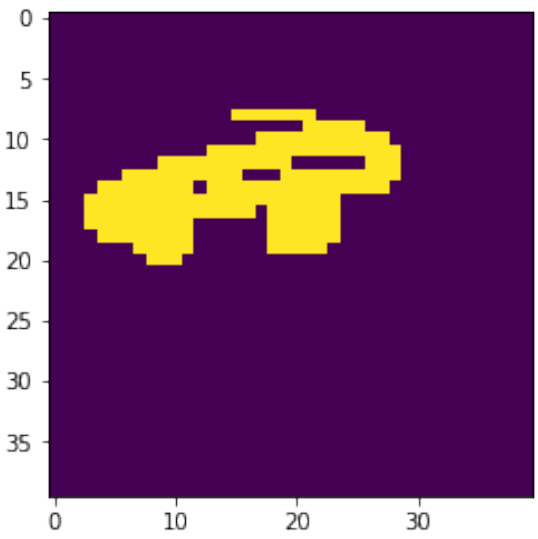

Figure 2: After Denoising, Centering & Resizing

3 Algorithms

After data processing, we used two different algorithms:

Logistic Regression

Bagging Decision Trees:

Bagging is a method that combines the prediction from a learning algorithm like Decision Trees to make a better accurate prediction. This procedure can reduce the algorithm variance.

4 Methodology

Training/Validation split:

Our goal is to create a model that generalizes well to new data. Splitting data in a good way makes the difference. Before choosing the percentage of each set, we should be sure that we meet some conditions:
- Testing set should have the same characteristics as training set and should be big enough to give meaningful results
- Less Training data begets high variance in parameters estimation - Less Testing data begets high variance in statistical performance That's why we chose to split the testing images into (80% for training,20% for testing.

Optimization Tricks:

Image denoising:s

To get better results while training the model, firstly we began by denoising all the images using $connectedComponentsWithStats$ function from $OpenCV\ cv2$ library which returns noiseless im-

ages based on connected pixels numbers in each image.

Denoised Image translation:

Second, To let all the images be in the same position in order to compare each other, we implemented $centerimg$ function, that translates the pixels (containing relevant information = perfectly denoised image) to the matrix top left. We use a library called $ndimage$ with which we can find the center of mass. With this information we can reposition the image and reduce its size to finally have only the part of the whole image that we need to compare.

Image resizing:

Third, We resized images from 100×100 to 40×40 to have less parameters to optimize during training.

Setting hyper-parameters:

Logistic Regression:

No Hyper-parameter tuning Here.

Bagging decision trees:

The hyper-parameter that we used for Bagging Decision Trees is the number of trees ($n\_estimators$). The algorithm performance is directly proportional to the trees number, but while we were increasing this variable the computing process got more expensive. We did the test with different number of trees and we used the model with the biggest accuracy. In this case, it was obtained with $n\_estimators = 600$. With this model we reached our best performance on Kaggle. The accuracy is better while we increase the hyperparameter number of trees, but this increment is significant until $n\_estimators = 600$. When we select for example $n\_estimators = 800$ the accuracy is not meaningful bigger and the results didn't give us a better accuracy on Kaggle.

5 Results

Algorithm	Accuracy
Simple baseline: Logistic Regression	22.85%
Best result: Bagging Decision Trees	55.76%

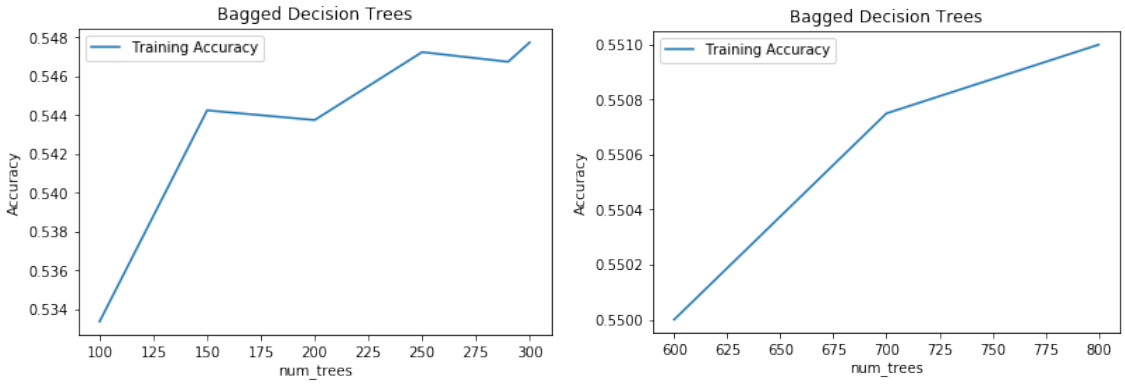

Figure 3: Trainning Accuracy evolving with tree numbers

6 Discussion

BDT vs. LR

Bagging Decision Trees performs better than Logistic Regression. After the application of the Decision Trees algorithm, we use Bagging (a meta-estimator), that reduces the variance of the Decision Tree method. It use random subsets of the dataset are drawn with replacement, that means that the same element could appear multiple times in the one sample. Bagging takes the original dataset and add their individual prediction by voting or averaging to form a final prediction. that what makes this algorithm give better result compared to Decision Trees and Logistic Regression.

Pre processing influence on classifiers performance

Denoising	**Translation**	**Image resizing** $\rightarrow (40 \times 40)$	**LR Accuracy**	**BDT Accuracy**
			4.2%	5.01%
✓			20.3%+(15.9%)	49.2%+(44.19%)
✓	✓		22.1%+(1.8%)	52.3%+(3.1%)
✓	✓	✓	22.85%+(0.75%)	55.76%+(3.46%)

Table 1: Influence of each pre processing function on Accuracy

The previous table shows that the most impactful choice that changed or algorithm performance was denoising, that made logistic regression Accuracy going from 4% to more than 20% and bagging decision trees going from 5% to almost 50%. The second most important choice was image centering, because even when the images were similar(belonging to the same class), if image position of the was different, the accuracy was lower. Another important step that leaded to our highest score is hyper parameter tuning, testing several hyper parameters and choosing the best one made our algorithm accuracy raises.

7 Statement of contributions

Team Member	**Problem Definition**	**Suggested Pre-pocessing**	**Best performance**	**Report Writing**
Alejandra	**Bagging Trees Image processing**	**Denoising & centering**	**n_trees = 600** 55.76%	✓
Ahmed	**CNN & other NN Image processing**	**Image resizing**	**CNN & other NN Algorithms** 51.2%	✓
Gabriel	**KNN-Kmeans & SVM**	**Data augmentation**	**KNN with** $k = 8$ 52%	✓

Table 2: Members' contribution on each part of the project

We hereby state that all the work presented in this report is that of the authors