kaggle Competition Project

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

November 29, 2018

Abstract

DONT FORGET README FILE TO DESCRIBE HOW TO RUN THE CODE
DOCUMENT EVERY FUNCTION,PLOTS,REMOVING OUTLIERS(it can help)

what we need to do: add data normalization(easy) + looking for missing values(easy) (won't change many things but it's important) +(maybe: removing outliers) + try to resize images to minimum possible size

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 .... , ... and Bagged 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 giving 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 to make the images superimposed which leads to more powerful learning (always changing almost the same weights).

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.**to finish**

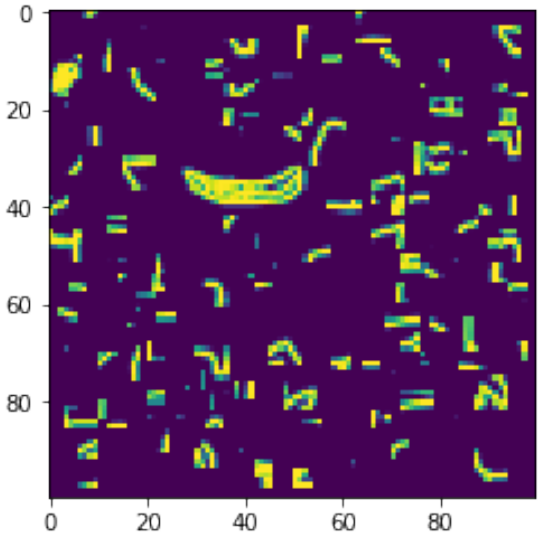

Figure 1: Original sample Image

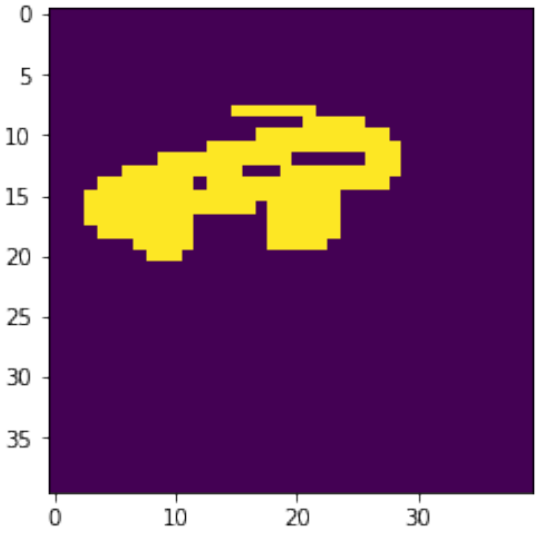

Figure 2: after Denoising, Centering & Resizing

3 Algorithms

After data processing, we used two different algorithms:

Logistic Regression:

Bagged 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 images 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 ($n\_estimators$). By default this number is 10. The algorithm performance is proportional to ($n\_estimators$) , so we need to choose the best value of this variable. We did the test with different number of trees and we used the model with the bigger accuracy. In this case the bigger accuracy obtained was with $n\_estimators = 280$. With this model we reached our best performance on Kaggle.

5 Results

Algorithm	Accuracy
Simple baseline: Logistic Regression	22.85%
Best result: Bagged Decision Trees	55.43%

need to put some plots

6 Discussion

Discuss why your best method performs better than the simple baseline.
Bagging Decision Trees performs better than Logistic Regression. After the application of the Decision Trees algorithm, we use Bagging, that is 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.

Which design choices where the most impactful on performance?

on doit ajouter le temps d'execution de l'agorithme d'entrainement
et ici on doit lancer l'algorithme sans debruitage, apres sans recize et sans translation et comparer l accuracy et le temps d'execution a chaque fois pour dire quelle operation nous a donné plus de performance (normalement c'est le denoising qui nous a donné plus de performance)
exemple de tableau

Denoising	Image resizing	Image translation	LR Accuracy	BDT Accuracy	Execution time
					sec +2 minutes
✓				52%	sec +2 minutes
✓	✓			52%	sec +2 minutes
✓	✓	✓	52%	+10%	sec +2 minutes

The first design choice that helped us getting better performance was the Image denoising, that made logistic regression Accuracy going from ..% to more than 20% . The second 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. An important step that leads to our highest score is hyper parameter tuning, testing several hyperparameters and choosing the best one made our algorithm raises After the choice of the algorithm, the decision about the hyperparameters that we were going to use affects the final performance of the algorithm. so we needed to determine the best value in order to get the best performance.

7 Statement of contributions

Team Member	**Problem definition**	**Methodology development**	**Solution Coding**	**Data analysis Performing**	**Report Writing**
Alejandra	contr	ccc	ccc	cc	cc
Ahmed	contr	ccc	ccc	cc	cc
Gabriel	contr	ccc	ccc	cc	cc

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