Universitat Politècnica de Catalunya
Facultat d'Informàtica de Barcelona
Aprenentatge Automàtic II

Labeling objects from its drawings

Javier Mestre, Gonzalo Córdova, Miquel Palet

2021/2022 Q1
10/12/21

Contents

1 Motivation, Goals and Previous Work

1.1 Motivation

It is known that machine learning is rapidly invading all areas of our lives and gaining traction across industries. A lot of companies have started to apply many technological techniques such as Machine Learning, Artificial Intelligence and Data Mining in order to predict market prices, client behavior or anything that can help improve business intelligence.

Our team has tried to take a disruptive approach when it comes to Machine Learning use cases, and tried to have fun with it, while gaining a profound understanding at the same time. We had previous experience with games that consisted in guessing what other players were drawing, and from there our curiosity and little knowledge on Machine Learning suggested studying how well the methods we were familiar with could play these games for us, therefore increasing our probability of winning.

We were also interested in finding the limits and meanings of these predictions, such as certain drawings that were always more difficult to label, or players whom drawings were never predicted because of their bad skills.

1.2 Problem and Dataset

We will be using a dataset from **Google Creative Lab**: *Quick, Draw!* Dataset.

The Quick Draw Dataset is a collection of 50 million drawings across 345 categories, contributed by players of the game *Quick, Draw!* (**Disclaimer**: this game is super addicting). This dataset has already been preprocessed and we have particularly chosen the simplified drawing files (see section [2.2] for more information on this topic).

Goal: Provide a classification model to label correctly a given drawing.

The main approach during the course of this work will be to develop, train and test a series of techniques and algorithms that, thanks to the available data, will be able to effectively and efficiently provide a good classification function. In this case, to be used for gaming purposes. Moreover, we want to demonstrate the usefulness of kernel methods in classification tasks, while comparing them to other non-kernel and non machine learning approaches.

1.3 Previous Work

This game was created by Google and is rather famous, therefore leading to a lot of work on its data. We will focus on how the original game recognizes drawings.

As exposed by the tutorial posted by Google on Recurrent Neural Networks for Drawing Classification, the game uses a RNN-based recognizer for this problem and is coded in TensorFlow. The model uses a combination of convolutional layers, LSTM layers, and a softmax output layer to classify the drawings. When training the model for 1M steps, the expected accuracy is of approximately 70% on the top-1 candidate. Also, the game does not use the top-1 candidate only but accepts a drawing as correct if the target category shows up with a score better than a fixed threshold.

2 Data Exploration

2.1 Visualization

When the Google Creative Lab built *Quick, Draw!*, they had the foresight to save anonymized copies of the drawings. At this point millions of people across the globe have played and Google has open sourced 50 million of the drawings they created. This means we have on average more than 100,000 drawings for each of the 300 words in the game to explore.

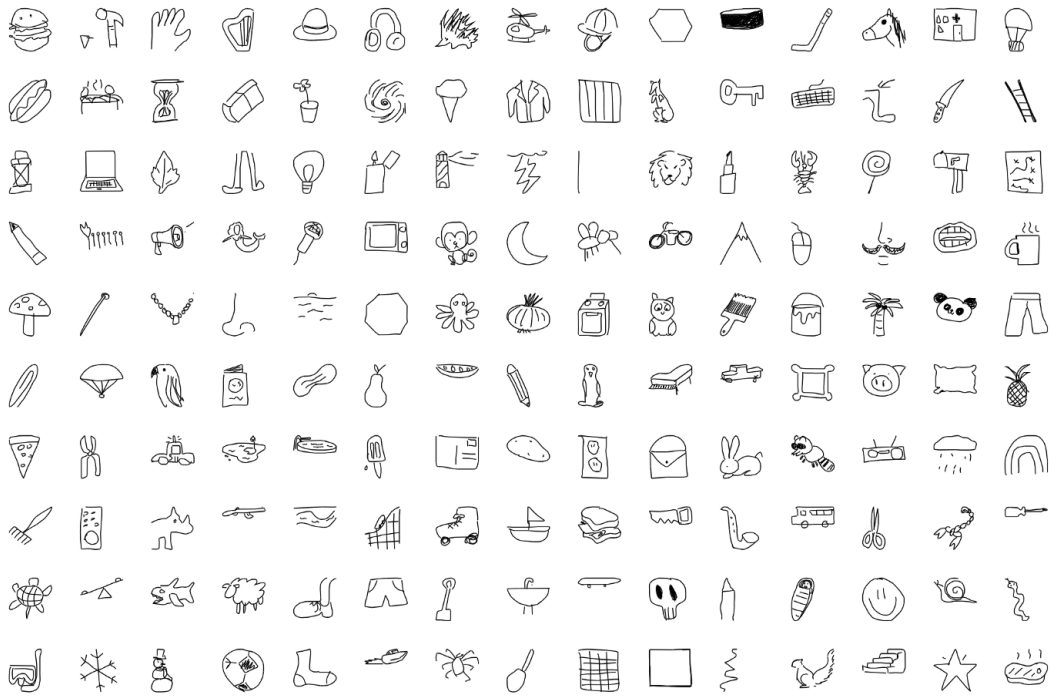


Figure 2.1: Some of the 300+ words in the dataset

Let's take a closer look at the data, what it is and what it is not.


```
word: "cat"
countrycode: "JP"
timestamp: "2017-03-15 21:41:50.79245 UTC"
recognized: true
key_id: "6429928587264000"
strokes: ▾Array(21) [
  0: ▾Array(8) [
    0: ▸Object {x: 348.75, y: 185.25}
    1: ▸Object {x: 344.25, y: 168.75}
    2: ▸Object {x: 350.25, y: 155.25}
    3: ▸Object {x: 384.75, y: 99.75000000000001}
    4: ▸Object {x: 399.75, y: 83.25000000000001}
    5: ▸Object {x: 407.25, y: 87.75000000000001}
    6: ▸Object {x: 414.75, y: 120.75000000000001}
    7: ▸Object {x: 423.75, y: 182.25}
  ]
  1: ▾Array(3) [
    0: ▸Object {x: 531.75, y: 89.25000000000001}
    1: ▸Object {x: 512.25, y: 164.25}
    2: ▸Object {x: 510.75, y: 183.75}
  ]
  2: ▾Array(6) [
    0: ▸Object {x: 540.75, y: 83.25000000000001}
    1: ▸Object {x: 546.75, y: 81.75000000000001}
    2: ▸Object {x: 554.25, y: 92.25000000000001}
    3: ▸Object {x: 561.75, y: 134.25}
    4: ▸Object {x: 566.25, y: 185.25}
    5: ▸Object {x: 561.75, y: 207.75}
  ]
  3: ▾Array(2) [
    0: ▸Object {x: 537.75, y: 174.75}
    1: ▸Object {x: 554.25, y: 176.25}
  ]
  4: ▾Array(5) [
    0: ▸Object {x: 378.75, y: 174.75}
```


Figure 2.2: How data is represented

This dataset contains categorical data, such as which word is being drawn and which country the drawing originated from. We also have a few time related dimensions, such as how long the drawing took to draw (duration), a timestamp of when the drawing was made. We also have the sequence of points that make up the drawing.
As we see in figure [2.2], the data is exceptionally rich and arises several interesting questions that could unmask interesting patterns in the data, such as the relation between how long it takes to draw and the quality of it, or the differences between the same concepts for different countries.

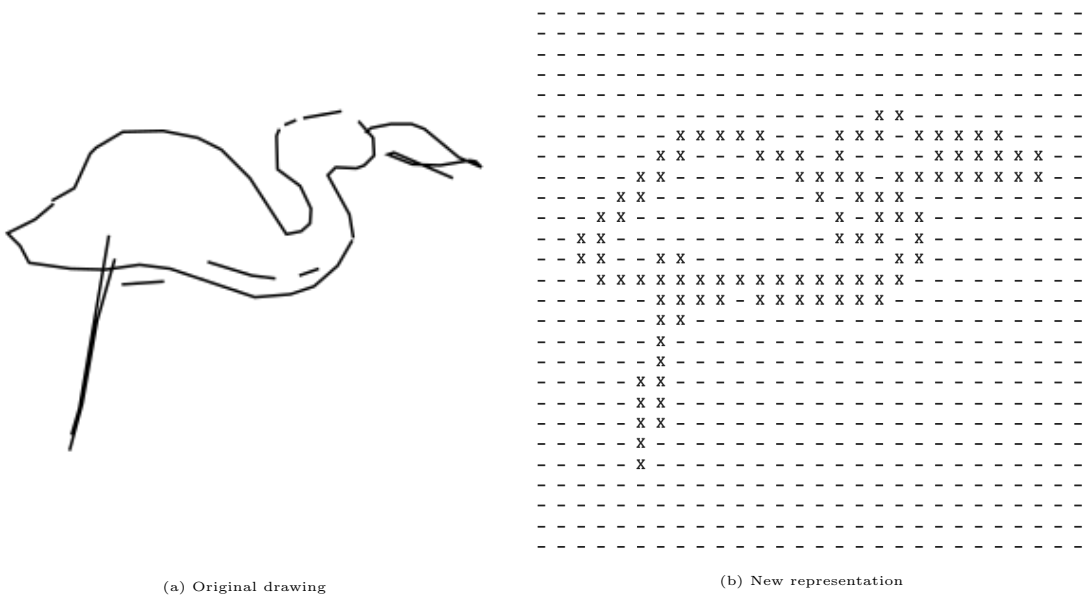
However, this is not the scope of our work, therefore we have decided to work with the simplified version of the dataset.

2.2 Transformation

After simplifying the vectors, removing the timing information, and positioning and scaling the data into a 256×256 region, the data has been exported in *.ndjson* format with the same metadata as the raw format. The simplification process was:

1. Align the drawing to the top-left corner, to have minimum values of 0.
2. Uniformly scale the drawing, to have a maximum value of 255.
3. Resample all strokes with a 1 pixel spacing.
4. Simplify all strokes using the Ramer–Douglas–Peucker algorithm with an epsilon value of 2.0.

Subsequently, we map all values to either 0 or 1 and create a 28×28 ($= 784$) array that represents the drawing.



(a) Original drawing

(b) New representation

Figure 2.3: Simplified data

We can appreciate in figure [2.3] the final representation of the data that will be used in chapter [3] for model training and testing.

Finally, we have only selected 4 categories and 10.000 drawings for each of them in order to reduce computational load. These categories are **cloud**, **sun**, **umbrella** and **eyeglasses**.

3 Modelling Phase

Since our task is a classification one, we will consider the following methods: **Support Vector Machines**, **Convolutional Neural Networks**, and **t-SNE** combined with **Random Forest** [1]. These methods have been analysed following a specific order due to the properties and shortcomings that each method presents. As mentioned in the statement of this work, we are going to compare a kernel method (SVM) to a non-kernel (CNN) and non machine learning (t-SNE) approach.

3.1 Resampling protocol

All methods split data in 75% training and 25% test. The estimation of the hyper-parameters are done under a maximization of the accuracy. In the following sub-sections, we present conclusions drawn from each method as well as the best set of hyperparameters per model.

3.2 Modelling methods

3.2.1 Support Vector Machines

Support Vector Machines (SVM for short) are a set of supervised machine learning techniques mainly used for regression and classification goals. Although the original use of the SVM [2] described by Vladimir Vapnik in 1963 concerned a linear classifier, it wasn't until the 1992 that the *kernel trick* was used to form a maximum margin classifier in nonlinear data and arbitrarily higher dimensions [3]. Thanks to the kernel map, we can induce a feature space and then solve for the separating hyper-plane to build our classifier. [1] [2]

We quickly realized that, because of our (complete) dataset being incredibly huge, our machines would have quite a hard time computing the best possible hyper-parameters for the SVMs and their kernels, so we decided to use a small subset to estimate them, specifically 1.000 drawings.

The hyper-parameters that need to be estimated in this model are C, the kernel and γ. We have used Grid Search to conduct this estimation. This method exhaustively considers all parameter combinations generating candidates from a grid of parameter values. In our case we have considered a range of three different kernels: polynomial, linear and RBF. The best score has been obtained by the polynomial kernel with $\gamma = 0,1$ and $C = 100$, with an accuracy of $0,925$, although the same score has been obtained for other combinations of C and γ. In table [3.1] we can observe the top 5 best-performing hyperparameters.

accuracy	C	γ	kernel
0,9255	100	0,1	polynomial
0,9255	1000	1	polynomial
0,9255	1000	0,1	polynomial
0,9255	1000	0,01	polynomial
0,9255	1	1	polynomial

Table 3.1: Set of best parameter combinations

After choosing the hyperparameters, we can already train and test the final model. The accuracy of the classifier on the validation set is 0,9492, and in figure [3.1] we can appreciate its confusion matrix.

[1] Code can be found inside the delivered folder, in three separate notebooks
[2] http://www.svms.org/history.html
[3] http://kernelsvm.tripod.com/

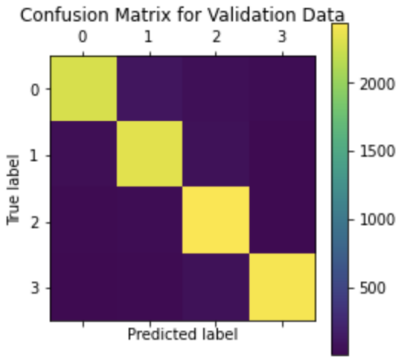


Figure 3.1: Confusion matrix of the final model

When analyzing this matrix we identify that the classifier uniformly predicts all classes without overperforming or underperforming in any case.

	cloud	sun	umbrella	eyeglasses
cloud	2291	114	58	39
sun	59	2333	60	19
umbrella	27	37	2444	11
eyeglasses	11	22	61	2424

Table 3.2: Numeric confusion matrix

If we further dive into the numeric confusion matrix to do a proper evaluation and differentiation between classes, we can observe in table [3.2] that the first to classes are the most ambiguous, and therefore the most difficult to distinguish.

We have to take into account when evaluating the accuracy of the classifier that this dataset is made from user drawings, and therefore can contain bad or even nonsense drawings. Moreover, even if the drawing is very explicit, such that a human would immediately recognize it, the classifier probably would not predict it correctly as it is not related at all to the rest of them. We can notice this curious phenomenon in the below figures.

(a) Anonymous QuickDraw artist

(b) The American flamingo, illustrated by John Audubon

3.2.2 Convolutional Neural Networks

Neural Networks are complex machine learning models inspired by a simplified take on how biological brains are supposed to work. They are made up of layers of neurons connected by weights. In comparison with other models in the scenario of classification, neural networks show better resolution in treating high dimensional input data and capturing complex relationships between explanatory variables. On the other hand, they usually require a great amount of training data, computing power and good interpretation skills. [3]

In this case we have chosen Convolutional Neural Networks (CNN for short) over Feed-Forward Neural Networks. This is due to the fact that the latter limits itself to flatten the image into a vector, as we did in the previous model, and feed it to a Multi-Level Perceptron for classification purposes. On the contrary, a CNN is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better. We have taken advantage of the knowledge obtained on the second part of this course in order to present better conclusions.

In order to define the CNN model we will use 2 fully convolutional layers, ReLU activation function and MaxPooling. This model has *kernel size* (different from the SVM kernel), *stride*, *padding* and *number of filters* as hyper-parameters. As the type of neural network that is used and its hyper-parameters cannot be easily estimated, we have based our choice on our work with the MNIST dataset. Moreover, the loss function has been the cross-entropy loss.
After this step, we can already train and test the final model. After 40 epochs and some computing time, the accuracy of the classifier on the validation set is around 80%, and in figure [3.3] we can appreciate the evolution of the loss curve and the evaluation accuracy.

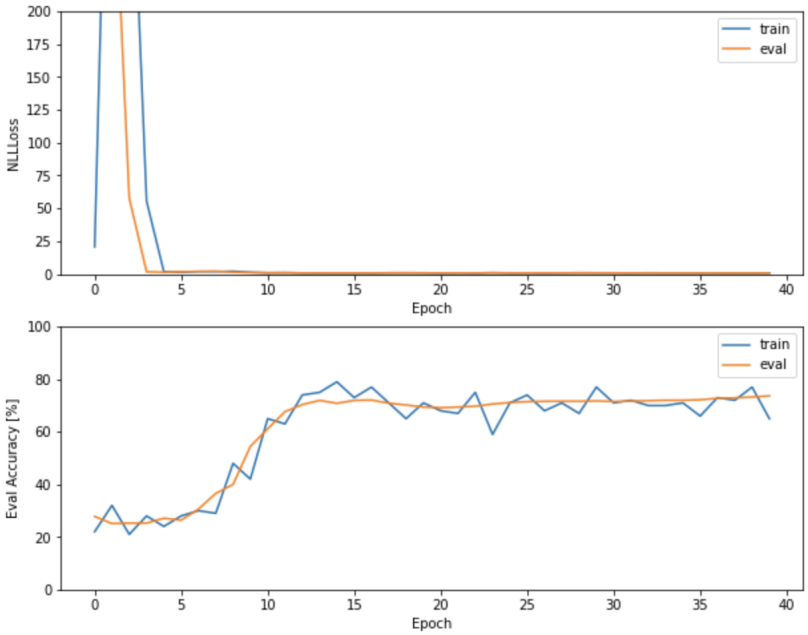


Figure 3.3: Loss curve and accuracy of the final model

When analyzing these two figures, we can easily realize that after 5 epochs the model does not actually improve. The loss function rapidly tends to 0 and the evaluation accuracy slightly climbs up to 80%.

Unexpectedly, this model predicts significantly worse than the kernel SVM that we previously trained. This could be due to the fact that the neural network that we used was not optimal, but we did not dare doing any changes to the architecture of the network as we do consider ourselves beginners. This result will be further discussed in chapter [4].

3.2.3 t-SNE

In order to present a non machine learning method, we are going to present t-SNE. t-SNE is short for t-Distributed Stochastic Neighbor Embedding, and is a statistical method for visualizing high-dimensional data by giving each datapoint a location in a two or three-dimensional map. It is a dimensionality reduction method, jsut as Principal Component Analysis (PCA), however PCA is not able to capture non-linear dependencies, due to the fact that is based on linear projections.

This technique finds clusters in data thereby making sure that an embedding preserves the meaning in the data. t-SNE reduces dimensionality while trying to keep similar instances close and dissimilar instances apart. As it is a visualization technique, we are not going to be able to compute metrics to evaluate its classification performance, but it will allow us to grasp how non machine learning approaches

The two most important hyper-parameters of these technique are *perplexity* and *step size*. The former is understood as the number of points whose distances will preserve in low dimension space, and the latter is the number of iterations of the algorithm. In both cases, we have used default values, as this was a mere exercise.

We can appreciate how t-SNE performs in figure [3.4], using a dataset with only 10.000 drawings and reducing the dimensionality to 2 components.

Figure 3.4: Output of the t-SNE method

As similar digits are clustered together, we can use a clustering algorithm on this output and easily assign labels to unseen test data.

Using a random forest approach with 6 estimators, the accuracy has been 0,8972.

Taking into account the simplicity of this model and its short computing time in comparison to the previous ones, the predicting accuracy is astounding, overperforming by 10 points the CNN. Hence, this has given us an insight on how powerful dimensionality reduction is.

4 Experiment Results

4.1 Performance metrics

Classification accuracy is a metric that summarizes the performance of a classification model as the number of correct predictions divided by the total number of predictions.

It is easy to calculate and intuitive to understand, making it the most common metric used for evaluating classifier models. Hence it is the one that has been considered in this work.

4.2 Discussion of the results

We can showcase the obtained results by each model in table [4.1]. In our case, the best model has been the SVM, followed by the t-SNE with random forest.

	SVM	CNN	t-SNE
accuracy	**0,949**	0,792	0,897

Table 4.1: Model performance

From a theoretical point of view, based on our knowledge and experience on the subject, we expected the CNN model to outperform the other methods, because it is what usually happens if (1) there is really a lot of data and (2) its structure is complex. However, in this case the data did not have many layers, therefore we could not exploit its full potential.

The random forest classifier, in contrast, doesn't need a lot of examples per class. So, when testing the classifiers with few examples, the latter yielded the best results.

SVMs do require substantially more training data than random forest, but less than neural networks (to get a decent performance). Moreover, training algorithms for SVMs have better guarantees than CNN due to the optimization problem being complex.

4.3 Scientific and personal conclusions

As machine learning students, we tend to go to SVMs naturally because we understand them better than neural networks, not even to mention CNNs. If we encounter a problem during the process of modelling with a SVM, we have the sufficient understanding of the topic to tackle it, however when working with neural networks, we do not feel confident enough. Nevertheless, at the speed at which the research is moving in deep neural networks and as we become machine learning experts, these gaps will be soon filled.

About working with statistical models, we find them very intuitive and easy to grasp. When working with the t-SNE we had a lot of fun, being able to play with different visualizations and then applying different classification algorithms. We think that every ML beginner is attracted to it because of how simple and intuitive it is. However, with large amount of data, it can be expensive to employ due to its overhead.

Finally, we thoroughly believe that the CNN model could have performed much better if implementing another architecture. However, as aforementioned, the complexity of this task was far beyond our knowledge and the scope of this work.

4.4 Possible extensions and limitations

During our research on how these classifiers could be improved, we came across an interesting approach on implementing a hybrid CNN-SVM [1] classifier for the MNIST dataset, in which the model combines the key properties of both the classifiers, implementing CNN as an automatic feature extractor and SVM as a binary classifier. The results of this study demonstrate the effectiveness of the proposed framework by achieving a recognition accuracy of 99,28% over MNIST handwritten digits dataset. We believe this would be fascinating to pursue in the near future over our dataset.

In addition, during our work we found some limitations regarding computational power. Our machines were, in some cases, not powerful enough to perform certain calculations. Most of the times, those regarding hyperparameter tuning and model training. Still, we managed to use Google Colaboratory's computing resources (including GPUs), therefore easing our duties.

Moreover, knowledge regarding Convolutional Neural Networks limited our comparison between models, and prevented us from reaching the desired performance our model as well.

4.5 Showcasing results

To exhibit the classifier in action, in figure [4.1] we can appreciate how the model predicts drawings correctly.

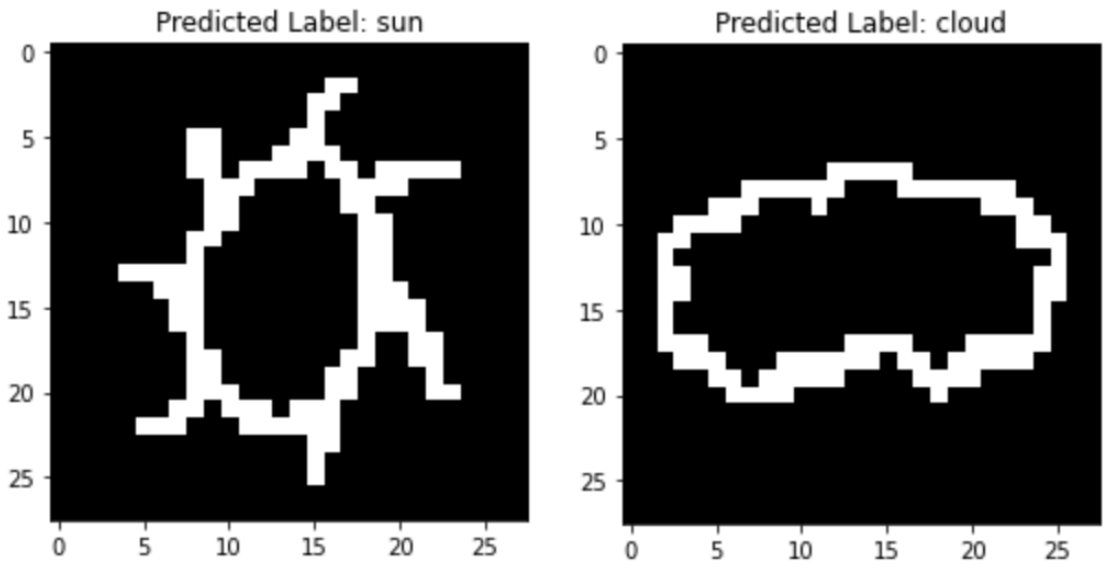


Figure 4.1: Example results

[1] https://www.sciencedirect.com/science/article/pii/S1877050920307754

Bibliography

[1] Alexander J. Smola Bernhard Schölkopf. *Learning with Kernels.* MIT Press, 2002.

[2] Christopher M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

[3] Brian D. Ripley. *Pattern Recognition and Neural Networks.* Cambridge University, 1996.