A TUTORIAL ON OBJECT RECOGNITION BY MACHINE LEARNING TECHNIQUES USING PYTHON

J.R. Ruiz-Sarmiento, J. Monroy, F.A. Moreno, J. Gonzalez-Jimenez

Machine Perception and Intelligent Robotics Group, System Engineering and Auto. Dept., Instituto de Investigación Biomédica de Málaga (IBIMA), University of Málaga (SPAIN)

Abstract

This paper presents our experiences towards a tutorial on a hot *Computer Vision* problem, object recognition employing *Machine Learning* techniques, which we consider that can be a resource didactically interesting for both practitioners' and lecturers' communities. The recognition of objects is an innate ability of human beings. When we look at a picture, we are able to effortlessly detect elements like animals, signals, objects of interest, etc. In the Computer Vision field, this process is carried out by Machine Learning tools, aiming to retrieve information about the content of an image. The possible applications of object recognition are numerous, to name a few: image panoramas, robot localization, face detection/recognition, autonomous driving, or pedestrian detection.

Given the relevance of this problem, numerous approaches have been explored to address it. Among the most popular techniques in both, real applications and academia, we can find those recognizing objects by means of hand-crafted features. Typical features are descriptors of their *keypoints* extracted according to a certain criterion (e.g. *Scale-Invariant Feature Transform*, SIFT), or those geometrically (size, orientation, shape, etc.) or visually (colour, texture, etc.) describing the regions to be recognized. The number of software tools developed to carry out this task is large, including well-known software libraries as *OpenCV* or *scikit-learn*. However, object recognition is far from being a one-step task, and most of them lack a comprehensive step-by-step description of the pipeline needed for accomplishing it. This pipeline usually includes the management and analysis of the data used to train the recognition model, a pre-processing of such data to convert them into an usable form, the fitting of the model and, finally, its validation.

The tutorial presented in this work encompasses those steps, giving detailed information and hints about how to complete each one. It is accompanied by a public implementation of the object recognition pipeline (https://github.com/jotaraul/object_recognition_in_python) using trendy python tools (*Pandas*, *seaborn* and *scikit-learn*), so it can be profitable by Computer Vision practitioners and lecturers aiming to gain insight/illustrate good practices for the design of a successful object recognition system. It is also open to any contribution from those communities. Finally, we also provide some directions and experiences regarding its utilization in academia.

1 INTRODUCTION

The purpose of a visual object recognition system is to identify the elements in a scene (represented as an image) as belonging to certain categories (*e.g.* tv, table, couch, chair, bowl, etc.). Humans innately accomplish this task, but its algorithmic definition to be implemented in a computer is far from being trivial [1]. This problem is framed within the *Computer Vision* field, being *Machine Learning* (ML) techniques widely adopted for developing these systems [2]. The aim of ML based methods is to fit a number of parameters according to certain training data in order to successfully recognize new objects [3][4]. Given their popularity, these methods are widely studied in courses from universities worldwide and in online platforms like Coursera or Udacity.

There exist numerous approaches and techniques for modeling the objects to be recognized, i.e., to build representations of them that are useful to recognize new objects. Among them, we could highlight these: employing descriptions of certain keypoints [5] (like HOG [6], SIFT [7] or SURF [8]), describing their geometry (size, orientation, shape, *CAD* models) or visual appearance (color, texture), or using prior information about them (semantic knowledge) [9]. It is worth mentioning that, nowadays, the recognition techniques based on Deep Learning are reaching a great success (*e.g.* [10] or [11]). However, these models need machines with great computational capacity as well as huge amounts of training data for their fitting, which are resources that might not be available depending on the problem domain/application, so ML techniques continue to be a demanded option.

ISBN: 978-84-09-08619-1

Before their usage for recognizing objects (working phase), ML-based systems need to go through a design phase. Figure 1 illustrates the needed steps towards such design, including: an analysis of the employed dataset, its split into training and testing data, a data preprocessing step where the objects are characterized through a number of features, and finally the model fitting and evaluation/validation. Given the popularity of these techniques, there are numerous frameworks that allow us to manipulate and evaluate them [3][12][13][14]. Unfortunately, it is not that common to find well-explained solutions to accomplish the aforementioned steps towards the design of a successful model.

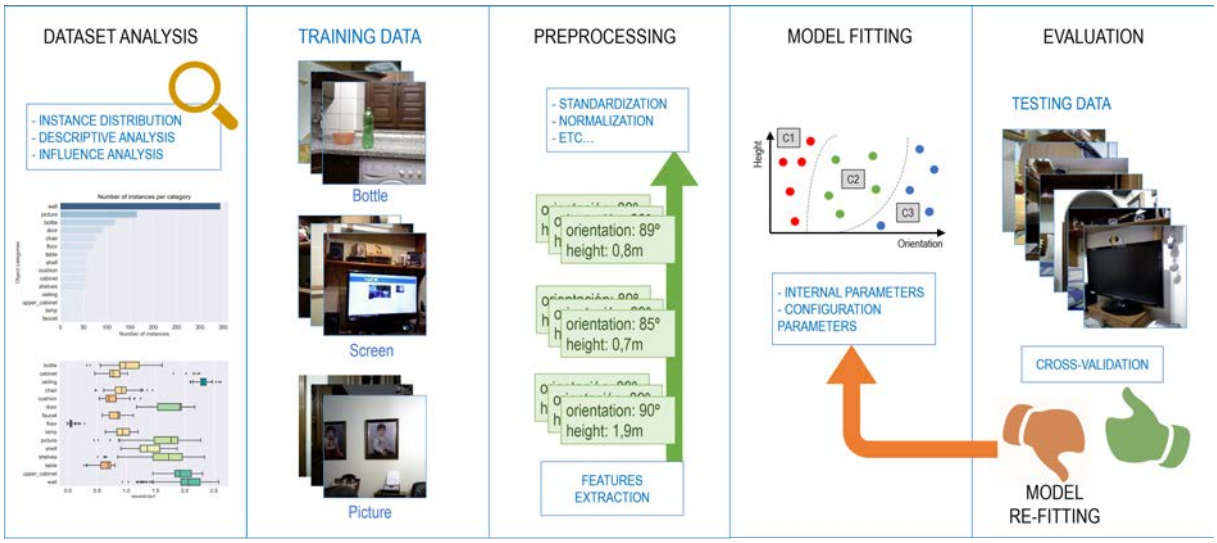

Figure 1. Typical steps in the design phase of a ML-based object recognition system.

This work provides a number of directions to successfully accomplish the design phase of a ML-based object recognition method, putting the spotlight on techniques that use features for modeling objects (geometry, appearance, etc.). It comes along with a number of Python scripts as well as a Jupyter Notebook for experimenting/practicing the hints and analyses contained herein, available at https://github.com/jotaraul/object_recognition_in_python. Concretely, the provided scripts make use of the following libraries: *Pandas* for loading and processing the dataset [15], *Seaborn* for graphical representations [16], and *Scikit-learn* for fitting recognition methods (*e.g.* decision trees, support vector machines, logistic regression, naïve Bayes classifier, etc.) [13].

We rely on the *Robot@Home* dataset [17] to exemplify the tasks carried out in each step of the methods' design phase. This repository contains RGB-D images providing both intensity and depth information of the scenes, gathered by a mobile robot in different houses. Robot@Home yields labels annotating objects in such images with their belonging categories, also including a description of the appearing objects using geometric and visual features [18].

The rest of the article provides, in Sec. 2, an introduction to the resorted Python libraries. Sec. 3 describes good practices for managing and analysing the chosen dataset (Robot@Home or any other), while Sec. 4 discusses how to characterize objects and prepare them for being used in the fitting of the recognition method. Sec. 5 describes how to fit, evaluate and validate methods using different techniques and metrics, Sec. 6 provides some directions and experiences regarding its utilization in academia, while Sec. 7 concludes the paper with a brief discussion.

2 PYTHON LIBRARIES

Pandas. The aim of this library is to facilitate the manipulation of datasets and to provide the needed components for implementing statistical models. It focuses on data structures and tools for working with datasets, permitting us to load/save data from/to memory or files with different formats (CSV, text files, Microsoft Excel, SQL databases, HDF5), or to restructure, pivot, or merge datasets, among other features. In order to be fast and efficient, its critical parts are implemented in Cython or C. Pandas also provides linear regression statistical models, but, according to its developers, their goal is to tightly work with those of Scikit-learn to join efforts and supply comprehensive models to the community (last page in [15]).

Seaborn. This library, written in Python and based on the popular *matplotlib*, offers a high-level interface for producing visually appealing statistical graphics [16]. One of its most relevant features is that it includes support for Pandas data structures, so both tools can be used in the same code without an explicit data type conversion step. Some of the provided functionalities are: different themes and color palettes for preparing eye-catching charts, functions for visualizing and comparing distributions of one or two variables, tools for fitting and showing linear regression models, or functions for visualizing data matrices. All the charts in this paper have been generated by Seaborn.

Scikit-learn. Perhaps the most popular Python ML library [13]. It encompasses a large variety of algorithms for both supervised (where data come with the attributes to be predicted, *e.g.* the objects' categories) and unsupervised problems (data without such attributes). Its purpose is to provide non-experts in the field with such algorithms through a high-level and general purpose programming language: Python. For that, they pay special attention to its ease of use, high performance, good documentation, and APIs consistency. Among other functionalities, it allows us to fit prediction models, make them persistent in disk, and evaluate them by different techniques and metrics.

3 DATASET MANAGEMENT AND ANALYSIS

The first step in the design of a recognition system is to study the baseline dataset (see Figure 1). At this point, the dataset must contain instances of objects belonging to the categories to be recognized. Such instances usually come in the form of objects' observations (sensorial information), and/or features extracted from them.

The analysis of the dataset is a key step that permits us to better understand the information therein, design a strategy to face the problem (enhancing the possibility for success), and detect limitations regarding the conclusions that we can extract when evaluating the method [19]. Concretely, its purpose is to acquire information concerning: the distribution of the instances of each object category (balance), the behavior of the features used to describe them, and their discriminant capacity. The completion of this analysis can save time, since it can detect errors that would lead to wrong method fittings. At this point it is convenient to employ the Pandas library, particularly suitable for the loading of large datasets and the extraction of this basic information in an efficient way.

3.1 Instances distribution

The purpose of this analysis is to determine the balance between the number of instances and object categories in the dataset. For the Robot@Home case, the figures reported by the analysis are 2309 object instances belonging to 180 categories, although this number is misleading if it is used for object recognition, since they appear categories with a low number of instances (or even just one). This is the case, for example, of the categories *fax*, *radiator*, *yogurt*, or *yoga ball*. Moreover, as we can conclude from the study, and as it is shown in Figure 2, the dataset is unbalanced. The balance of the employed dataset has to be taken into account during the learning of the method parameters since, depending on the chosen one, its performance may be affected. In such an event, the dataset could be balanced to avoid that issue, as shown in the Python scripts.

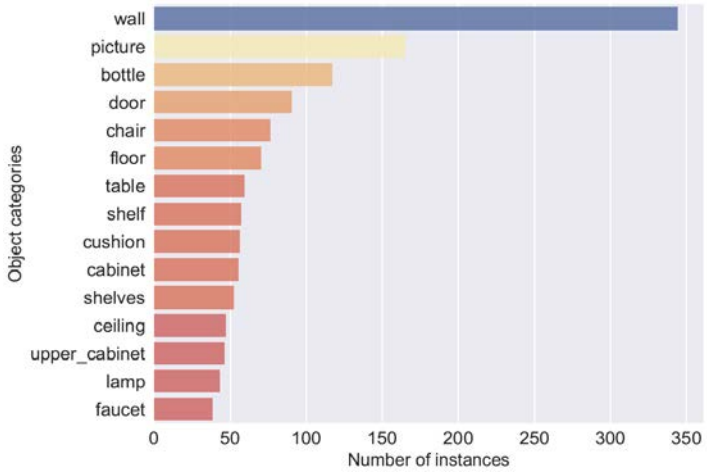

Figure 2. Number of instances of the most appearing categories in Robot@Home.

3.2 Descriptive analysis

This analysis pursuits the description of the different features used to categorize objects, permitting us to understand the central tendency, dispersion and shape of their distributions, initially validate their utilization, and detect misleading behaviors (typically coming from a wrong implementation). Table 1 reports the basic description of a number of features from Robot@Home (computed by Pandas). The kind of conclusions that can be extracted are, for example, that the minimum height (height of the object part closest to the floor) is lower on average (logically) than the maximum, that there are more vertical than horizontal objects (given the orientation mean value), or that the distribution of the objects' volume is positively skewed with a predominance of small objects (according to the percentiles). We can also see that, as reported by the lowest and highest hue values, the pure red doesn't appear in the data (it has a hue value of 0). This type of information allows us to check if the values of the given variables/features make sense, as well as to gain insight into the observed objects.

Table 1. Basic description of central tendency, dispersion and shape for the distributions of 6 features from the Robot@Home dataset.

	Min. Height	Max. Height	Orientation	Volume	Planarity	Hue
Mean	0,54*m.*	1,44*m.*	66,15º	0,80m^3.	0,21	117,37
Std. Deviation	0,59*m.*	0,66*m.*	32,33º	1,57m^3.	0,12	54,50
Lowest val.	-0.50*m.*	-0,01*m.*	0,01º	0,00m^3.	0,00	24,65
25%	0.00*m.*	0,87*m.*	57,21º	0,03m^3.	0,11	75,20
50%	0.41*m.*	1,54*m.*	84,76º	0,23m^3.	0,19	108,95
75%	0.95*m.*	2,00*m.*	88,36º	0,83m^3.	0,30	154,26
Highest val.	2.37*m.*	2,61*m.*	89,99º	13,16m^3.	0,49	299,10

Another interesting analysis can be carried out through the visualization of the distribution of such features for each object category in the dataset. For example, Figure 3 shows, using tools from Seaborn, the distribution of the values of the features *height of the centroid* (left) and *planarity* (right). In those charts, the colored boxes represent the range that comprises the 50% of the data, while their lower and upper limits represent the 25 and 75 percentiles, respectively. The inner, vertical lines show their mean values, while the outer vertical lines indicate values up to 1.5 times away of the distance between the 25 and 75 percentiles. Finally, values further away from these lines are shown as points, and are considered outliers. Intuitively, for a feature being discriminant among categories it must take values in a small range (low dispersion), and such range must be different for each category. We can observe that in the planarity case the dispersion is significant, and the taken values are quite similar for the different categories. Conversely, in the case of the centroid height such dispersion is lower, and the difference between categories is higher. This points out that, a priori, the latter feature is more discriminative and would help to recognize objects to a larger extent.

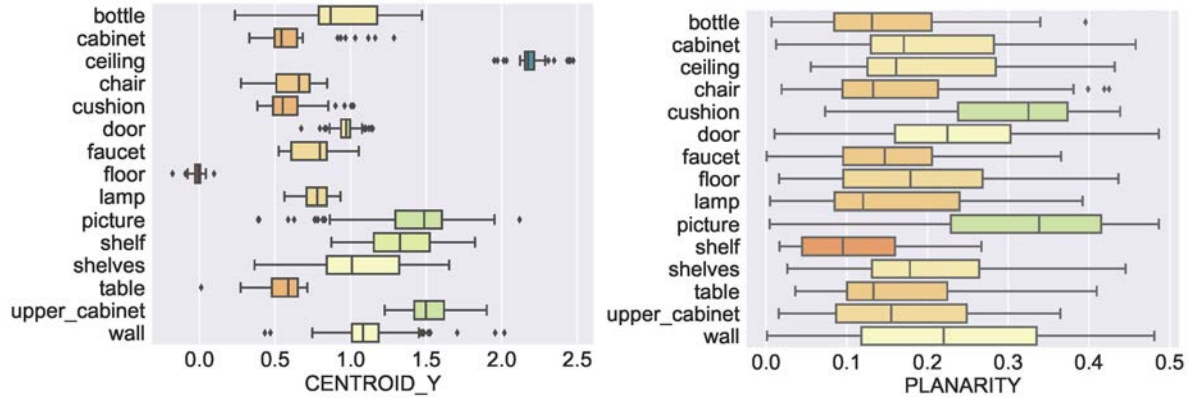

Figure 3. Distribution of the values taken by the planarity and centroid_y (height of the centroid from the floor) features according to the objects' categories.

3.3 Influence analysis

This analysis carries out a statistical estimation of the discriminatory power of features with respect to the object categories, that is, how promising they are for discerning the different categories. This study is performed in an individual way, being the recognition method the one in charge of properly combining them. At his point, a simple but effective method is to undertake a chi-square test (χ^2) for each feature. This test makes the hypothesis that there is no relation between a feature and the object categories, builds a model considering that such a relation does not exists, and returns a measurement of the error obtained comparing the model outcome with the expected one in the dataset. The test execution also reports a *p-value*, which is used to statistically validate the result if it takes low values (typically lower than 0,05). In this way, for a feature having discriminatory power the chi-square test has to return a high value along with a low p-value. It is worth mentioning that in order to carry out this test the features have to be binarized, *i.e.* they can only take two different values. It can be performed with features taking more values, but this requires an extra checking or a post-hoc step, so in this work we will focus on the simplest version.

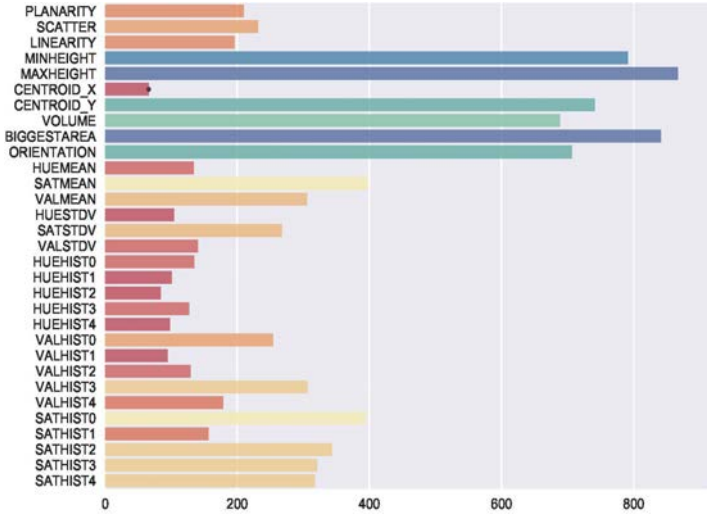

Figure 4. Results of the chi-square test for the features in Robot@Home.

Figure 4 reports the results of this test for all the features in Robot@Home. We can see how there are 6 features that obtain a measurement higher than the rest (in blue/green), strongly rejecting the hypothesis that there is no relation between them and the object categories. There are other features (in orange) also showing a high value for such statistical, while the remaining ones (in red) are less promising. In every case their associated p-values are considerably lower than 0,05.

4 PREPROCESSING

In the previous section we have conducted a preliminary analysis of the discriminatory power of the features describing objects in Robot@Home. This has been done directly since such a dataset provides, among other information, characterizations of the observed objects employing the features listed in Figure 4. If the employed dataset does not provide that information, or the analysis of the features concludes that they are unsuitable, the dataset has to be complemented with a new set of discriminant features. In that way, as shown in the evaluation section, the set of features within Robot@Home is quite comprehensive, since they describe the objects' size, shape, spatial position and appearance. As previously mentioned, objects can be described using features of different nature, including HOG, SIFT, SURF, or even Haar features. In some of these cases the previous analyses are not applicable, while others must be adapted for producing valid results.

In any case, ML learning methods often benefit from a features preprocessing step in order to standardize their values, that is, ensure that they follow a Gaussian distribution with zero mean and unitary standard deviation. This is due to the way in which they are fitted, where if a feature has a higher variance than other, it could dominate the model fitting and compromise their suitability. Scikit-learn includes different standardization techniques that also permit us to work with disperse data. This library also provides, in the preprocessing module, tools to perform non-linear transformations of the

features, normalize the data (scale each description so it has unitary norm), binarize them, or even generate polynomial features to add complexity to the model in order to integrate more complex data.

5 METHOD FITTING AND EVALUATION

Once the objects' descriptions have been retrieved and standardized, they are ready to fit the model (recall Figure 1). In this step, the method internal parameters are fitted according to the available training data in such a way that, given a new object description, it is able to successfully infer its category.

Metrics. Once the method has been fitted, it is validated by evaluating its performance using the *testing data*. This checking is done in accordance with a certain metric, or a set of them, being considered as a valid method if it reaches the required success for the particular application where it is going to operate. The most used metrics are *accuracy* (ratio between the number of objects correctly recognized and the number of them in the dataset), *precision* (which expresses the method ability to not label an object as belonging to a wrong category), and *recall* (which measures the ability to correctly categorize an object as belonging to a certain category) [20].

These metrics can be computed at the instance (object) level (including in that case the *–micro* prefix), or at the level of object category (including *–macro*). There exist metrics that combine the previous ones, like the *F1-score*, a weighted average of precision and recall. The metric to choose will depend on the requirements to be fulfilled by the final system.

Cross-validation. The split of the available data into *training* and *testing data* is not trivial and, in cases where it is not properly addressed, the validity of the conclusions reached when evaluating the method may be compromised. A commonly adopted approach to face this is the utilization of *cross-validation* techniques. These techniques aim to validate the model through a statistical analysis in charge of: supporting the obtained results, giving a certain level of generality to them, and reducing the possibility of obtaining a good result just because how the data were divided into training and testing sets. In a nutshell, it consists of performing an iteration where the dataset is divided into two groups, using one of them for fitting and the other one for evaluation. To reduce the variability of the obtained result, this iteration is repeated using different groups, combining the validation results in order to estimate the method performance.

Numerous cross-validation techniques can be found in the literature, differing i) in the way in which they divide the dataset and ii) in the number of iterations to be completed. In the cases where it is unfeasible to check all the possible dataset splits, one of the most resorted techniques is *k-folds*, where in each iteration the dataset is randomly divided into *k* groups or folds of the same size, using one of them to evaluate the method and the remaining ones to fit it. The method success is obtained after performing a high number of iterations (the higher the number, the more confident the reached conclusions) and averaging the yielded results.

Methods. Scikit-learn provides a variety of classification methods that can be used for object recognition. Perhaps the most popular are: decision trees, random forests, extra-trees, support vector machines, logistic regression, k-neighbors, or the naïve Bayes classifier. The purpose of this work is not to thoroughly evaluate each method, but we are going to use them to illustrate some points. For example, Figure 5-left illustrates how the *accuracy* metric varies according to the number of cross-validation iterations completed (figures obtained using the 15 object categories with more instances in the dataset and all the available features after standardizing them). That is, the vertical axis represents the averaged values for this metric given the iterations completed in the horizontal axis. We can see how, with a low number of iterations, the results for a given method considerably oscillate, hence being unreliable. A figure of this type is useful for retrieving the required number of iterations for the metric being stable and informative. In this case, we can check that after 100 iterations the values are stable for most methods.

Each method, in addition to the internal parameters to be fitted during training, also includes a number of configuration parameters that directly affects their performance, and that have to be fitted according to the application peculiarities. For example, decision tress could have a *max_depth* parameter setting the maximum depth of the tree. Cross-validation can be also used for fitting those parameters, incorporating the configuration parameters to the variables *number of iterations* and *number of groups*.

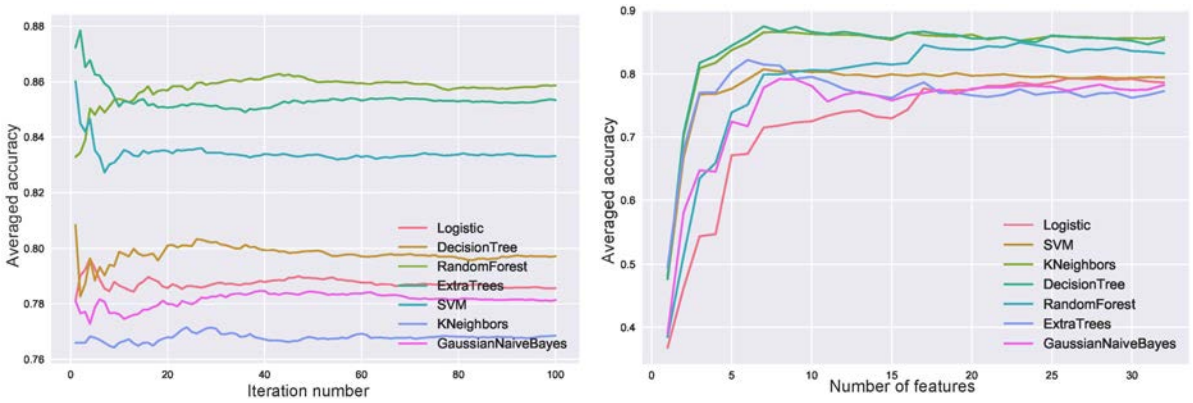

Figure 5. Left, accuracy of different methods while recognizing objects according to the number of cross-validation iterations carried out. Right, accuracy on the different methods depending on the number of features used to describe the objects.

Confusion matrix. A visual way to check different performance aspects of a method is through the so-called confusion matrix. In this matrix, rows index expected object categories, while columns index the categories to which the objects have been assigned by the method. In this way, the higher the concentration of assignments in the matrix diagonal, the more successful the method. This matrix also permits us to retrieve the objects categories showing a low performance, so the method design phase can be adapted to solve it.

Figure 6 shows the confusion matrix obtained using random forests, built accumulating the results reported in 200 iterations of cross-validation (to improve visibility, they are only shown the 10 categories with more instances). We can check how the general behavior of the method is promising (values close to 1 in the diagonal), and that the most conflictive category is cabinet. This category is often confused with chair and wall, hence to improve these results it would be needed to add a feature able to better discriminate between these categories.

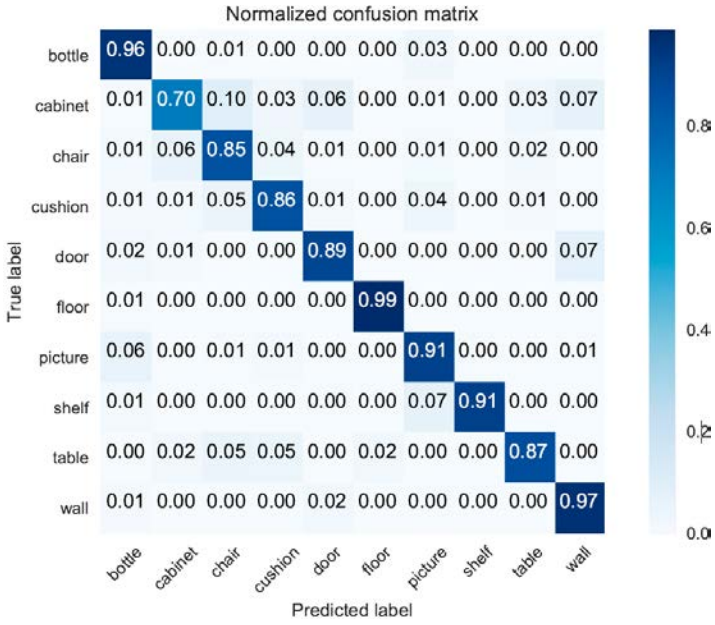

Figure 6. Confusion matrix for the 10 object categories with more instances (using random forests).

Features. In Sec. 3 we analyzed the discriminatory power of the features within Robot@Home. The utilization of features with a low power has a different influence in recognition methods: some of them are able to isolate those features and give a lower weight to them, not affecting their performance, but others are seriously affected with their usage. Figure 5-right reports the accuracy evolution of the methods used in this work according to the number of features used (executing again 200 iterations of cross-validation). Such a number always refers to the most promising features as shown in Figure 4. Another factor to take into account is the time needed for fitting these models, which can be critical

depending on the application. For example, while the methods' fitting using 6 features takes 32 seconds, the one using the 32 increases to 72 seconds (measurements taken with an Intel(R) Core(TM) i7-3820 CPU @ 3.60GHz microprocessor and a 16GB RAM memory).

6 UTILIZATION IN ACADEMIA

Both the paper content and the provided Python code can become a handy resource in academia, including Computer Vision or more general Machine Learning courses. Concretely, the Python scripts are highly configurable through the config.py module, which permits lecturers and students to practice and explore the different steps in the design of an object recognition method. Moreover, the included Jupyter Notebook is a plus in that way, since its code can be modified and executed interactively with the only requirement of having a browser (see Figure 7). We are integrating this tool in undergraduate courses of the University of Málaga with promising results.

Figure 7. Snippets of the provided Jupyter Notebook running on a browser.

7 DISCUSSION

This work has provided a number of directions for the design of successful object recognition methods based on *Machine Learning*. The text comes along with a number of Python scripts as well as an interactive Jupyter Notebook, available at https://github.com/jotaraul/object_recognition_in_python, which permits the user to easily put into practice each of the described steps in the design phase of such methods. Using tools from the Python libraries Pandas, Seaborn and Scikit-learn, the dataset used to train the methods can be analyzed, divided into training and testing data, the objects therein can be described, and such descriptions can be used to fit, evaluate and validate the methods. In this way, this paper serves as a guiding thread for the design of successful methods.

We believe that this work can be a handy resource for lecturers, as well as for Machine Learning practitioners looking for designing their own recognizer.

ACKNOWLEDGEMENTS

Work partially funded by the WISER project ([DPI2014-55826-R]), financed by the Spanish Ministry of Economy, Industry and Competitiveness, and by a postdoc contract from the I-PPIT-UMA program, financed by the University of Málaga.

REFERENCES

[1] M. Oliveira, L. Seabra Lopes, G. H. Lim, S. H. Kasaei, A. D. Sappa, and A. M. Tomé. “Concurrent learning of visual codebooks and object categories in open-ended domains,” *In International Conference on Intelligent Robots and Systems (IROS)*, pages 2488–2495, 2015.

[2] C. M. Bishop. “Pattern Recognition and Machine Learning” *In Information Science and Statistics. Springer-Verlag New York, Inc.*, NJ, USA, 2006.

[3] J.R. Ruiz-Sarmiento, C. Galindo, and J. Gonzalez-Jimenez. "A survey on learning approaches for undirected graphical models. application to scene object recognition." *International Journal of Approximate Reasoning* 83: 434-451, 2017.

[4] J.R. Ruiz-Sarmiento, C. Galindo, J. Gonzalez-Jimenez, “UPGMpp: a Software Library for Contextual Object Recognition,” *In 3rd. Workshop on Recognition and Action for Scene Understanding (REACTS)*, 2015.

[5] J. Zhang, M. Marszalek, S. Lazebnik, and C. Schmid. “Local features and kernels for classification of texture and object categories: A comprehensive study,” *In 2006 Conference on Computer Vision and Pattern Recognition Workshop (CVPRW’06)*, pages 13–13, 2006.

[6] N. Dalal, and B. Triggs. "Histograms of oriented gradients for human detection." *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Vol. 1. IEEE, 2005.

[7] D. G. Lowe. “Distinctive image features from scale-invariant keypoints,” *Int. Journal of Computer Vision*, 60(2):91–110, 2004.

[8] J. Knopp, M. Prasad, G. Willems, R. Timofte, and L. Van Gool. ”Hough transform and 3D SURF for robust three dimensional classification.” In *European Conference on Computer Vision* (pp. 589-602). Springer, Berlin, Heidelberg, 2010.

[9] J.R. Ruiz-Sarmiento, C. Galindo, J. Monroy, F.A. Moreno, J. Gonzalez-Jimenez, “Ontology-based conditional random fields for object recognition”, *In Int. Journal of Knowledge-Based Systems*, 2019, https://doi.org/10.1016/j.knosys.2019.01.005.

[10] K. He, G. Gkioxari, P. Dollar, and R. B. Girshick. “Mask r-cnn,” *In 2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017.

[11] J. Redmon, and A. Farhadi. "Yolov3: An incremental improvement." *arXiv preprint arXiv:1804.02767,* 2018.

[12] The OpenCV library: https://opencv.org/ [Accessed online Jan’2019].

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel et al. "Scikit-learn: Machine learning in Python." *In Journal of machine learning research* 12, no.: 2825-2830, 2011.

[14] J.R. Ruiz-Sarmiento, F.A. Moreno, J. Monroy, and J. Gonzalez-Jimenez. “mVision, a Toolbox for Computer Vision Courses,” *In The 12th annual International Technology, Education and Development Conference (INTED2018)*, 2018.

[15] W. McKinney. "Data structures for statistical computing in python." Proceedings of the 9th Python in Science Conference. Vol. 445. 2010.

[16] seaborn: statistical data visualization. https://seaborn.pydata.org/ [Accessed Jan’2018].

[17] J. R. Ruiz-Sarmiento, C. Galindo, and J. Gonzalez-Jimenez. “Robot@home, a robotic dataset for semantic mapping of home environments,” *The International Journal of Robotics Research*, 36(2):131–141, 2017.

[18] J. R. Ruiz-Sarmiento, Cipriano Galindo, and Javier Gonzalez-Jimenez. “OLT: A Toolkit for Object Labeling Applied to Robotic RGB-D Datasets,” *In European Conf. on Mobile Robots*, 2015.

[19] R. W. Hoerl, R. D. Snee, and R. D. De Veaux. "Applying statistical thinking to ‘Big Data’problems." Wiley Interdisciplinary Reviews: Computational Statistics 6.4: 222-232, 2014.

[20] J.R. Ruiz-Sarmiento, C. Galindo, and J. Gonzalez-Jimenez. "Building multiversal semantic maps for mobile robot operation," *In Knowledge-Based Systems,* 119: 257-272, 2017.