Implementing LIME framework to Explain Image Classification

Andrea Gigli

Introduction

A common problem of many state-of-the-art algorithms for machine learning is that their outputs and the underlying working principles are hardly comprehensible to human users. The lack of understanding of machine learning predictions constitutes a serious limitation in complex tasks, and several studies are trying to provide methodologies to explain the results of machine learning predictors. To 'explain' the prediction of a model means to provide a set of clearly interpretable features of the input that motivate the decisions of the model. A proper understating of the overall behavior of a prediction model is crucial for choosing the most appropriate model in a pool of possible ones, designing and tuning the model's parameters, forecasting the performance of the model in unobserved conditions. The explanation is helpful also in assessing the reliability of the model. This is particularly relevant in sensitive decision-making tasks like, for example, medical diagnosis, machine-learning-based robotic control or terrorism detection.

LIME: Local Interpretable Model-agnostic Explanations

In 2016, Ribeiro et al. [1] proposed LIME, a framework aimed at producing humanly interpretable explanations for the outcomes of a general black-box predictor. LIME's basic working principle is to define an interpretable representation of the input instance, i.e. a set of humanly interpretable features, and to tell which of these interpretable features best motivate the model's prediction. To do so, LIME approximates the black-box prediction model with a linear model that uses interpretable features and then determines which of those features better allow the approximated model to replicate the outcome of the original one.

In the following I will denote as $f(\cdot)$ the black-box prediction model, $\mathbf{x}$ the non-interpretable representation of the input sample, $f(\mathbf{x})$ the prediction to be explained, $\mathbf{x}'$ the interpretable representation of the input and $g(\cdot)$ the approximated model. Formally, LIME explains the prediction $f(\mathbf{x})$ by approximating $f(\cdot)$ in the proximity of $\mathbf{x}$ with a linear model $g(\cdot)$ that takes as input an interpretable representation $\mathbf{x}'$ of the original instance. The explanation of $f(\mathbf{x})$ will thus consist of the interpretable features x'_i that are most relevant to the approximated model.

An interpretable representation of an instance consists of a set of features that a human can easily understand in relation to the task of interest. A bidirectional relationship must exist between the non-interpretable and the interpretable representation of the same instance. In the context of image classification, an interpretable representation of the image can be the set of its superpixels (patches), while an uninterpretable representation is a set of color gradients, edges and curves. A human user will better evaluate the outcome of an image classifier if it is explained by relevant image patches rather than a set of color gradients.

The approximated model $g(\cdot)$ is required to be interpretable (i.e. to use an interpretable representation of the input), to be simple enough for humans to understand it, and to approximate $f(\cdot)$ locally. Therefore, the approximated model is chosen to be linear and is trained

on an artificial dataset of interpretable neighbors of $\mathbf{x}$ with the relative labels. The neighbors $\{\mathbf{z}'_1, \mathbf{z}'_2, \cdots, \mathbf{z}'_N\}$ are obtained by randomly altering of $\mathbf{x}'$. The label associated to each neighbor $\mathbf{z}'_i$ corresponds to the confidence with which the original predictor classifies $f(\mathbf{z}_i) = f(\mathbf{x})$.

The dataset, as it is, can be used to determine which interpretable features of the input better explain the classification $f(\mathbf{x})$. This can be done by using, for example, an embedded feature selection method. This class of methods picks the most explanatory features of a dataset by fitting a linear regularized prediction model on that dataset and then observing the parameters of such model. Regularization biases the model toward lower complexity, thus a restricted number of parameters. Moreover, in linear prediction models, such parameters (a.k.a. weights) correspond one-to-one to the input features and give an indication of their overall contribution to the classification of a new instance. The higher weights correspond to the most explanatory features, the lower weights to the most misleading ones. Due to this characteristic of the approximating model and the definition of the neighbors' label, the interpretable features of $\mathbf{x}'$ corresponding to the top positive weights will provide an explanation for $f(\mathbf{x})$.

LIME implementation for CNN-based image classification

In this work, I implemented the LIME procedure to explain the outcome of an object recognition task. Object recognition consists in classifying the object present in the input image. Image classifiers generally make use of complex Convolutional Neural Networks, which base the classification on hardly comprehensible visual features like color gradients and curves.

To interpret CNN predictions, the authors propose, first, to use super-pixelation to obtain an interpretable representation of the input image, then to fit a linear model on a neighborhood of the interpretable image, thus to use the weights of such approximated model as an indication of each superpixel's relevance.

In this implementation, each neighbor $\mathbf{z}$ of the original instance $\mathbf{x}$ is an image obtained by blacking-out a random subset of the superpixels of $\mathbf{x}$. The interpretable version $\mathbf{z}'$ of each neighbor is a binary vector indicating which superpixels have been blacked-out and which have been maintained unaltered.

It is assumed that the closer a neighbor is to the original image the higher is its relevance in the local approximation of the original model. A measure of such relevance π_i is obtained by computing the cosine distance between each $\mathbf{z}'$ and $\mathbf{x}'$, and then processing this quantity with a Gaussian kernel function:

$$\pi_i(\mathbf{x}', \mathbf{z}'_i) = \exp(d_{cos}(\mathbf{x}', \mathbf{z}'_i)^2/\sigma^2) \tag{1}$$

where σ is the bandwidth of the exponential kernel.

The label l_i associated to $\mathbf{z}'_i$ is chosen to be the confidence P with which the CNN classifies a neighbor $\mathbf{z}_i$ as the original image:

$$l_i = P(f(\mathbf{z}_i) = f(\mathbf{x})) \tag{2}$$

The neighbors dataset is used to fit an interpretable linear model $g(\cdot)$ by solving the equation:

$$\min_g \sum_i^N \pi_i (f(\mathbf{z}_i) - g(\mathbf{z}'_i))^2 + \alpha \|g\|_1 \tag{3}$$

where $g(\mathbf{x}) = \mathbf{w}^T \cdot \mathbf{z}'$, with $\mathbf{w}$ vector of scalar weights. The L1-norm regularization in this formulation, usually referred to as LASSO, determines the sparsity of the weights $\mathbf{w}$, implicitly realizing feature selection while fitting the model. Each weight of $g(\cdot)$ is associated to a specific

superpixel of the interpretable representation of the image. Due to the definition of the neighbor labels and the linearity of the approximated model, the top positive weights are associated to the superpixels that best explain the classification of the input image $f(\mathbf{x})$; the top negative weights correspond to the most misleading areas of the image with regard to the given classification.

Finally, the explanation is created by highlighting in green the patches of the original image that best explain the prediction (corresponding to the top positive weights) and in red the misleading ones (top negative weights).

The overall procedure is summarized in algorithm 1 and illustrated in Figure 1.

Algorithm 1 Explaining CNN predictions with LIME

Require: CNN classifier $f(\cdot)$, input image $\mathbf{x}$, interpretable representation of the image $\mathbf{x}'_i$ (boolean vector of active superpixels), number of neighbors N

NeighborDatabase $\leftarrow \{\}$
for $i \in 1, 2, \cdots, N$ **do**
    Generate neighbor $\mathbf{z}'_i$ by randomly altering $\mathbf{x}'$
    Compute the relevance $\pi_i(\mathbf{x}', \mathbf{z}'_i)$ with (1)
    Obtain the RGB representation $\mathbf{z}_i$ of the neighbor
    Compute the label l_i with (2)
    NeighborDatabase $\leftarrow \{\mathbf{z}'_i, \pi_i, l_i\}$
Fit the model $g() = \mathbf{w}^T \cdot \mathbf{z}'$ on the *NeighborDatabase* with equation (3)
Explanation of $f(\mathbf{x}) \leftarrow \{\mathbf{x}'_j\}_{j=1}^3$ corresponding to the 3 top-positive $\mathbf{w}_j$
Adversarial explanation of $f(\mathbf{x}) \leftarrow \{\mathbf{x}'_j\}_{j=1}^3$ corresponding to the 3 top-negative $\mathbf{w}_j$
Plot the explanation on the original image

Notes about the implementation

The implementation is done in *Python* language. Image classification was realized by means of a VGG-16 CNN pre-trained on ImageNet. Such architecture is available in *Keras*, a well known Python framework for Deep Learning. Image segmentation was made with the quickshift algorithm provided in the *scikit-image* Python module. The LASSO procedure for feature selection made use of the Python module *scikit-learn*.

The program will automatically explain the top-n predictions for the input image, where the number n is provided by the user. The execution can be launched with the following command:

```
python explaincnnprediction.py
        --pathimage [..]
        --ntoppred [..]
        --nneighbors [..]
        --save [..]
```

where *pathimage* is the path to the image to be classified; *ntoppred* is the number n of top-predictions to be explained; *nneighbors* is the number of neighbors of the original image needed to fit the approximated model; *save* is the output path. An example call is:

```
python explaincnnprediction.py
        --pathimage ./data/orangetap.jpg
        --ntoppred 3
        --nneighbors 1000
        --save ./results/orangetap_1000.png
```

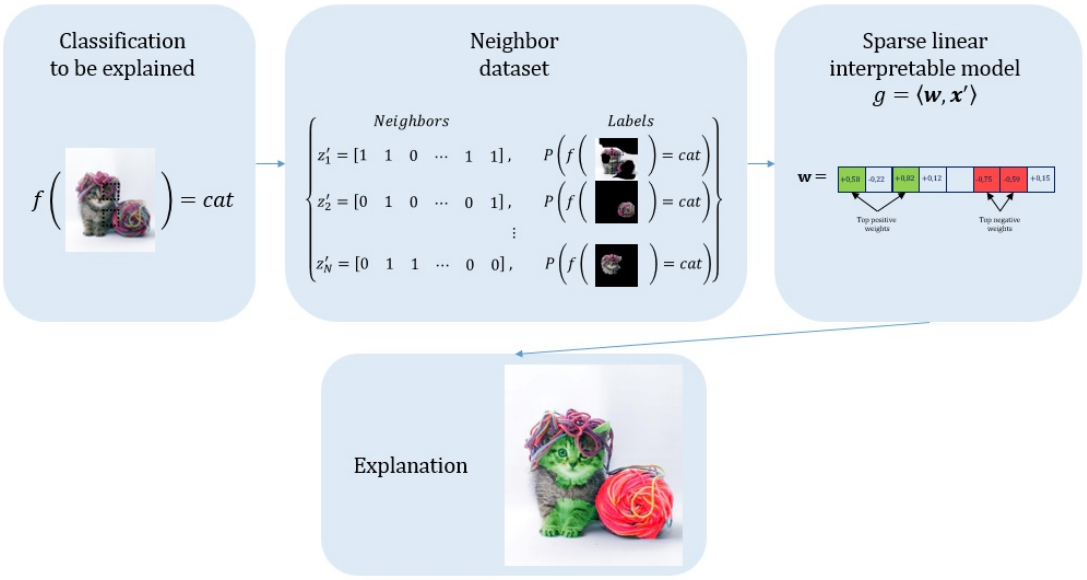

Figure 1: Graphical representation of the LIME procedure for explaining the prediction of a general image classifier.

Results

The implementation I proposed has been compared to that provided by the paper's authors[1]. For the sake of clarity, I will refer to these two implementations respectively as 'personal' and 'original'.

Both the implementations were tested qualitatively by explaining the top-3 CNN predictions for a set of equivocal images. Such images were chosen with the purpose of pushing the object classifier to provide conceptually distinct top-predictions for each image. In the tests, 1000 neighbors of the input images have been used to fit the approximated model. The two implementations were also compared by observing the differences in the explanations produced for the same set of images (and the same CNN).

Figure 2a shows the explanation provided by the 'personal' implementation for the top-3 CNN predictions of an image containing a rotary dial-phone where the handset is replaced by a lobster. The predictions 'American_lobster' and 'Crayfish' are explained by the carapace and the claws of the animal, and contrasted by the rotary dialer. The explanation for 'dial_telephone' is perfectly complementary to the other two. Figure 2b reports the explanations for the same top-3 predictions provided by the 'original' implementation. The results are affine to those of the 'personal' implementation and clearly interpretable. Noticeably, the 'original' implementation does not provide adversarial explanations for these predictions, but this is probably due to the choice of free parameters like the α of approximating model and the bandwidth of the Gaussian kernel.

Figure 3 reports the explanations provided by the two implementations for the same top-3

[1]The implementation is publicly available at `github.com/marcotcr/lime`. In order to compare it with my implementation, I slightly modified the authors' one to use a VGG-16 CNN instead of the default Google Inception V3 CNN.

predictions of an image containing an orange merged with a tap. Both the implementations perfectly explain the 'Orange' and 'Lemon' labels with the orange circle, while the 'whistle' label is explained by the presence of a metallic cylinder and a semicircular shape (resembling a resonating chamber). The 'personal' implementation shows that the tap is misleading for the fruit prediction, while the only negative explanation reported by the 'original' implementation for 'lemon' is a background patch, which is apparently nonsensical.

Figure 4 shows the explanations provided by the 'personal' implementation for an image of a toothed-wallet. The presence of a mask is suggested by teeth surrounded by the irregular shape of the wallet's edge (maybe resembling the mouth-hole of a mask), while the glowing metallic edge and the leather of the wallet explain the presence of a buckle. Analogously, Figure 5 displays the explanations for the top-3 CNN predictions for an image containing a goose head merged with a banana. The explanation is totally consistent, being 'bald_eagle' and 'goose' supported by the animal's head, beak and flipper and contradicted by the banana fruit.

References

[1] M. T. Ribeiro, S. Singh, and C. Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144. ACM, 2016.

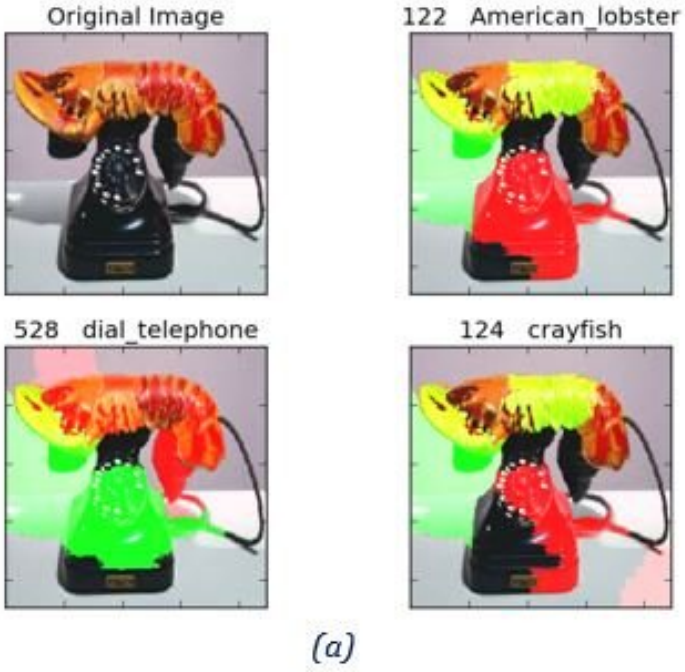

Figure 2: Comparison of the outputs of the 'personal' and the 'original' LIME implementations for the same input image (telephone and lobster).

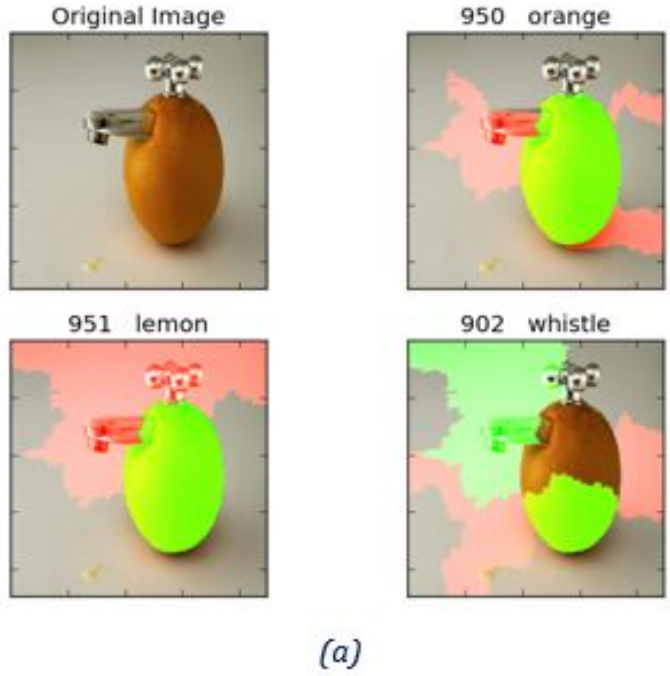

Figure 3: Comparison of the outputs of the 'personal' and the 'original' LIME implementations for the same input image (orange and tap).

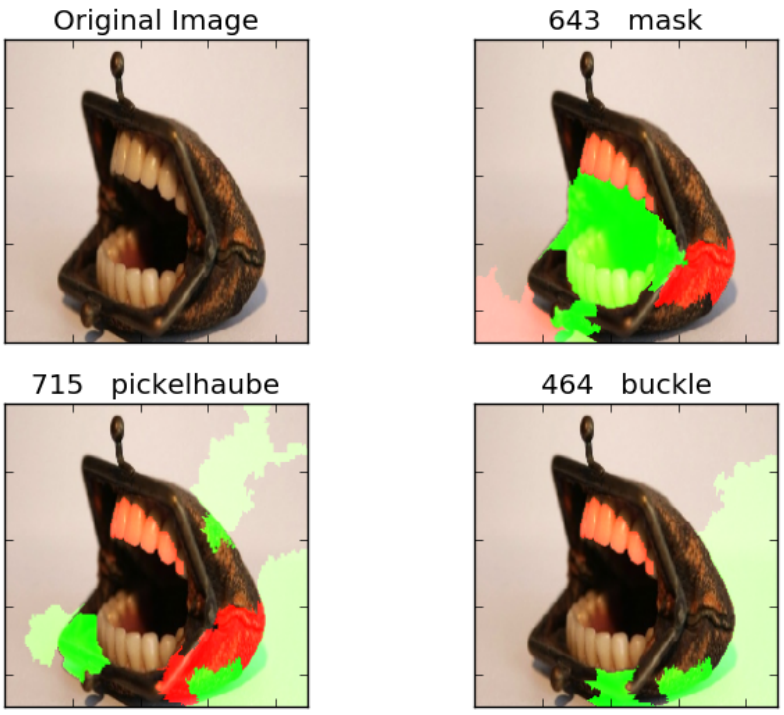

Figure 4: Explanations provided by the 'personal' implementation of LIME for the top-3 CNN predictions of an image containing a toothed wallet.

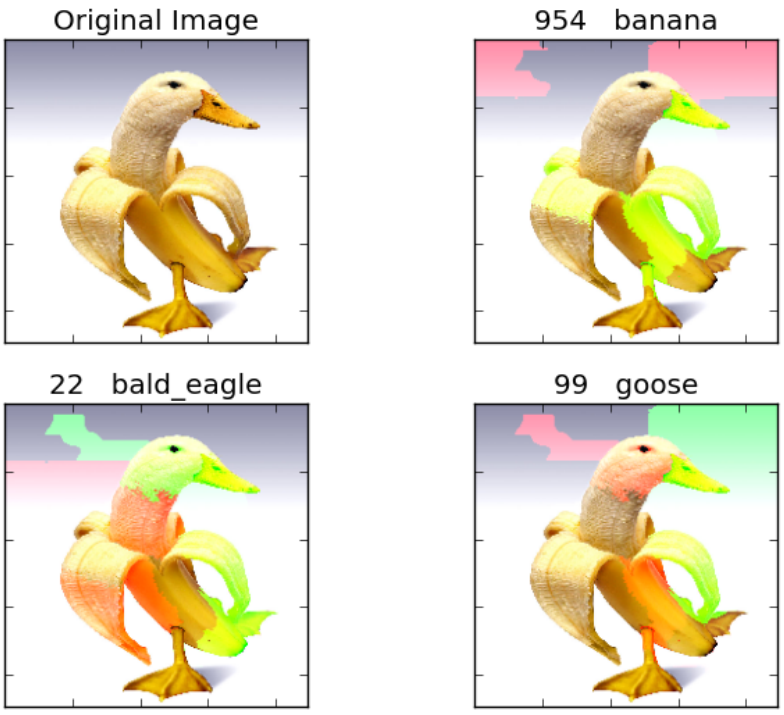

Figure 5: Explanations provided by the 'personal' implementation of LIME for the top-3 CNN predictions of an image containing a banana merged with a goose-head.