Classifying objects with Machine Learning techniques

Digital Image Processing project
Slideshow version

Falcone Federico - 86048 Garda Massimiliano - 85965

University of Brescia
Prof. Nicola Adami

Academic Year 2014/2015

Contents

Contents 1

Introduction

Google Image $\rightarrow$ Classifying images by keywords found on Internet

Our Project $\rightarrow$ Classifying objects using a dataset of images grouped by category.

Introduction

Key Ideas

How the system should work...

- Download images from Google
- Subdividing images into categories
- Store images in a database.

...what it is necessary to implement

- Features extraction
- Machine Learning Techniques
- Taking probabilities into account

Contents 2

Features

Interesting questions before starting:

- **Global** vs **Local** features: are they so different?
- How to **describe** an image to compare it with another one?

Features

Global features

Global features describe the entire image in terms of:

- Shape
- Texture
- Color

Attention: Global features are sensitive to *clutter* and *occlusion*

→Example: Color Histogram Method

Features

Local features

A **local feature** is an image pattern which differs from its immediate neighborhood.

It is usually associated with a *change* of an image property or several properties simultaneously, although it is not necessarily localized exactly on this change.

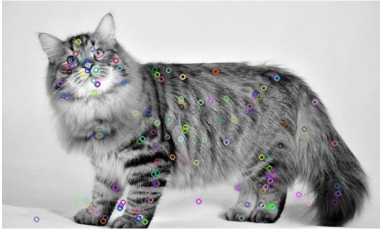

Figure 1: An example of keypoints of an image (MSER-SIFT)

Features detectors and descriptors

Base concepts

Features detectors

The first step of the feature extraction process is to find a set of distinctive **keypoints** that can be reliably localized under varying imaging conditions, viewpoint changes, and in the presence of noise.

Features descriptors

Once a set of interest regions has been extracted from an image, their content needs to be encoded in a descriptor that is suitable for discriminative matching.

Features detectors and descriptors

Base concepts

Features detectors

The first step of the feature extraction process is to find a set of distinctive **keypoints** that can be reliably localized under varying imaging conditions, viewpoint changes, and in the presence of noise.

Features descriptors

Once a set of interest regions has been extracted from an image, their content needs to be encoded in a descriptor that is suitable for discriminative matching.

Features detectors and descriptors

Brief Explanation

- *HARRIS detector*: the keypoints it finds corresponds to **corner-like** structures
- *HOG descriptor*: local object appearance and shape within an image can be described by the *distribution of intensity gradients* or *edge directions*.
  The image is divided into small connected regions called **cells**
- *SIFT descriptor*: this descriptor aims to achieve robustness to **lighting variations** and **small positional shifts**
- *SURF detector*: similar to SIFT
- *MSER detector*: extracts **homogeneous intensity regions**

Features detectors and descriptors

Examples of various algorithms

Figure 2: HARRIS-SIFT

Figure 3: ORB-ORB

Figure 4: SIFT-SIFT

Figure 5: SURF-SURF

Machine Learning Techniques

Different approaches & classification strategies

Use a classifier as:

- **Global** Classifier
- **OneClass** Classifier

Construct the dataset using as attributes:

- **Keypoints** of each image as a single row of the classifier
- All keypoints of a **single image** as a row of the classifier

Machine Learning Techniques

Dimensionality Reduction: PCA

Question: *Is our dataset suitable for PCA analysis?*

Attention

If the attributes are not correlated each other, the PCA analysis leads to variables with *low explained variance* and so they cannot capture the variability of the data

We noticed that reducing the number of components to 2, 3,. . . ,10,. . . and so on the overall explained variance ratio is low, with peaks of 30% maximum . . .

This is caused by the way the features are computed!

Machine Learning Techniques

Learning algorithms: Random Forest©

Random Forest© is an ensemble learning algorithm that, instead of using only one-time learners, builds a bag of learners sampling with replacement from the training set (*Bagging*).

Example

$n = 50$ trees are grown in order to reduce the probability of misclassification→this works if the failure rate of a single tree classifier is less than $\epsilon = 0.5$, from the *binomial distribution*

$$p_{err} = \sum_{i=26}^{50} \binom{50}{i} \epsilon^i (1-\epsilon)^{50-i}$$

This represents the probability that at least 26 learners out of 50 make the wrong prediction

It can be shown that, under the previous condition on ϵ, $p_{err} \ll \epsilon$.

Our Idea

How to Classify Objects

Procedure to guess the correct classification:

1. Extract features from the images.
2. Do a k-fold validation from images contained into the dataset (splitting it in training and test images).
3. For each fold (training and test set) make a prediction of the images and compute the related probability of belonging to a class.
4. Evaluate the models

Contents 3

Case Study

Main outline

To implement our system we will follow these steps:

- Use of built-in functions provided by `OpenCV` to deal with features detection and description,
- explain the structure of our scripts written in `Python`,
- introduce the ML algorithms and the validation methods to build a strong learner.

OpenCV

Wrapper methods in OpenCV

OpenCV lets the programmer use some *wrapper methods* in order to get the features **detector** and **descriptor** from an image as it is shown in the code below:

Listing 1: detectAndExtract

```
class detectAndExtract:
  _detector = None
  _descriptor = None

  def __init__(self,detector,descriptor):
    self._detector = detector
    self._descriptor = descriptor

  def elabora(self.path):
    # Calculate features ...
    return out.tolist()
```

The FeaturesFile's Class

Files' creation and loading

Now we have to save features in a file in order to train our model with Machine Learning algorithms.
So we have decided to create this class that can save all features of a given keyword in a file.
It wouldn't have been a good idea if this file had been created every time ex-novo and so if the file exists, the class loads into an array the features contained in the file.

Features file

File is a `.csv` file and its name is composed like this:
`keywordsname_DETECTOR_DESCRIPTOR.csv`

We can have different files, one for every features extracted.

Machine Learning Implementation

Random Forest© Classifier

Random Forest Classifier

It's a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting.

Listing 2: Parameters of RandomForestClassifier

```
clf = RandomForestClassifier(n_estimators=300,n_jobs=-1)
```

The parameters are the following:

- `n_estimators`: 300 → The number of trees grown in the forest.
- `criterion`: *gini* → The function to measure the quality of a split (Gini impurity or entropy).
- `n_jobs`: -1 → The number of jobs to run in parallel for both fit and predict. (-1 stands for: all cores).

The Features Learning's Class

To implement the algorithm and use it easily with the other part of the code, we have created `FeaturesLearning` Class. Given positive and negative instances, it can train a model using a `cross_validation.StratifiedKFold` [1] and then it returns an array of probabilities for each fold that indicates belonging to the class.
This procedure is invoked in the `Main Class` for each category in the database, given an image to classify.

[1] to preserve the proportion between positive and negative examples

The Features Learning's Class

Code's overview

Then in Class `FeaturesLearning` we have these methods.

Listing 3: Methods of Fearures Learning Class

```
class FeaturesLearning(object):
  """docstring for FeaturesLearning"""
  X_positive = None
  X_negative = None
  clf = RandomForestClassifier(n_estimators=300,n_jobs=-
  def __init__(self, X_positive, X_negative,ffe):
    # Constructor Method ...
  def trainModel(self):
    # To train model with cross_validation kfold ...
  def _prediction(self,featuresFile,class_test=None):
    # To make a prediction ...
```

Probability

predict_proba() function

The predicted class probabilities of an input sample is computed as the mean predicted class probabilities of the trees in the forest.
The class probability of a single tree is the fraction of samples of the same class in a leaf.

Making the final decision...

If the ensemble is composed of n trees, then the prediction is made counting how many trees predicted class value "0" and class value "1".
If $n = 300$ and 200 trees predicted "0" then $p_0 = \frac{200}{300} \approx 0.67$ and $p_1 = 1 - p_0 \approx 0.33$

Evaluation

Evaluation parameters

For the evaluation of the models we consider:

- *Confusion Matrix*[2]: $\begin{bmatrix} TP & FN \\ FP & TN \end{bmatrix}$
- *Precision*: $\frac{TP}{TP+FP}$
- *Recall*: $\frac{TP}{TP+FN}$
- *Accuracy*: $\frac{TP+TN}{TP+TN+FP+FN}$

[2] it tells the classifier's class assignment

Evaluation

Examples of various implementations: Example 1

Description

- Stratified k-fold cross-validation with $n_{folds} = 3$
- Classification using images as a single row of the classifier
- `sklearn.multiclass.OneVsRestClassifier` permits an hybrid approach (see next slides)

	Precision	*Recall*	*Accuracy*
Fold 1	0.4686	0.4681	0.4697
Fold 2	0.4211	0.4142	0.4198
Fold 3	0.4855	0.4892	0.488

Evaluation

Examples of various implementations: Example 1

$$\begin{bmatrix} 7 & 1 & 1 & 2 & 3 & 1 & 1 & 0 \\ 1 & 9 & 0 & 2 & 3 & 0 & 2 & 0 \\ 2 & 2 & 6 & 1 & 1 & 1 & 1 & 3 \\ 3 & 0 & 0 & 4 & 1 & 2 & 6 & 0 \\ 1 & 2 & 2 & 2 & 5 & 0 & 5 & 0 \\ 1 & 0 & 1 & 2 & 1 & 10 & 1 & 1 \\ 1 & 1 & 0 & 1 & 2 & 1 & 11 & 0 \\ 0 & 1 & 4 & 0 & 2 & 3 & 1 & 3 \end{bmatrix}$$

Confusion Matrix: example for Fold 2

```
Original
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5
 5 5 5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7 7
 7 7 7 7 7 7 7 8 8 8 8 8 8 8 8 8 8 8 8 8 8]
################
Predicted
[8 8 8 8 1 4 5 1 1 5 3 8 3 4 4 5 8 3 2 6 6 1 3 6 2 5 1 2 4 4 1 2 2 3 3 3 3
 5 3 3 3 1 3 1 4 5 1 3 1 3 3 6 4 4 4 6 4 4 3 8 7 6 8 4 4 4 6 1 2 5 3 2 6 7
 7 1 3 1 5 1 2 3 5 1 3 6 3 1 2 3 1 6 4 1 4 5 6 6 6 1 1 7 7 6 4 7 8 7 1 7 4
 4 2 4 2 7 7 7 5 8 6 5 4 8 8 8 5 8 2 5 2 1]
################
Evaluation

[[ 3  0  2  3  3  0  0  6]
 [ 3  5  3  2  1  3  0  0]
 [ 4  0 10  1  2  0  0  0]
 [ 0  0  1  8  0  4  1  2]
 [ 5  3  3  0  3  1  2  0]
 [ 5  1  3  2  1  5  0  0]
```

Evaluation

Examples of various implementations: Example 2

Description

- Stratified k-fold cross-validation with $n_{folds} = 3$
- Classification using keypoints of an image as a single row of the classifier

	Precision	*Recall*	*Accuracy*
Fold 1	0.7863	0.7419	0.8493
Fold 2	0.7695	0.6918	0.8264
Fold 3	0.7724	0.6604	0.7958

Evaluation

Examples of various implementations: Example 2

$$\begin{bmatrix} 9 & 1 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 45 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 3 & 7 & 1 & 0 & 0 & 1 & 0 \\ 0 & 3 & 0 & 32 & 0 & 0 & 0 & 0 \\ 2 & 3 & 0 & 2 & 17 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 & 6 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Confusion Matrix: example for Fold 2

```
Original
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5
 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
 6 6 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8
 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8]
################
Predicted
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 4 1 1 8 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Evaluation

Examples of various implementations: Example 3

Description

- Stratified k-fold cross-validation with $n_{folds} = 3$
- Classification using images as a single row of the classifier
- "*Fair*" OneVsAll classifier: model trained using 50% of positive examples and % of negative examples for each fold

	Precision	*Recall*	*Accuracy*
Fold 1	0.7333	0.6111	0.6765
Fold 2	0.5217	0.7059	0.5294
Fold 3	0.6923	0.5294	0.6471

Evaluation

Examples of various implementations: Example 3

$$\begin{bmatrix} 12 & 4 \\ 7 & 11 \end{bmatrix}$$ Confusion Matrix: example for Fold 1

```
[  0   1   3   4   6   7   9  10  11  13  14  15  16  19  20  23  24  25
  26  28  31  33  34  35  36  38  40  41  42  43  45  48  49  53  54  57
  59  60  63  65  67  68  69  70  74  77  78  79  80  81  82  83  85  86
  87  89  90  91  92  93  95  96  98 101 103 104]
[ 2  5  8 12 17 18 21 22 27 29 30 32 37 39 44 46 47 50 51 52 55 56 58 61 62
 64 66 71 72 73 75 76 84 88]
################
Original
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
################
Predicted
[0 0 0 1 0 0 0 0 1 1 0 0 0 1 1 1 0 0 0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1]
################
Evaluation

[[11  6]
 [ 4 13]]
Precision Score:
0.684210526316
Recall Score:
0.764705882353
```

Figure 8: Screenshot of the output of the script

Evaluation

Examples of various implementations: Example 4

Description

- Stratified k-fold cross-validation with $n_{folds} = 3$
- Classification using images as a single row of the classifier
- "*Pure*" OneVsAll classifier
- **Idea**$\rightarrow$*Cost sensitive classifier*: the decisions of the classifier are weighted over a *cost matrix* for TP,TN,FP,FN

$\rightarrow$ Some troubles with `costcla` libraries for `Python`, due to the structure of data...

Contents 4

Final Considerations

- Is database approach faster?
- The influence of preprocessing (*example*$\rightarrow$ in literature, saliency maps for object extraction...)
- Other type of dimensionality reduction methods, but...be careful!
- Deep knowledge of the features detector/description algorithms could lead to a better classification...
- Choose the approach you feel more confident with: focus on keypoints or images as instances, considering pros and cons of every approach (see examples before...)