Project 2 report

Patrick Indri
January 3, 2020

Problem statement

The assignment requires the implementation of an image classifier based on the bag-of-words approach. The provided dataset, taken from [1], consists of $\approx$ 4500 images from 15 categories. The dataset is already divided into a test and a training set. The task has been carried out using the Python programming language. Please refer to the provided `Readme.md` for quick details on how the project is organised into folders.

Disclaimer on `OpenCV`

Since version 3.0, OpenCV no longer includes widely used patented algorithms such as SIFT and SURF. However, those algorithms are included as "extra modules" in the opencv_contrib GitHub repository. The library must be compiled from source, specifying the inclusion of the "non free" algorithms. As an alternative, one could use the ORB algorithm, a free alternative to SIFT or SURF.

1. Visual vocabulary

The first step requires the construction of a visual vocabulary. Images are read using a generator that yields the image path and its folder name (that is, its class label). Then, for each of the images in the training

set, the SIFT algorithm is used to extract keypoints and compute descriptors. The number of features to be retained can be passed as an argument, but was left at the default value (which, incidentally, is not clearly specified by the documentation). The descriptors and the labels are associated to an image ID and everything is stored in a Pandas `Dataframe`. As required, 1000000 descriptors are sampled and clustered using k-means. The `sklearn.cluster.KMeans` function returns an object containing info on the clusters including the coordinates of the centroids. Different numbers of clusters have been tested.

Results

Training set, descriptors dataframe.

image_id	descriptor	label
3	[8.0, 2.0, 46.0, 42.0, 2.0, 19.0, ...	Industrial
3	[18.0, 2.0, 0.0, 0.0, 4.0, 49.0, ...	Industrial
3	[10.0, 1.0, 0.0, 20.0, 4.0, 14.0, ...	Industrial
3	[57.0, 33.0, 121.0, 3.0, 0.0, 1.0, ...	Industrial
4	[16.0, 20.0, 8.0, 4.0, 24.0, 17.0, ...	Industrial
4	[35.0, 9.0, 11.0, 27.0, 0.0, 0.0, ...	Industrial
4	[38.0, 0.0, 1.0, 9.0, 4.0, 0.0, 0.0, ...	Industrial

2. Histogram representation

Each of the k cluster centroids is interpreted as a visual word, thus constituing a vocabulary. Training set images can now be represented as histograms having k bins: for each image, the code cycles through the list of its descriptors and increases by one the bin correspoinding to the closest word in the vocabulary (that is, the closest centroid). Euclidian norm is used to compute all the distances while `np.argmin` is used to identify the smallest one. Since different images might have a different number of descriptors, the histograms are then normalised to make them comparable with each other. The normalised histograms are added as a column to the descriptors dataframe.

Results TBD

Training set, descriptors dataframe.

image_id	descriptor	label
3	[8.0, 2.0, 46.0, 42.0, 2.0, 19.0, ...	Industrial
3	[18.0, 2.0, 0.0, 0.0, 4.0, 49.0, ...	Industrial
3	[10.0, 1.0, 0.0, 20.0, 4.0, 14.0, ...	Industrial
3	[57.0, 33.0, 121.0, 3.0, 0.0, 1.0, ...	Industrial
4	[16.0, 20.0, 8.0, 4.0, 24.0, 17.0, ...	Industrial
4	[35.0, 9.0, 11.0, 27.0, 0.0, 0.0, ...	Industrial
4	[38.0, 0.0, 1.0, 9.0, 4.0, 0.0, 0.0, ...	Industrial

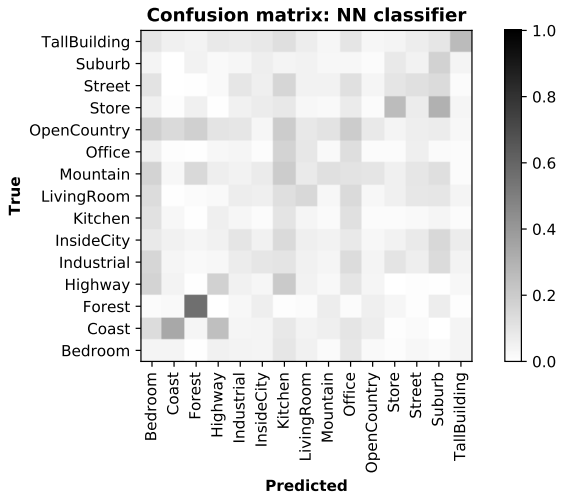
3. Nearest Neighbour classifier

Test images can be classified using a nearest neighbour classifier. In particular, histograms for the test images are computed as previously done for the training set (not that, in this step, the visual vocabulary must not be rebuilt: the training one is used to compute the histograms). Then the histogram of a test image is compared with all the training histograms: the classifier assigns to the test image the label corresponding to the closest train histogram. Distances between histograms can be computed using the *Earth Mover Distance*, implemented in `scipy.stats.wasserstein_distance`.

It should be noted that the histograms for the test set are precomputed once and then fed to different classifiers, reducing classification overhead. An alternative approach would be to designate the computation to each of the classifiers. All the implemented classifiers present a similar interface, accepting the train and test dataframes as inputs and returning both the true and the predicted labels for the test set.

Results

Confusion matrix (for $k = 48$) and accuracy for different values of k.



Clusters (k)	Accuracy
24	tbd
48	0.16
96	tbd
192	tbd

The plot and the table above synthesize the results of the nearest neighbour classification. The accuracy values and the confusion matrix show that this classifier does not perform well on the test set. It performs significantly better than the random classifier ($\approx 7\%$ accuracy) and the naive classifier that always predicts the most frequent class ($\approx 10\%$ accuracy); however, all classes but the *forest* one seem to be randomly predicted.

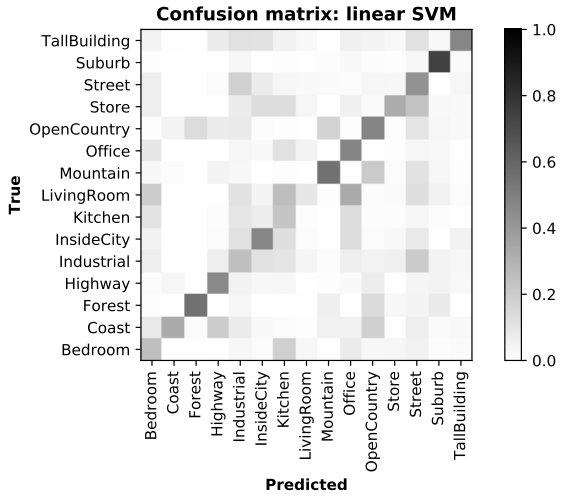
4. and 5. Linear SVM

In this steps a multi-class Support Vector Machine classifier with linear kernel was implemented. In particular, the SVM is implemented using the one-vs-rest approach, training a binary SVM for each of the classes to separate it from the remaining ones. The normalised histograms constitute the input vectors. To overcome the unclassifiable regions problem, distance from the separating hyperplanes is used as the classification criterion: the image is assigned to the class corresponding to the most distant SVM separating hyperplane.

Concerning implementation details, the SVM linar classifier is provided by `sklearn.svm.SVC` using `kernel = linear`. The distance from the hyperplane is computed, as suggested by the documentation, taking the value of the classifier `decision_function` on the histogram and dividing it by the classifier `coef_`.

Results

Confusion matrix (for $k = 48$) and accuracy for different values of k.



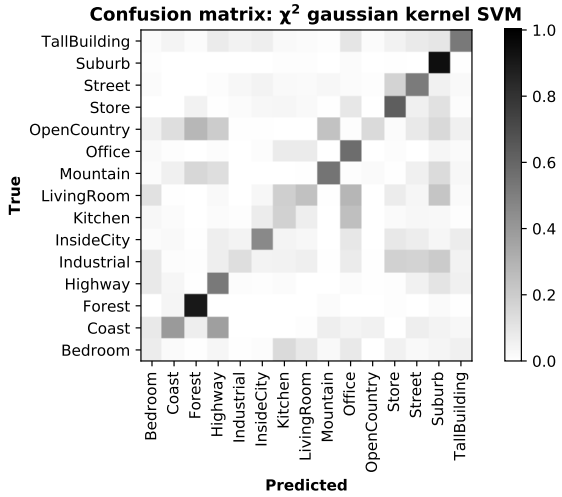
Clusters (k)	Accuracy
24	tbd
48	0.39
96	tbd
192	tbd

6. and 7. Gaussian kernel SVM

A multi-class SVM classifier with Gaussian kernel was implemented as well. In this case, the `sklearn.multiclass.OneVsRestClassifier` function, fed with `skearn.svm.SVC(kernel = precomputed)`, is employed to build the classifier. In order to fit such classifier on the training data, the Gram matrix of the training set must be provided as an argument. The proposed implementation of the Gram matrix takes two sets of histograms as arguments; by default it uses a Gaussian kernel and the χ^2 distance, but Earth Mover Distance was tested as well. The predictions are carried out passing the Gram matrix build over both train and test histograms to the classifier.

Results

Confusion matrix (for $k = 48$) and accuracy for different values of k.



Clusters (k)	Accuracy
24	tbd
48	0.42
96	tbd
192	tbd

0.21 for edm distance with k = 48

8. Error Correcting Output Code SVM

Error Correcting Output Code is an ensemble method for multi-class classification. The approach, detailed in [2] and [3], combines many binary classifiers in order to solve a multi-class problem. In particular, each class is represented by a binary string where the length of the string corresponds to the number of binary classifiers; such representation is stored in a binary coding matrix:

Example: coding matrix using 15 classifiers for 10 classes.

Class	Z_1	Z_2	Z_3	Z_4	Z_5	Z_6	...	Z_{15}
0	1	0	1	0	0	1	...	1
1	0	0	1	0	1	1	...	1
⋮								
9	1	0	0	1	1	0	...	0

Following the proposed example, the first binary classifier Z_1 should learn the representation 1 for class 0, representation 0 for class 1, and so on. Once all the classifiers have been fitted they can be evaluated on a test observation to produce a bit representation of it. The class whose expression (in the coding matrix) is the closest to the test observation representation is chosen as the predicted label.

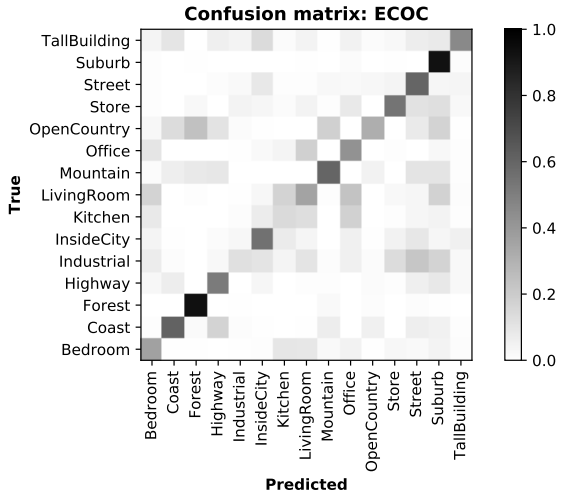
The coding matrix can be manually constructed to maximise expressiveness (guaranteeing a good separation between the classes representations) but, particularly for a large number of classes, a randomly generated matrix performs well. As the authors of [3] argue, the main benefit of the approach is variance reduction via model averaging, which is comparable for optimally constructed and random coding matrices. The approach is not prone to overfitting and a large number of classifiers provides better results.

In the proposed implementation the coding matrix is randomly generated and the distance between binary strings is computed using the Hamming distance.

Results

Plot varying B.

Confusion matrix (for $k = 48$) and accuracy for different values of k. 100 binary classifiers.



Clusters (k)	Accuracy
24	tbd
48	0.48
96	tbd
192	tbd

References

[1] Svetlana Lazebnik, Cordelia Schmid e Jean Ponce. «Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories». In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. Vol. 2. IEEE. 2006, pp. 2169–2178.

[2] Thomas G Dietterich e Ghulum Bakiri. «Solving multiclass learning problems via error-correcting output codes». In: *Journal of artificial intelligence research* 2 (1994), pp. 263–286.

[3] Gareth James e Trevor Hastie. «The error coding method and PICTs». In: *Journal of Computational and Graphical Statistics* 7.3 (1998), pp. 377–387.