Computer Vision, Object Recognition Practical Work

Miquel Perelló Nieto and Marc Albert Garcia Gonzalo

***Abstract*—This document contains the student written report for the third practical work, of the Computer Vision course, in the Master in Artificial Intelligence.**

The report contains the results of the experimentation based on the script provided by the professor.

***Keywords*—Computer Vision, Object recognition, Machine learning, Supervised learning, Support Vector Machine, SVM.**

I. Introduction

OBJECT recognition is the branch of computer vistion that focus on the task of finding a given object in an image.

In this practical work, we will experiment with different techniques for object recognition.

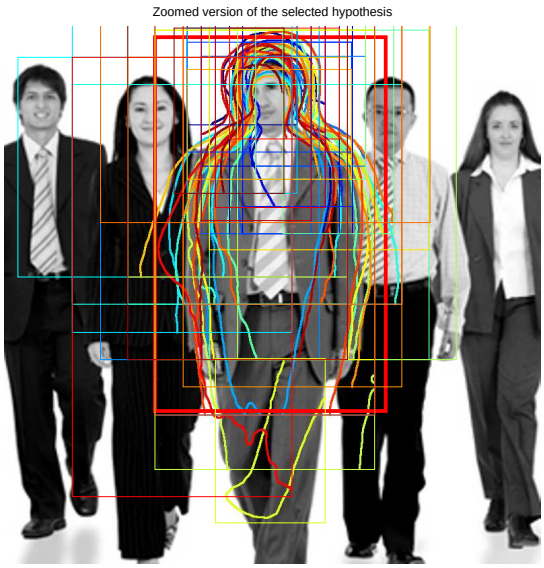
In the first part, we will experiment on poselets. Poselets are image fragments which capture part of the pose of an object from a given viewpoint. Specifically we will focus on the poselets for people.

The second part is devoted to the recognition of handwritten digits, using Support Vector Machines. Linear and Radial Basis Function kernels are used.

In the last part, we focus on image classification. Different techniques for Support Vector Machines are used, and changes on the procedure will be analyzed. Datasets for aeroplans, motorbikes and people are used.

II. Poselets

THE next image is the result of applying the UC Berkeley poselets demo, to a random image where some people is visible.



As we can appreciate, most of the hypothesys (displayed with different colors over the image, match approximately the contour of the most visible person on the set.

Experimentation with different image, show how the hypothesys are better, as more *normal* is the pose of the individual. In the reported image, the individual is standing, facing front, and with an average body type. People with less conventional poses, or body types, like obese people, have less accurate hypothesys.

III. SVM

IN this section, we describe the results of applying Support Vector Machines to classify handwritten digits. As there are 10 different digits to classify, we will use the one-vs-all method of classification. In all cases, we used different C soft magin parameters with values 0.01, 0.1, 1, 2 and 4. We report the results for the parameter with best validation values.

A. SVM with linear and 1000 instances

In this case, we trained the SVM with 100 samples of each class, and a linear kernel. The accuracy obtained is 0.8091, and the best value for the C soft margin parameter was 0.01.

Next we present the confusion matrix we obtained:

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	859	0	3	3	2	5	6	0	1	1
1	11	1006	8	3	1	2	2	0	2	0
2	110	21	728	20	17	3	3	12	16	2
3	92	1	44	724	2	16	1	8	16	6
4	56	1	8	18	750	0	8	2	3	36
5	136	11	9	47	24	526	10	5	21	3
6	54	3	41	0	22	38	699	0	1	0
7	38	17	19	31	13	1	0	773	4	32
8	152	16	9	40	10	57	5	16	555	14
9	85	8	5	23	59	4	0	45	18	662

B. SVM with linear kernel and 5000 instances

For this experiment, we used the same kernel, and the same parameters as in the previous, but with five times more data. We trained with 500 instances of each class. Note that as the available dataset is the same, increasing the number of instances for training, decreases the number of instances for validation.

As expected, the accuracy in this experiment is higher than the experiment before, because usually, more data generates better results. The accuracy in this case is 0.8434, more than a 4% better. The best value for the C soft margin parameter has been the same as the previous case, 0.01.

Next we present the confusion matrix we obtained:

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	467	0	1	2	2	4	1	1	1	1
1	1	627	2	2	0	3	0	0	0	0
2	31	10	468	4	3	1	2	8	3	2
3	22	0	34	430	1	8	1	2	11	1
4	12	1	4	2	456	2	2	0	2	1
5	21	4	4	47	15	288	4	2	7	0
6	19	5	23	1	9	35	362	4	0	0
7	16	6	21	27	14	10	0	425	1	8
8	44	5	16	15	2	51	4	5	328	4
9	22	7	2	12	41	16	0	27	16	366

C. SVM with RBF kernel and 1000 instances

Finally, we performed the experiment, again, using 100 instances of each class for training, but in this case using a Radial Basis Funcion (RBF) kernel. This kernel performs a gaussian transformation on the data space, allowing the SVM to find a margin which is not linear in the original space. This kernel has another parameter *sigma* (sometimes called *gamma*), for which we tried different values, as in the case of the C parameter. Specifically, we performed the experiment with the values 1, 4, 16, 64.

In this case, the accuracy obtained in the best case, is similar to the one obtained for the linear kernel, with the same number of instances. The accuracy is 0.8073, and the best parameters we obtained this accuracy with, are 4 for the C soft margin parameter, and 64 for the sigma parameter of the kernel.

Next we present the confusion matrix we obtained:

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	868	0	0	1	0	6	3	1	1	0
1	6	1010	4	0	1	5	4	0	5	0
2	86	17	779	2	14	1	16	6	11	0
3	62	8	101	683	1	25	3	11	9	7
4	30	8	9	0	814	1	4	0	2	14
5	53	12	6	120	29	547	10	1	9	5
6	82	5	13	0	13	58	686	0	1	0
7	75	29	24	6	27	2	0	746	1	18
8	83	23	21	41	10	52	13	8	614	9
9	44	7	5	10	190	13	0	102	19	519

D. Results summary

In the next table, the best parameters and accuracies for each experiment are summarized:

	training / 10	**C**	**accuracy**
Linear	100	0.01	0.8091
Linear	500	0.01	0.8434
RBF (sigma=64)	100	4.00	0.8073

IV. Image classification

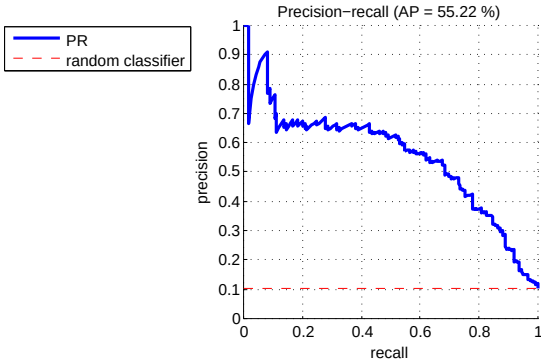
A. Stage C: Classify the test images and assess the performance

Note the bias term is not needed for this ranking, only the classification vector w. Why?

In general, SVMs are used for classification, so the bias term is important, as removing it supposes displacement of the hyperplane in the direction of one of the classes, and it will missclassify the elements.

But in this case, we are using the hyperplane to calculate how much every instance is far to one side of it, considering positive the ones at that side and negatives in the other side. If all we want is to create a ranking with these distances, a displacement on the hyperplane on the bias direction is like adding or substracting a constant to each distance, so it will not have any effect on the final result.

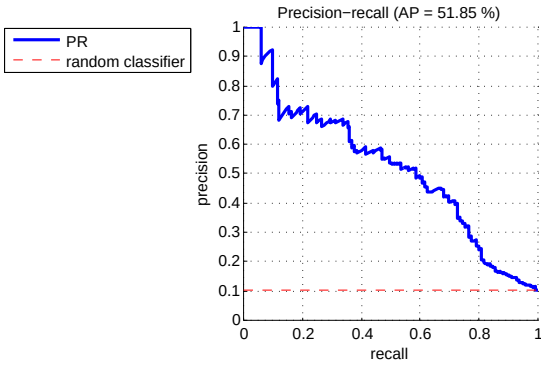
1) Aeroplanes:



Number of training images : 112 positive, 1019 negative
Number of testing images : 126 positive, 1077 negative
Test AP : 0.55
Correctly retrieved in the top 36 : 24

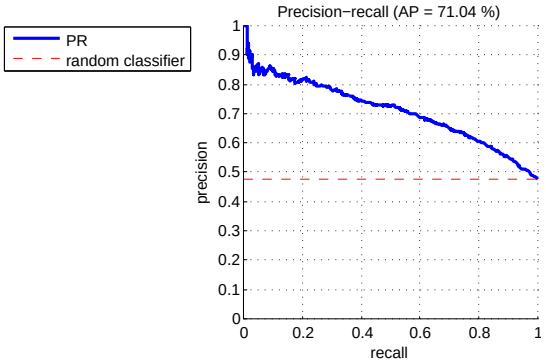
B. Stage D: Learn a classifier for the other classes and assess its performance

1) Motorbikes:



Number of training images : 120 positive, 1019 negative
Number of testing images : 125 positive, 1077 negative
Test AP : 0.52
Correctly retrieved in the top 36 : 26

2) Person:



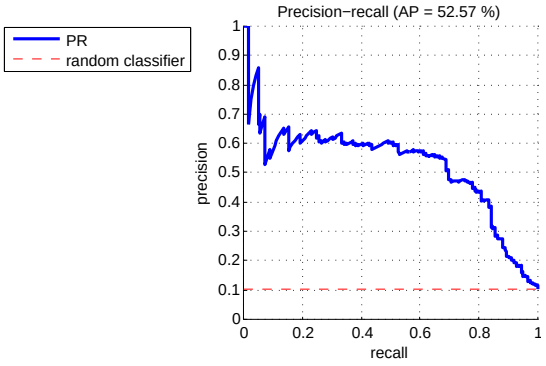
Number of training images : 1025 positive, 1019 negative
Number of testing images : 983 positive, 1077 negative
Test AP : 0.71
Correctly retrieved in the top 36 : 30

Does the AP performance match your expectations based on the variation of the class images?

Yes, because for the case of motorbikes the AP is similar, as the training amount of data is also similar. The small decrease is probably explained by how difficult is to predict a kind of object and the other, and it looks like it is a little bit easier to predict an aeroplane than a motorbike. But the cause can also be simply because of statistical noise.

In the case of people, as the dataset is approximately 10 times betters, we can expect to have better results, and it is the case. Again, part of the better accuracy can be explained because it is possible easier to identify a person, than identifying an aeroplane or a motorbike. We could expect this, as the variations among people look smaller than the variances among different kinds of aeroplanes and motorbikes.

C. Stage E: Vary the image representation

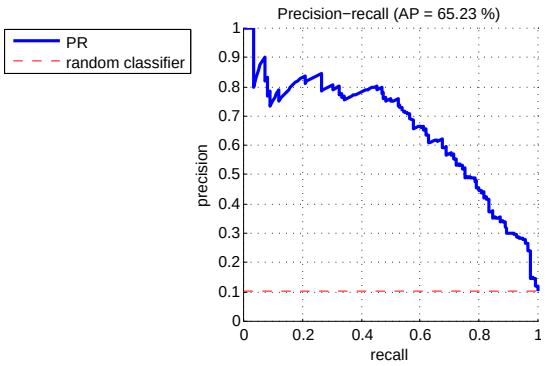


Number of training images : 112 positive, 1019 negative
Number of testing images : 126 positive, 1077 negative
Test AP : 0.53
Correctly retrieved in the top 36 : 22

What is your explanation for the change in performance?

The positional information has been removed from the features, and the performance has decreased, so we can consider that this information was relevant, and not noise. This makes sense, as the probability of finding the wings of an aeroplane in an image is probably not the same for all regions of the image. This is not an accurate description, because the windows will not probably match a wing size in most cases, but it gives an idea of the importance of the positions.

D. Stage F: Vary the classifier



Number of training images : 112 positive, 1019 negative
Number of testing images : 126 positive, 1077 negative
Test AP : 0.65
Correctly retrieved in the top 36 : 30

Why is this procedure equivalent to using the Hellinger kernel?

The Hellinger kernel is defined as follows:

$$k(h, h') = \sum_{i=1}^{N} \sqrt{h(i)h'(i)} \tag{1}$$

This can also be expressed as next, because the square of a product is the product of the squares:

$$k(h, h') = \sum_{i=1}^{N} \sqrt{h(i)h'(i)} = \sum_{i=1}^{N} \sqrt{h(i)}\sqrt{h'(i)} \tag{2}$$

So, for performing the kernel computation, the instances are simply squared. Instead of this computation in the kernel, we can compute the squared root in the data, so the kernel equation will be:

$$k(h, h') = \sum_{i=1}^{N} h(i)h'(i) \tag{3}$$

Which is the definition of the linear kernel. This makes equivalent the use of the Hellinger kernel, and the use of the linear kernel with the features squared.

Why is it an advantage to keep the classifier linear, rather than using a non-linear kernel?

The advantage is that the model gets simpler, less likely to overfit, and with lower computational cost. For a linear SVM it is not necessary to compute the kenel, so it is not necessary to store the support vectors, and only the weights and the bias. A non-linear classifier can have a large set of support vectors, depending on its complexity, so this difference can be important.

Try removing the L2 normalization step. Does this affect the performance? Why?

Number of training images : 112 positive, 1019 negative
Number of testing images : 126 positive, 1077 negative
Test AP : 0.66
Correctly retrieved in the top 36 : 30

It looks like the performance does not change in a significant way. This is probably caused because the histograms are already normalized, so the features are indirectly normalized too.

Go back to the linear kernel and remove the L2 normalization step. What do you observe?

Number of training images : 112 positive, 1019 negative
Number of testing images : 126 positive, 1077 negative
Test AP : 0.55
Correctly retrieved in the top 36 : 30

We observed that the performance does not change in a significant way either. The reason should be the same as in the previous case.

E. Stage G: Vary the number of training images

What performance do you get with the linear kernel? And with the Hellinger kernel?

Do you think the performance has 'saturated' if all the training images are used, or would adding more training images give an improvement?

1) Linear kernel:

Number of training images : 12 positive, 101 negative
Number of testing images : 126 positive, 1077 negative
Test AP : 0.57
Correctly retrieved in the top 36 : 26

Number of training images : 56 positive, 510 negative
Number of testing images : 126 positive, 1077 negative
Test AP : 0.65
Correctly retrieved in the top 36 : 31

Number of training images : 112 positive, 1019 negative
Number of testing images : 126 positive, 1077 negative
Test AP : 0.66
Correctly retrieved in the top 36 : 30

As we can see, with the linear, the performance improves as we get more data.

The diference from 10% to 50% of the dataset is very significant, from 0.57 to 0.65. But then, from 50% to 100% the performance improves, but in a very subtle way, so we do think the performance starts to be saturated for the data growth, and we would not expect great improvements by adding more data.

2) Hellinger kernel:

Number of training images : 12 positive, 101 negative
Number of testing images : 126 positive, 1077 negative
Test AP : 0.41
Correctly retrieved in the top 36 : 20

Number of training images : 56 positive, 510 negative
Number of testing images : 126 positive, 1077 negative
Test AP : 0.50
Correctly retrieved in the top 36 : 30

Number of training images : 112 positive, 1019 negative
Number of testing images : 126 positive, 1077 negative
Test AP : 0.55
Correctly retrieved in the top 36 : 30

Again, the performance of the model gets better as we use more data for training. This is the expected behavior for any machine learning model, unless some special circumstances occur.

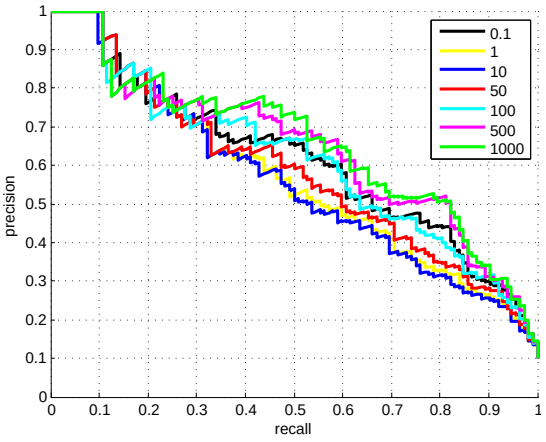
In this case, the performance seems to improve significantly as more data is added. The increment from using 10% of the dataset, to using 50% is an AP of 0.41 to 0.50. From 50% to 100% is lower, from 0.50 to 0.55, but it is still significant. So we would expect the performance to improve, if we can train the model with more data.

F. Links and further work

In Part I, vary the C parameter in the range 0.1 to 1000 (the default is C=100), and plot the AP on the training and test data as C varies for the linear and Hellinger kernels.

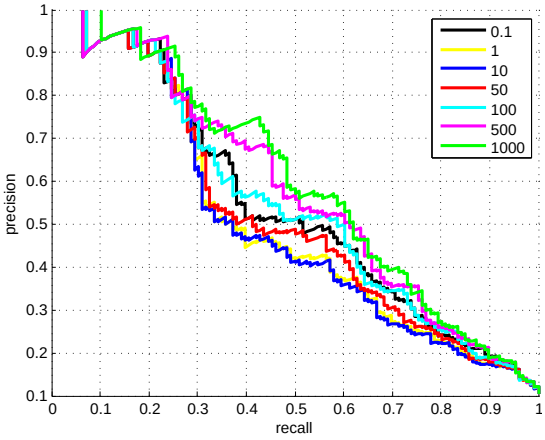
	0.10	**1**	**10**	**50**	**100**	**500**	**1000**
Train linear	0.62	0.57	0.56	0.59	0.61	0.64	0.66
Test linear	0.54	0.51	0.50	0.52	0.55	0.58	0.60
Train Hel.	0.80	0.84	0.99	1	1	1	1
Test Hel.	0.67	0.69	0.69	0.64	0.66	0.66	0.66

1) Linear kernel, training data:



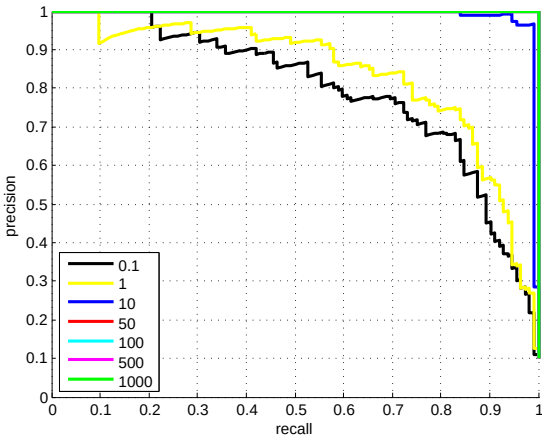
In this case, there are no big differences between the performance, for different values of C. But in general, we can see how larger values obtain better results.

2) Linear kernel, test data:



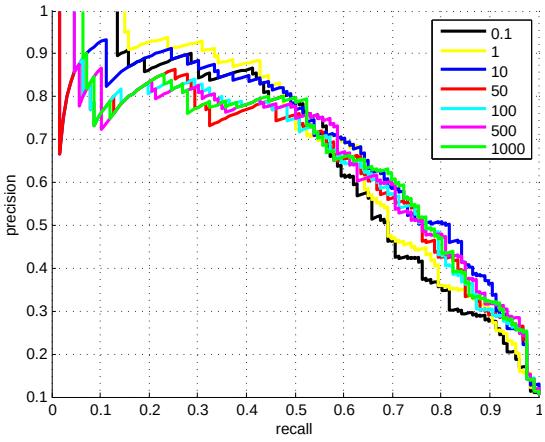
Again, we do not observe big difference, but larger values perform better. As expected, the performance is worse with test (independent) data, but the difference does not seem large, so the model does not look to be overfitting in a significant way.

3) Hellinger kernel, training data:



In this case, we can see how the performance is perfect for large values of C. This is likely because the model is overfitting the training data. As we can see, smaller values of C, prevent this overfitting, and seem to be generalizing better.

4) Hellinger kernel, test data:



Seeing the performance with independent data, we can confirm that the model was clearly overfitting. The graphic shows how performance is far from perfect for data not in the training set. We can also see how small values of C, are getting better results for the best scored cases. This is the result of better generalization.

V. Conclusion

THE puropse of this practical work has been to see *in action* different techniques in object recognition.

First we have seen how poselets could predict, in normal cases, the contour of people in images.

Then, we saw how to use Support Vector Machines for recognizing handwritten digits. We used a one-vs-all multiclass classifier, with different kernels and different parameters. The results were acceptable.

Finally, we experimented with image classification with different variations. We used different objects, and we have seen the differences on the results among them, and how some objects seen easier to predict than others. We also have seen how the amount of training data is correlated with the performance. We have seen how to evaluate the performance based on ranking the data, instead of directly classifying, and how to interpret the precision, the recall, precision-recall plots, and the average precision (AP). We finally saw how the soft margin C parameter affects the results.