Object Recognition Practicum

Computer Vision

University of Barcelona

December 14, 2015

Authors:
Camps Sereix, Julià

1 Digit Classification with Support Vector Machines

- Train a SVM with 100 random samples per class and a linear kernel (you can use different C soft margin parameters: 0.01, 0.1, 1, 2, 4). Report the accuracy you get on the test set (the rest of samples) and the best C value. Report the confusion matrix you get.

  For this exercise I run several times the SVM with different C values, and for each pair of digit sets I trained a SVM and tested it with some values of C that had an interesting influence on the obtained accuracy results, because for big C values it performed wrong, and also for too small C's. The C values that I choose where: $C = 0.0001$, $C = 0.0003$ and $C = 0.0005$. And for each pair of digits classification I kept the best C value, the accuracy obtained and the confusion matrix generated.

  From tables table 1, table 2, table 3 and table **??** how the digits are classified pairs of digits classification information, the table is sorted according to the best accuracy obtained for each digits pair on each table.

Classifications	Accuracy	Confusion Matrix	Best C value
9vs4	0.91261	[921 86;88 896]	0.0005
5vs3	0.91851	[827 90;65 920]	0.0005
8vs3	0.92389	[836 13;138 997]	0.0005
8vs2	0.92672	[875 48;99 984]	0.0005
9vs7	0.9298	[981 115;28 913]	0.0005
9vs8	0.93091	[988 116;21 858]	0.0005
8vs5	0.93408	[875 24;99 868]	0.0005
9vs5	0.94424	[979 76;30 816]	0.0005
3vs2	0.94466	[990 93;20 939]	0.0005
5vs2	0.95426	[873 69;19 963]	0.0005
9vs3	0.95443	[978 61;31 949]	0.0005
6vs5	0.95838	[908 27;50 865]	0.0005
7vs2	0.96117	[993 45;35 987]	0.0005
6vs2	0.96231	[938 55;20 977]	0.0005
7vs3	0.96663	[976 16;52 994]	0.0005
7vs1	0.97041	[964 0;64 1135]	0.0005
8vs6	0.97101	[934 16;40 942]	0.0005
8vs1	0.97202	[933 18;41 1117]	0.0005

Table 1: Best result table, sorted in accuracy ascending order, for each pair of different digits classification results using linear SVM, and 100 instances for training.

Classifications	Accuracy	Confusion Matrix	Best C value
4vs2	0.97219	[964 38;18 994]	0.0005
6vs0	0.97265	[936 31;22 949]	0.0005
7vs5	0.97344	[989 12;39 880]	0.0005
8vs7	0.97353	[947 26;27 1002]	0.0005
9vs2	0.97452	[993 36;16 996]	0.0005
7vs4	0.97463	[981 4;47 978]	0.0005
5vs0	0.97489	[864 19;28 961]	0.0005
8vs4	0.97751	[949 19;25 963]	0.0005
8vs0	0.97851	[938 6;36 974]	0.0005
2vs0	0.97913	[994 4;38 976]	0.0005
6vs4	0.98093	[949 28;9 954]	0.0005
2vs1	0.98154	[998 6;34 1129]	0.0005
5vs4	0.98506	[879 15;13 967]	0.0005
9vs6	0.98577	[982 1;27 957]	0.0005
6vs3	0.98628	[943 12;15 998]	0.0005
7vs6	0.98741	[1010 7;18 951]	0.0005
3vs1	0.98834	[990 5;20 1130]	0.0005
9vs0	0.98894	[994 7;15 973]	0.0005
5vs1	0.99013	[884 12;8 1123]	0.0005
6vs1	0.99188	[947 6;11 1129]	0.0005
7vs0	0.99253	[1022 9;6 971]	0.0005
9vs1	0.99254	[1002 9;7 1126]	0.0005
4vs0	0.99286	[978 10;4 970]	0.0005
4vs3	0.99347	[975 6;7 1004]	0.0005
4vs1	0.99386	[970 1;12 1134]	0.0005
3vs0	0.99447	[1006 7;4 973]	0.0005
1vs0	0.99858	[1135 3;0 977]	0.0005

Table 2: Best result table, sorted in accuracy ascending order, for each pair of different digits classification results using linear SVM, and 100 instances for training.

Classifications	Accuracy	Confusion Matrix	Best C value
0vs8	0.31781	[615 968;365 6]	0.0005
3vs9	0.36057	[723 1004;287 5]	0.0005
2vs3	0.41479	[359 522;673 488]	0.0005
3vs8	0.4254	[660 790;350 184]	0.0005
1vs5	0.44746	[25 10;1110 882]	0.0005
1vs4	0.45678	[1 16;1134 966]	0.0005
1vs6	0.45772	[0 0;1135 958]	0.0005
1vs8	0.46183	[0 0;1135 974]	0.0005
1vs3	0.4704	[0 1;1135 1009]	0.0005
1vs9	0.47108	[3 2;1132 1007]	0.0005
3vs7	0.47448	[962 1023;48 5]	0.0005
1vs7	0.47527	[0 0;1135 1028]	0.0005
1vs2	0.47623	[0 0;1135 1032]	0.0005
2vs9	0.50563	[1032 1009;0 0]	0.0005
3vs4	0.50703	[1010 982;0 0]	0.0005
3vs6	0.51321	[1010 958;0 0]	0.0005
2vs8	0.51446	[1032 974;0 0]	0.0005
0vs2	0.5159	[30 24;950 1008]	0.0005
2vs7	0.51602	[1032 997;0 31]	0.0005
6vs7	0.51762	[0 0;958 1028]	0.0005
2vs6	0.5201	[1032 955;0 3]	0.0005
4vs6	0.52887	[982 914;0 44]	0.0005
4vs5	0.54322	[137 11;845 881]	0.0005
6vs9	0.55669	[86 0;872 1009]	0.0005
0vs3	0.58392	[161 9;819 1001]	0.0005
2vs4	0.60129	[993 764;39 218]	0.0005
0vs9	0.60382	[235 43;745 966]	0.0005
2vs5	0.60551	[1031 758;1 134]	0.0005
0vs5	0.60791	[966 720;14 172]	0.0005
4vs7	0.61692	[217 5;765 1023]	0.0005
5vs8	0.62272	[887 699;5 275]	0.0005
0vs4	0.63354	[974 713;6 269]	0.0005
8vs9	0.6349	[931 681;43 328]	0.0005
7vs8	0.67433	[1017 641;11 333]	0.0005
4vs8	0.6907	[390 13;592 961]	0.0005
6vs8	0.70704	[410 18;548 956]	0.0005
7vs9	0.71183	[963 522;65 487]	0.0005
5vs7	0.79583	[511 11;381 1017]	0.0005
5vs6	0.79838	[884 365;8 593]	0.0005
0vs7	0.82321	[660 35;320 993]	0.0005
5vs9	0.8627	[648 17;244 992]	0.0005
0vs6	0.86326	[804 89;176 869]	0.0005
0vs1	0.86998	[738 33;242 1102]	0.0005
3vs5	0.88486	[842 51;168 841]	0.0005
4vs9	0.93119	[886 41;96 968]	0.0005

Table 3: Best result table, sorted in accuracy ascending order, for each pair of different digits classification results using linear SVM, and 500 instances for training.

Classifications	Accuracy	Confusion Matrix	Best C value	Gamma
2vs5	0.46362	[0 0;1032 892]	0.0007	10
5vs7	0.46458	[892 1028;0 0]	0.0007	10
3vs5	0.46898	[0 0;1010 892]	0.0007	10
5vs9	0.46923	[892 1009;0 0]	0.0007	10
4vs5	0.47599	[0 0;982 892]	0.0007	10
0vs5	0.4765	[0 0;980 892]	0.0007	10
5vs8	0.47803	[892 974;0 0]	0.0007	10
5vs6	0.48216	[892 958;0 0]	0.0007	10
3vs6	0.48679	[0 0;1010 958]	0.0007	10
6vs9	0.48704	[958 1009;0 0]	0.0007	10
3vs8	0.49093	[0 0;1010 974]	0.0007	10
3vs4	0.49297	[0 0;1010 982]	0.0007	10
4vs9	0.49322	[982 1009;0 0]	0.0007	10
0vs6	0.49432	[0 0;980 958]	0.0007	10
2vs9	0.49437	[0 0;1032 1009]	0.0007	10
6vs8	0.49586	[958 974;0 0]	0.0007	10
0vs8	0.49846	[0 0;980 974]	0.0007	10
2vs7	0.49903	[0 0;1032 1028]	0.0007	10
3vs9	0.49975	[0 0;1010 1009]	0.0007	10
0vs4	0.50051	[0 0;980 982]	0.0007	10
4vs8	0.50204	[982 974;0 0]	0.0007	10
3vs7	0.50442	[0 0;1010 1028]	0.0007	10
7vs9	0.50466	[1028 1009;0 0]	0.0007	10
2vs3	0.50539	[1032 1010;0 0]	0.0007	10
4vs6	0.50619	[982 958;0 0]	0.0007	10
0vs9	0.50729	[0 0;980 1009]	0.0007	10
0vs3	0.50754	[0 0;980 1010]	0.0007	10
8vs9	0.50883	[0 0;974 1009]	0.0007	10
4vs7	0.51144	[0 0;982 1028]	0.0007	10
0vs7	0.51195	[0 0;980 1028]	0.0007	10
2vs4	0.51241	[1032 982;0 0]	0.0007	10
0vs2	0.51292	[0 0;980 1032]	0.0007	10
7vs8	0.51349	[1028 974;0 0]	0.0007	10
2vs8	0.51446	[1032 974;0 0]	0.0007	10
6vs7	0.51762	[0 0;958 1028]	0.0007	10
2vs6	0.51859	[1032 958;0 0]	0.0007	10
1vs2	0.52377	[1135 1032;0 0]	0.0007	10
1vs7	0.52473	[1135 1028;0 0]	0.0007	10
1vs3	0.52914	[1135 1010;0 0]	0.0007	10
1vs9	0.52938	[1135 1009;0 0]	0.0007	10
1vs4	0.53614	[1135 982;0 0]	0.0007	10
0vs1	0.53664	[0 0;980 1135]	0.0007	10
1vs8	0.53817	[1135 974;0 0]	0.0007	10
1vs6	0.54228	[1135 958;0 0]	0.0007	10
1vs5	0.55994	[1135 892;0 0]	0.0007	10

Table 4: Best result table, sorted in accuracy ascending order, for each pair of different digits classification results using RBF SVM, and 100 instances for training.

From here on, we will evaluate how the SVM performs on one against all classification. We will take the mentioned number of instances for each of the classes and each time a different one will be the positive and the rest will be negative samples. Also the tested C values will be different as the previously proposed C values did not perform well with such unbalanced data, so I proposed the following list of C values, and I have used them all for each SVM run in order to decide the best C value in each case. List of common C values.

$$C = [0.0001, 0.001, 0.01, 0.1, 0.5, 1, 2.5, 5, 10, 50, 100, 1000, 10000, 100000]$$

Classifications	Accuracy	Confusion Matrix	Best C value
8	0.9495	[630 161;344 8865]	0.1
9	0.9521	[736 206;273 8785]	0.1
5	0.9639	[657 126;235 8982]	0.1
3	0.9702	[799 87;211 8903]	0.1
2	0.9717	[790 41;242 8927]	0.1
4	0.972	[833 131;149 8887]	0.1
7	0.9756	[866 82;162 8890]	0.1
6	0.9804	[854 92;104 8950]	0.1
0	0.9883	[903 40;77 8980]	0.1
1	0.9911	[1086 40;49 8825]	0.1

Table 5: One against all accuracy table, sorted in accuracy ascending order, using linear SVM, and 100 instances of each class for training, 100 positive instances and 900 negative instances. Using the common C values

	Positive samples classified in reality belong to the digits classes (%)									
Digit	0	1	2	3	4	5	6	7	8	9
0	95.76	0	0.74	0.32	0.11	1.17	0.85	0.11	0.85	0.11
1	0	96.45	0.44	0	0.09	0.27	0.18	1.51	0.8	0.27
2	0.24	0.12	95.07	2.05	0.12	0.24	0.72	1.32	0	0.12
3	0.23	0.23	3.16	90.18	0	3.27	0	1.35	1.02	0.56
4	0.1	0	2.49	0.31	86.41	0.52	3.84	2.18	1.04	3.11
5	2.43	0.26	0.38	4.73	0.38	83.91	2.81	0.26	4.6	0.26
6	2.33	0.42	2.22	0.63	1.16	1.69	90.27	0.21	0.95	0.11
7	0.11	0	2.43	1.48	0.32	2.32	0.11	91.35	0.63	1.27
8	0.88	1.39	5.94	2.28	1.77	5.69	1.14	0.51	79.65	0.76
9	0	0	1.91	0.74	9.34	1.17	0.32	7.01	1.38	78.13

Table 6: One against all positive predicted % for each class, using linear SVM, and 100 instances of each class for training, 100 positive instances and 900 negative instances. Using the common C values.

- Train a SVM with 500 random samples per class and a linear kernel (you can use different C soft margin parameters: 0.01, 0.1, 1, 2, 4). Report the accuracy you get on the test set (the rest of samples) and the best C value. Report the confusion matrix you get.

Classifications	Accuracy	Confusion Matrix	Best C value
8	0.9602	[703 127;271 8899]	0.1
9	0.9615	[731 107;278 8884]	0.01
5	0.9728	[714 94;178 9014]	0.1
3	0.974	[824 74;186 8916]	0.1
4	0.9784	[862 96;120 8922]	0.1
2	0.9791	[873 50;159 8918]	0.1
7	0.9829	[910 53;118 8919]	0.1
6	0.9856	[874 60;84 8982]	0.1
0	0.9913	[934 41;46 8979]	0.01
1	0.9938	[1099 26;36 8839]	0.1

Table 7: One against all accuracy table, sorted in accuracy ascending order, using linear SVM, and 500 instances of each class for training, 500 positive instances and 4500 negative instances. Using the common C values.

	Positive samples classified in reality belong to the digits classes (%)									
Digit	0	1	2	3	4	5	6	7	8	9
0	95.79	0	0.82	0.31	0.1	0.62	1.03	0.21	0.92	0.21
1	0	97.69	0.36	0.09	0.09	0.09	0.18	0.62	0.44	0.44
2	0.54	0.33	94.58	1.63	0.33	0	0.54	1.95	0	0.11
3	0.33	0.11	1.67	91.76	0.11	3.79	0	0.78	1.11	0.33
4	0.1	0	1.25	0	89.98	0.84	3.13	1.15	0.94	2.61
5	1.98	0.25	0.5	3.47	0.12	88.37	2.1	0.25	2.72	0.25
6	1.5	0.11	1.28	0.32	0.64	1.39	93.58	0	1.07	0.11
7	0.21	0	1.35	1.35	0.1	1.04	0	94.5	0.21	1.25
8	0.72	1.57	3.01	1.2	1.33	6.14	0.36	0.24	84.7	0.72
9	0	0	0.36	0.24	5.49	0.6	0	5.13	0.95	87.23

Table 8: One against all positive predicted % for each class, using linear SVM, and 500 instances of each class for training, 500 positive instances and 4500 negative instances. Using the common C values.

- Train a SVM with 100 random samples per class and a RBF kernel (you can use different C soft margin parameters and different gamma values). Report the accuracy you get on the test set (the rest of samples) and the best C value. Report the confusion matrix you get.

Classifications	Accuracy	Confusion Matrix	Best C value	Gamma
9 against ALL	0.974	[811 62;198 8929]	100	0.01
8 against ALL	0.9759	[800 67;174 8959]	100	0.01
3 against ALL	0.9795	[873 68;137 8922]	1000	0.01
5 against ALL	0.9798	[745 55;147 9053]	1000	0.01
2 against ALL	0.9812	[888 44;144 8924]	1000	0.01
4 against ALL	0.9816	[848 50;134 8968]	5	0.01
7 against ALL	0.9823	[911 60;117 8912]	5	0.01
6 against ALL	0.9881	[873 34;85 9008]	100	0.01
0 against ALL	0.9928	[955 47;25 8973]	1000	0.01
1 against ALL	0.9944	[1105 26;30 8839]	5	0.01

Table 9: One against all accuracy table, sorted in accuracy ascending order, using rbf SVM, and 100 instances of each class for training, 100 positive instances and 900 negative instances. Using the common C values and this gamma values: $\gamma = [0.01, 0.1, 1, 10, 100]$.

	Positive samples classified in reality belong to the digits classes (%)									
Digit	0	1	2	3	4	5	6	7	8	9
0	95.31	0	0.5	0.2	0.1	1.2	1.8	0.1	0.6	0.2
1	0	97.7	0.18	0	0.09	0.27	0.27	0.71	0.27	0.53
2	0.11	0.21	95.28	1.39	0.32	0.11	0.43	1.72	0.43	0
3	0.11	0.11	1.81	92.77	0	3.61	0.11	0.21	0.74	0.53
4	0	0	0.67	0	94.43	0.89	0.56	1.11	0.45	1.89
5	1.38	0.25	0	1.38	0.25	93.13	1	0	2.5	0.13
6	0.88	0.44	0.99	0.11	0.44	0.22	96.25	0	0.55	0.11
7	0.1	0	1.34	1.24	0.1	0.93	0	93.82	0.51	1.96
8	0.35	0.23	2.65	1.38	0.35	2.19	0.35	0	92.27	0.23
9	0	0	0.57	0.23	3.44	0.34	0	1.95	0.57	92.9

Table 10: One against all positive predicted % for each class, using rbf SVM, and 100 instances of each class for training, 100 positive instances and 900 negative instances. Using the common C values and this gamma values: $\gamma = [0.01, 0.1, 1, 10, 100]$.

From table 9 we can deduce that the class with more difficulties for classifying was class digit 9, which has the lowest best accuracy for training with different C and γ values.

From table 10 we can appreciate with which classes each one is getting confused and is giving false-positives. This can't completely reflect the accuracy results. Because this table is ignoring the recall, just the precision is being expressed. Due that we're only taking into account the positive retrieved elements as the total for the %s represented. So could be the case that a class is giving 100% of the correct class, but is just retrieving a 2% of the total real positive samples.

- There are some classes that are not obviously separated: 3 vs 5, 4 vs 9, 5 vs 8 and 7 vs 9. But keep in mind, this is just a PCA to two dimensions. Is this set of classes the classes the one with higher values in the confusion matrices you have reported?

  Yes it is, in order to answer better to this question some of the previous tables shown were sorted in accuracy order, and we can observe that the first instances appearing are the ones mentioned on this question, the order may vary depending on the SVM used (linear or rbf) and also depending on the number of instances used for the training (100 or 500), but it agrees on the worst performance results with the PCA plot shown in this exercise.

The only thing to point out is that it has a really high accuracy anyway, so perhaps are the worst scenarios for classification, but not as bad when all available dimensions without reduction, are considered.

2 Poselets

Report an image you have selected with all voting hypothesis drawn on the image.

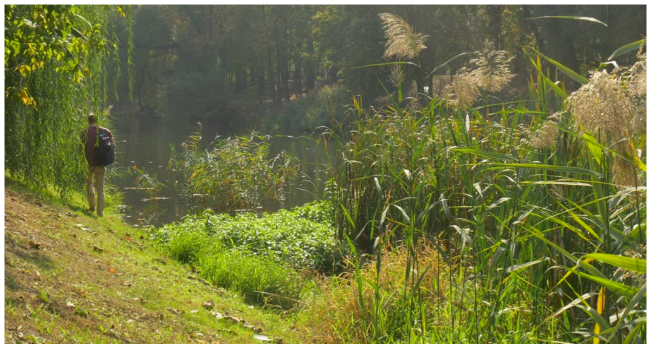

Figure 1: Original image used for the poselets test.

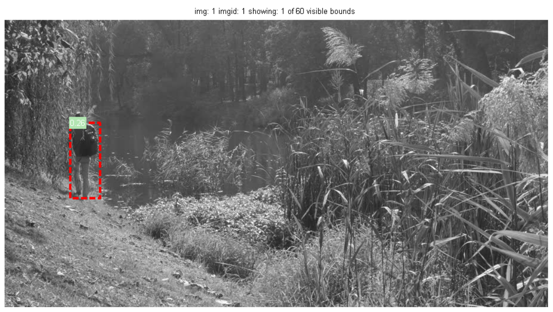

Figure 2: Part that after applying poselets is considered to contain a person with maximum voting rate, and that actualy dose contain a person.

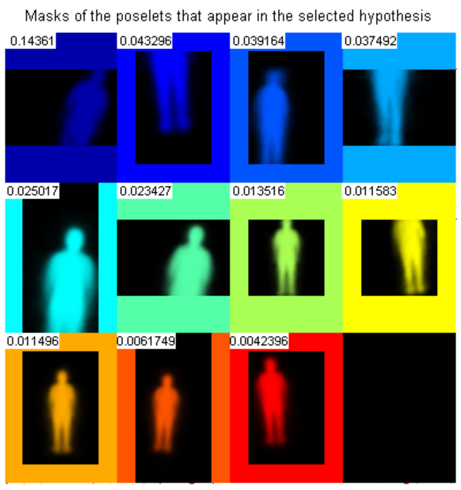

Figure 3: Poselets hypothesis generated that support the previous selection to contain a person.

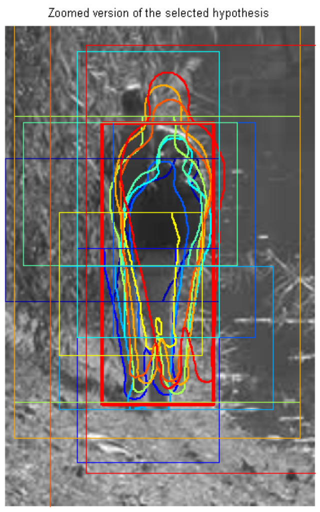

Figure 4: Poselets hypothesis voting on the selection supported as candidate to contain a person.

3 Image Classification

3.1 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?

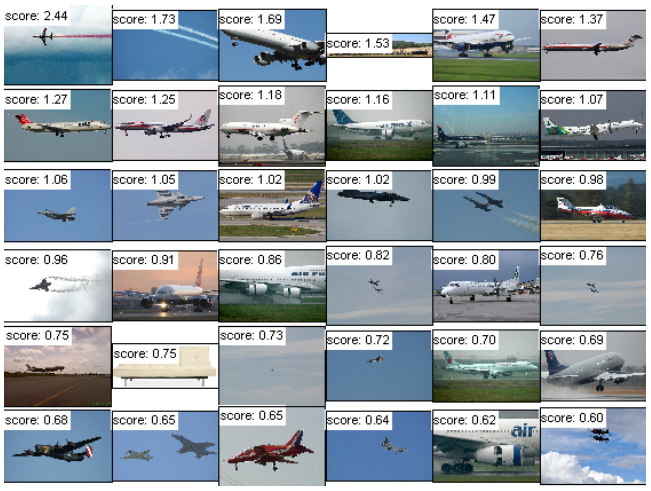

Figure 5: Ranking obtained using the bias parameter.

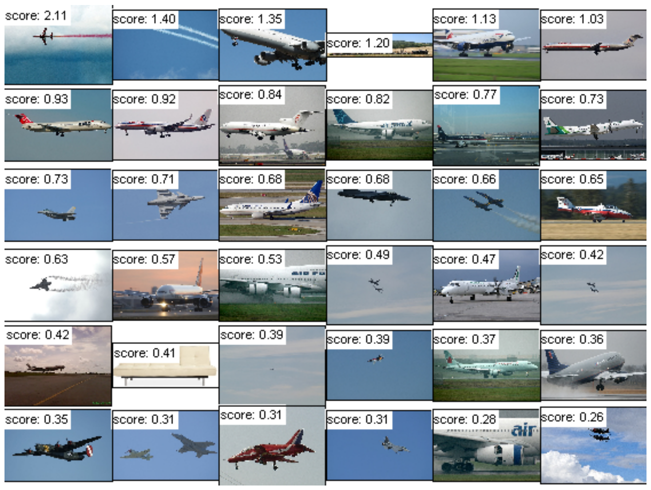

Figure 6: Ranking obtained without using the bias parameter.

As we can appreciate, removing the bias from the formula $testScores = w{*}testHistograms{+}bias$, and having just $testScores = w'testHistograms$, will not affect the obtained ranking. The reason for this is due to the dimensionality, because the value of the bias is not significant for a separating hyperplane with 512 dimensions. It changes the scores of the results but not the best results, because are still the most reliably classified samples.

So we can say that as the dimensionality increases the bias relevance decreases, in fact is well known that in high dimensional spaces there are always crossing coordinates origin hyperplanes that can separate all the data we may represent represented.

But the bias would be relevant in a low dimensionally space if in order to demonstrate this we ran again changing the bias values to $bias = 10^5 0bias$, and as can be seen in the ranking retrieved, the bias becomes important due to its magnitude difference in comparison to the classification vector w.

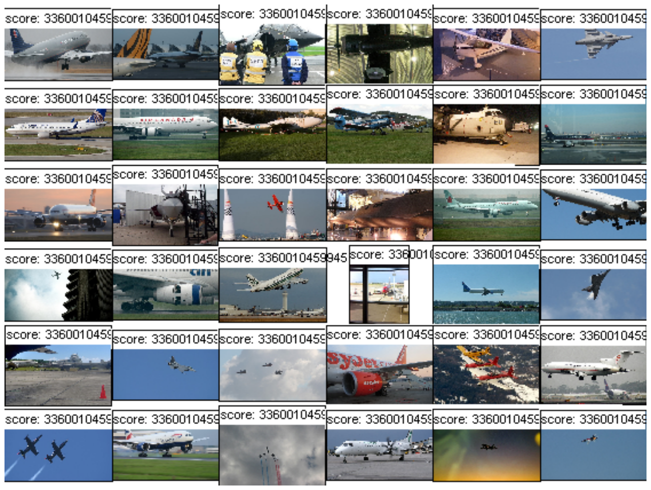

Figure 7: Ranking obtained with $bias = 10^5 0bias$.

The w values are within the range $[-2.9728, 2.6100]$, and the bias term became (after the modification we did) $bias = 3.3600e + 49$. So it affects to the ranking output, although, I was still surprised by the fact that all images retrieved are still plane images, which I didn't expect with such a order increase of the bias.

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

Rerun exercise 1, for the other classes of images. *Does the AP performance match your expectations based on the variation of the class images?* **AP for planes** Test AP: 0.49 Correctly retrieved in the top 36: 18 **AP for motorbikes** Test AP: 0.52 Correctly retrieved in the top 36: 24 **AP for people** Test AP: 0.72 Correctly retrieved in the top 36: 32

It does for planes and motorbikes, but I expected people to have a worse AP, it actually retrieves images in the ranking that do not contain people, when planes only shows planes in the ranking, but at the end gets a worse AP rate.

3.3 Stage E: Vary the image representation

Make sure you understand the reason for the change in performance. **AP for planes** Test AP: 0.48 Correctly retrieved in the top 36: 19 **AP for motorbikes** Test AP: 0.45 Correctly retrieved in the top 36: 24 **AP for people** Test AP: 0.70 Correctly retrieved in the top 36: 29

When we set up the representation to a "bag of visual words" we can observe that all types perform worse as was expected in general, because we are in fact reducing the information provided for classifying.

As we expected after looking at the images set, the aeroplane is the class that maintains better its performance, this fact is due to that aeroplanes can fly, so aeroplanes features can appear at any part on an image, sky, landing, on the airport, etc. So aeroplane class is the less affected when the position information is lost.

On the other hand, we might have that most motorbikes in images are in the ground area from the image, with exceptions, but it's highly bounded to position, rather than aeroplanes. The same happens for people as for motorbikes.

3.4 Stage F: Vary the classifier

In this stage we will use only the aeroplane class for answering to the exercise questions.

- *Edit exercise1.m so that the square root of the histograms is used for the feature vectors. Note, this involves writing a line of Matlab code for the training and test histograms.*

```
histograms = sign(histograms).*sqrt(abs(histograms)) ;
testHistograms = sign(testHistograms).*sqrt(abs(testHistograms)) ;
```

- *Retrain the classifier for the aeroplane class, and measure its performance on the test data.*

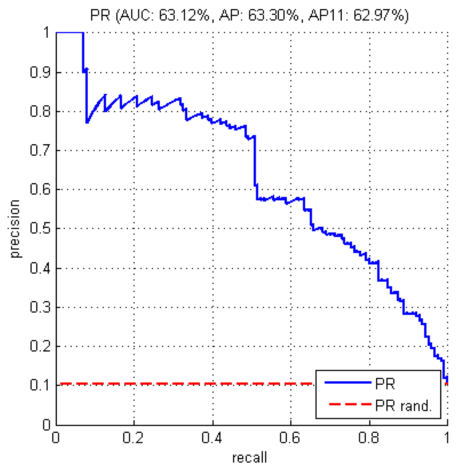

Figure 8: Test AP: 0.70 — Correctly retrieved in the top 36: 32

- *Why is this procedure equivalent to using the Hellinger kernel?* Because

  we have calculated the feature map as it is done for the Hellinger kernel. So we have projected our data as Hellinger kernel does and then separate it with a linear classifier.

- *Why is it an advantage to keep the classifier linear, rather than using a non-linear kernel?* Since we are using a non-linear representation, such as

  histograms with high dimensionality data, we know that our problem can be solved by a lineal classifier.

- *Try removing the L2 normalization step. Does this affect the performance? Why? (Hint: the histogram are L1 normalized by construction)*

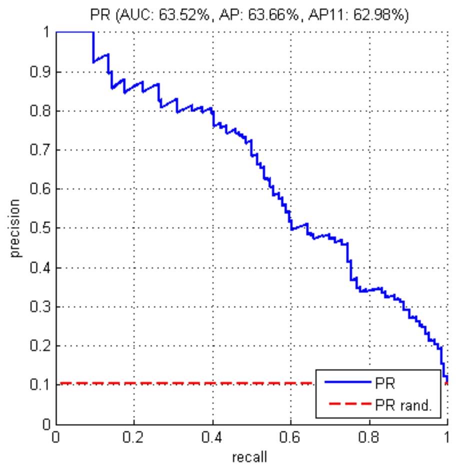

Figure 9: Test AP: 0.67 — Correctly retrieved in the top 36: 31

As we can appreciate the performance improves by just maintaining the L1 normalitzation implicit in the hystogram construction. This is due to the fact that histogram information is more representative without normalitzation. This can be clearly seen in the training step rather than the testing, where we can perceive that for we receive maximum performance when we remove the L2 code. And this fact does not lead us to an overfitted model, because we still have good performance results in the testing.

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

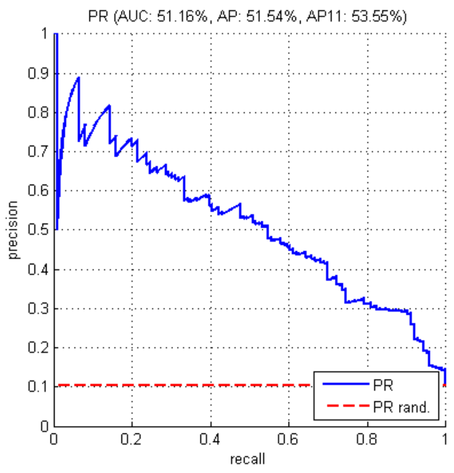

Figure 10: Test AP: 0.56 — Correctly retrieved in the top 36: 27

We have the same case than with the Hellinger kernel, we obtain better results when L2 normalitzation step is not performed.

3.5 Stage G: Vary the number of training images

For this section I will use the aeroplane class in for the report questions.

Up to this point we have used all the available training images. Now edit exercise1.m the fraction variable to use 10% and 50% of the training data.

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

Class	Train Data %	Kernel	C	PA
aeroplane	100	linear	10	0.55571
aeroplane	100	Hellinger	10	0.66653
aeroplane	10	linear	10	0.62366
aeroplane	10	Hellinger	10	0.53282
aeroplane	50	linear	10	0.51159
aeroplane	50	Hellinger	10	0.63521
motorbike	100	linear	10	0.46034
motorbike	100	Hellinger	10	0.58247
motorbike	10	linear	10	0.25675
motorbike	10	Hellinger	10	0.2935
motorbike	50	linear	10	0.40201
motorbike	50	Hellinger	10	0.47356
person	100	linear	10	0.69504
person	100	Hellinger	10	0.71712
person	10	linear	10	0.62983
person	10	Hellinger	10	0.63864
person	50	linear	10	0.66363
person	50	Hellinger	10	0.70755

Table 11: Table of Precision Average results.

In this table we can see the different PA values obtained when playing with the training set size.

- *Do you think the performance has 'saturated' if all the training images are used, or would adding more training images give an improvement?*
  - For aeroplane class we can observe that the SVM with linear kernel is performing better with just 10% and when we increase the size of the training set, the performance decreases, so the model is overfitted with more than 10% of training data used. On the other hand we see than for the Hellinger SVM performance continues increasing as we use more data for training.
  - For the motorbike class we can see than a 10% is to few data, and the model is underfitted, we know this due to that we end up doubling the performance when we use the whole training dataset.
  - For the person class is the only really clear case of saturation when we use the 50% of the training data. We can observe that even that it improves when we use more instances for training (so it's not overffited yet), the improvements in relation with the training size increases are not important. This situation is more evident when using Hellinger, that the change from 50% of data to 100% just improves in 1% the resulting performance.

4 Links and further work

- If there is a significant difference between the training and test performance, then that indicates over fitting. The difference can often be reduced, and the test performance (generalisation) improved by changing the SVM C parameter. *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.*

Class	Train Data %	Kernel	C	PA
aeroplane	100	linear	0.1	0.46167
aeroplane	100	linear	1	0.63347
aeroplane	100	linear	10	0.55571
aeroplane	100	linear	100	0.66003
aeroplane	100	linear	1000	0.67158
aeroplane	100	Hellinger	0.1	0.70657
aeroplane	100	Hellinger	1	0.70121
aeroplane	100	Hellinger	10	0.66653
aeroplane	100	Hellinger	100	0.66653
aeroplane	100	Hellinger	1000	0.66653
aeroplane	10	linear	0.1	0.36601
aeroplane	10	linear	1	0.46595
aeroplane	10	linear	10	0.62366
aeroplane	10	linear	100	0.53195
aeroplane	10	linear	1000	0.50431
aeroplane	10	Hellinger	0.1	0.62149
aeroplane	10	Hellinger	1	0.54364
aeroplane	10	Hellinger	10	0.53282
aeroplane	10	Hellinger	100	0.53282
aeroplane	10	Hellinger	1000	0.53282
aeroplane	50	linear	0.1	0.34512
aeroplane	50	linear	1	0.41159
aeroplane	50	linear	10	0.51159
aeroplane	50	linear	100	0.57481
aeroplane	50	linear	1000	0.57693
aeroplane	50	Hellinger	0.1	0.54074
aeroplane	50	Hellinger	1	0.60681
aeroplane	50	Hellinger	10	0.63521
aeroplane	50	Hellinger	100	0.63521
aeroplane	50	Hellinger	1000	0.63521

Table 12: Table of Precision Average results for different C values, going from 0.1 to 1000.

motorbike	100	linear	0.1	0.23188
motorbike	100	linear	1	0.34011
motorbike	100	linear	10	0.46034
motorbike	100	linear	100	0.48061
motorbike	100	linear	1000	0.48061
motorbike	100	Hellinger	0.1	0.33797
motorbike	100	Hellinger	1	0.59147
motorbike	100	Hellinger	10	0.58247
motorbike	100	Hellinger	100	0.58247
motorbike	100	Hellinger	1000	0.58247
motorbike	10	linear	0.1	0.16556
motorbike	10	linear	1	0.18391
motorbike	10	linear	10	0.25675
motorbike	10	linear	100	0.25617
motorbike	10	linear	1000	0.25617
motorbike	10	Hellinger	0.1	0.15659
motorbike	10	Hellinger	1	0.27503
motorbike	10	Hellinger	10	0.2935
motorbike	10	Hellinger	100	0.2935
motorbike	10	Hellinger	1000	0.2935
motorbike	50	linear	0.1	0.1878
motorbike	50	linear	1	0.34436
motorbike	50	linear	10	0.40201
motorbike	50	linear	100	0.39512
motorbike	50	linear	1000	0.39512
motorbike	50	Hellinger	0.1	0.41432
motorbike	50	Hellinger	1	0.48462
motorbike	50	Hellinger	10	0.47356
motorbike	50	Hellinger	100	0.47356
motorbike	50	Hellinger	1000	0.47356
person	100	linear	0.1	0.63068
person	100	linear	1	0.66784
person	100	linear	10	0.69504
person	100	linear	100	0.69038
person	100	linear	1000	0.67865
person	100	Hellinger	0.1	0.73322
person	100	Hellinger	1	0.71153
person	100	Hellinger	10	0.71712
person	100	Hellinger	100	0.70467
person	100	Hellinger	1000	0.70467

Table 13: Table of Precision Average results for different C values, going from 0.1 to 1000.

person	10	linear	0.1	0.53607
person	10	linear	1	0.58788
person	10	linear	10	0.62983
person	10	linear	100	0.61832
person	10	linear	1000	0.61832
person	10	Hellinger	0.1	0.61535
person	10	Hellinger	1	0.64446
person	10	Hellinger	10	0.63864
person	10	Hellinger	100	0.63864
person	10	Hellinger	1000	0.63864
person	50	linear	0.1	0.53889
person	50	linear	1	0.59296
person	50	linear	10	0.66363
person	50	linear	100	0.66712
person	50	linear	1000	0.66625
person	50	Hellinger	0.1	0.60964
person	50	Hellinger	1	0.73796
person	50	Hellinger	10	0.70755
person	50	Hellinger	100	0.70557
person	50	Hellinger	1000	0.70557

Table 14: Table of Precision Average results for different C values, going from 0.1 to 1000.

From this table we can see how the C value is affecting each class, for linear and non-linear SVM, and the resulting performance obtained from different C values when we vary the training set size.