

Pattern Project Report

-Ankit Nayan
-Ankita Balotia
-Pritesh Kumar

Opencv implementation of BOW model for classification of objects

Steps in Bow model :

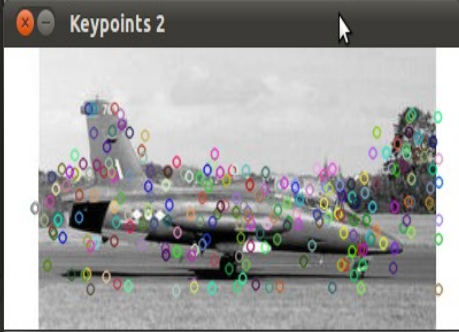
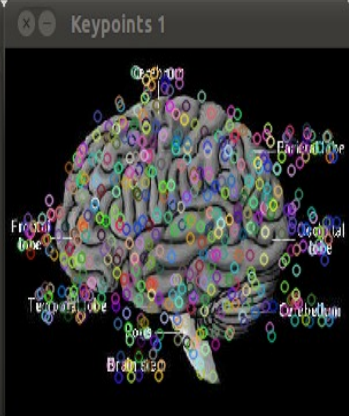
1. Calculate SURF keypoints (a fast implementation of SIFT) for all training images
2. Cluster all the keypoints by K-means clustering
3. Make a Dictionary of those codewords (final cluster centres)
4. Match the keypoints of an image to codewords by Nearest Neighbour match
5. Calculate descriptors of the images by making histogram of the keypoints ,ie, frequency of keypoints of an image in different codewords.
6. Normalized histogram is written in csv file
7. Classify the objects by classifiers such as Naive Bayes, SMO, Logistic using **Weka**

- Keypoints are basically the points of extrema
- Keypoints are extracted with SURF algorithm with $\text{minHessian} = 400$
- SURF finds keypoints by finding laplacian, approximated using difference of gaussians
- SURF has feature descriptor (of each keypoint) of size 64 that contains gradients in $dx, |dx|, dy, |dy|$ for 4x4 subregions each of size 5x5 samples
- SURF descriptors are scale and rotation invariant

Some surf keypoints

Keypoints 2


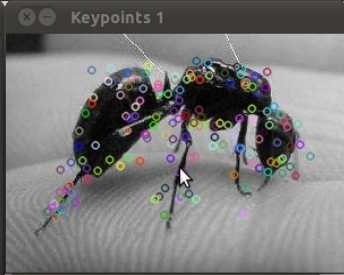
```
ankit@ubuntu: ~/Desktop/Desktop/surf_view_keypoints
-- Good Match [16] Keypoint 1: 227 -- Keypoint 2: 34
-- Good Match [17] Keypoint 1: 233 -- Keypoint 2: 93
-- Good Match [18] Keypoint 1: 253 -- Keypoint 2: 127
-- Good Match [19] Keypoint 1: 285 -- Keypoint 2: 4
-- Good Match [20] Keypoint 1: 294 -- Keypoint 2: 38
-- Good Match [21] Keypoint 1: 299 -- Keypoint 2: 147
-- Good Match [22] Keypoint 1: 311 -- Keypoint 2: 127
-- Good Match [23] Keypoint 1: 315 -- Keypoint 2: 141
-- Good Match [24] Keypoint 1: 316 -- Keypoint 2: 34
-- Good Match [25] Keypoint 1: 337 -- Keypoint 2: 52
-- Good Match [26] Keypoint 1: 373 -- Keypoint 2: 133
-- Good Match [27] Keypoint 1: 396 -- Keypoint 2: 80
-- Good Match [28] Keypoint 1: 420 -- Keypoint 2: 185
-- Good Match [29] Keypoint 1: 421 -- Keypoint 2: 37
-- Good Match [30] Keypoint 1: 463 -- Keypoint 2: 34
-- Good Match [31] Keypoint 1: 467 -- Keypoint 2: 174
-- Good Match [32] Keypoint 1: 470 -- Keypoint 2: 217
-- Average distances in Good Matches : 0.418195
ankit@ubuntu:~/Desktop/Desktop/surf_view_keypoints$ ./flann_keypoints brain.jpg
aero.jpg
keypoints_1=472 keypoints_2=222
-- Max dist : 0.803001
-- Min dist : 0.302505
```



flann_keypoints

Keypoints 2

```
ankit@ubuntu: ~/Desktop/Desktop/surf_view_keypoints
-- Average distances in Good Matches : 0.418195
ankit@ubuntu:~/Desktop/Desktop/surf_view_keypoints$ ./flann_keypoints brain.jpg
aero.jpg
keypoints_1=472 keypoints_2=222
-- Max dist : 0.803001
-- Min dist : 0.302505
^C
ankit@ubuntu:~/Desktop/Desktop/surf_view_keypoints$ ./flann_keypoints ant.jpg bo
nsai.jpg
keypoints_1=160 keypoints_2=359
-- Max dist : 0.730755
-- Min dist : 0.197613
-- Good Match [0] Keypoint 1: 58 -- Keypoint 2: 244
-- Good Match [1] Keypoint 1: 62 -- Keypoint 2: 143
-- Good Match [2] Keypoint 1: 70 -- Keypoint 2: 216
-- Good Match [3] Keypoint 1: 71 -- Keypoint 2: 179
-- Good Match [4] Keypoint 1: 95 -- Keypoint 2: 102
-- Good Match [5] Keypoint 1: 103 -- Keypoint 2: 304
-- Good Match [6] Keypoint 1: 109 -- Keypoint 2: 273
-- Good Match [7] Keypoint 1: 118 -- Keypoint 2: 102
-- Good Match [8] Keypoint 1: 139 -- Keypoint 2: 216
-- Good Match [9] Keypoint 1: 143 -- Keypoint 2: 202
-- Average distances in Good Matches : 0.242301
```



flann_keypoints

11 items, Free space: 19.4 GB

Simple image matching is done by calculating the distance between each pair of keypoints descriptors and then applying the formula :

$$d1/d2 < 0.7$$

that is, ratio of nearest to 2nd nearest neighbour should be less than 0.7

- if match is not confident then $d1/d2 \sim 1$

The screenshot shows a desktop environment with a file manager window titled 'Good Matches' and a terminal window. The file manager window displays two images of an Adidas t-shirt. The left image is a reference image with keypoints marked as small circles. The right image is a test image with the same keypoints. Colored lines connect corresponding keypoints between the two images, demonstrating feature matching. The terminal window shows the output of a matching program, including Hough transform results for angle and scale, and the final match counts.

```
Angle transformation
6 9 4 5 6 8 48 45 4 3 3
2 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
max votes in angle axis is in index 6 count_angle=48
Most Voted Angle Diff 0
*****
Scale transformation
0 0 0 0 0 0 0 0 5 3
2 1 1 0 0 0 0 0 0 0 0
max votes in scale axis is in index 10 count_scale=32
Most Voted Scale Diff 0
total_matches = 72
after_angle_matches = 39
after_scale(view)_matches = 39
```

Good Matches

Back Good Matches

Places

- ankit
- Desktop
- File System
- Network
- System
- 63 GB
- Stuff
- Trash
- Documents
- Music
- Pictures
- Videos
- Downloads

ob23.jpg README_hough.txt

```

total_matches = 120
after_angle matches = 34
after_scale(view)_matches = 33
  
```

42 items, Free space: 19.9 GB

Good Matches

Back Good Matches

Places

- ankit
- Desktop
- File System
- Network
- System
- 63 GB
- Stuff
- Trash
- Documents
- Music
- Pictures
- Videos
- Downloads

© AX200 MMX

virgin america

virgin america

Scale 1:200

Scale 1:200

virgin america WINGS

Matcher.cpp nokia_logo.jpg nokia_lo

```

Scale transformation
0 0 0 0 0 0 0 3 3 9 6 7
0 0 0 0 0 0 0 0 0 0 0 0
ax votes in scale axis is in index 8 count_scale=9
Most Voted Scale Diff -40
total_matches = 102
after_angle matches = 28
after_scale(view)_matches = 15
  
```

ob23.jpg README_hough.txt

42 items, Free space: 19.9 GB

Results

Tested on 4 classes : nautilus, dolphin, crab & garfield each with 50 images
Results depend on vocabulary size here 100 and 200 are chosen.

K in K-means clustering equals vocabulary size.

- | | Vocab size = 100 | Vocab size = 200 |
|-----------------|------------------|------------------|
| • SVM : | 76% | 79.9% |
| • Naive Bayes : | 76.6% | 75.5% |
| • Logistic : | 69% | 70.1% |

vocabulary size = 200

svm : 79.9%

==== Confusion Matrix ====

```
a b c d <-- classified as
42 1 6 1 | a = nautilus
5 35 10 0 | b = dolphin
4 5 41 0 | c = crab
1 1 3 29 | d = garfield
```

naïve bayes : 75.5%

==== Confusion Matrix =====

```
a b c d <-- classified as
38 5 4 3 | a = nautilus
8 34 8 0 | b = dolphin
3 5 40 2 | c = crab
1 1 5 27 | d = garfield
```

logistic : 70.1%

==== Confusion Matrix ====

```
a b c d <-- classified as
36 7 7 0 | a = nautilus
12 24 14 0 | b = dolphin
4 4 41 1 | c = crab
1 2 3 28 | d = garfield
```

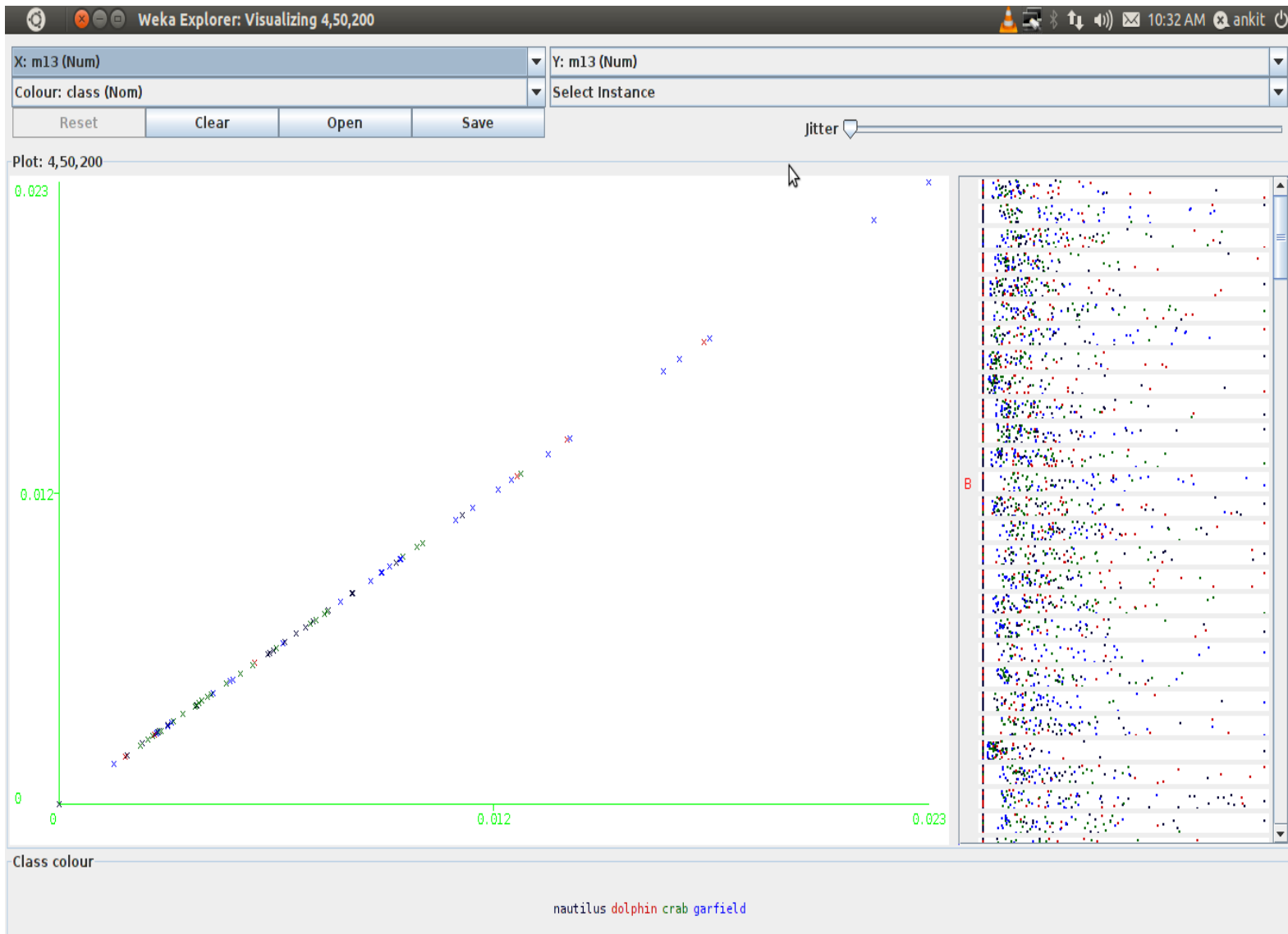
For 4 classes 25% accuracy is equivalent to no result since even if the objects were classified randomly there is $\frac{1}{4}$ probability of it being correct.

For the case of 4 classes the best two attributes that were found are m13 and m17

- m13 – 13th codeword in the dictionary of size 200
- m17 – 17th codeword in the dictionary of size 200

The figures below helps us visualize the distribution of 4 classes over the values of m13 and m17.

Not visible to human eyes but classifier models this distinctness of values for different classes to classify.

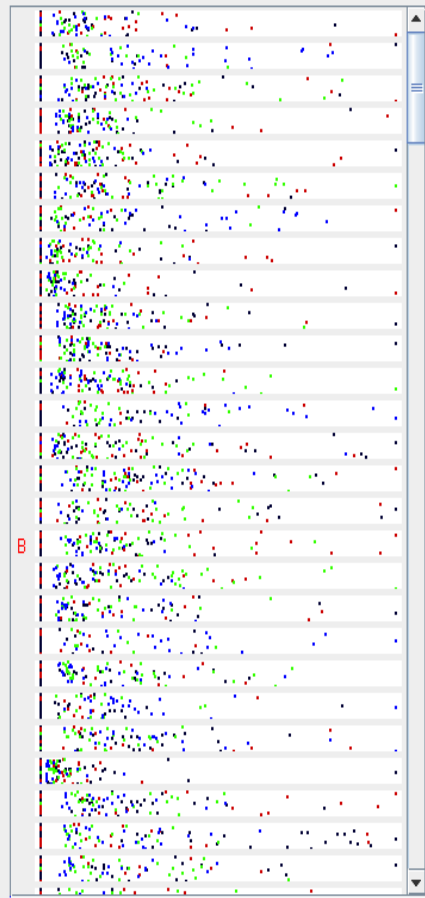
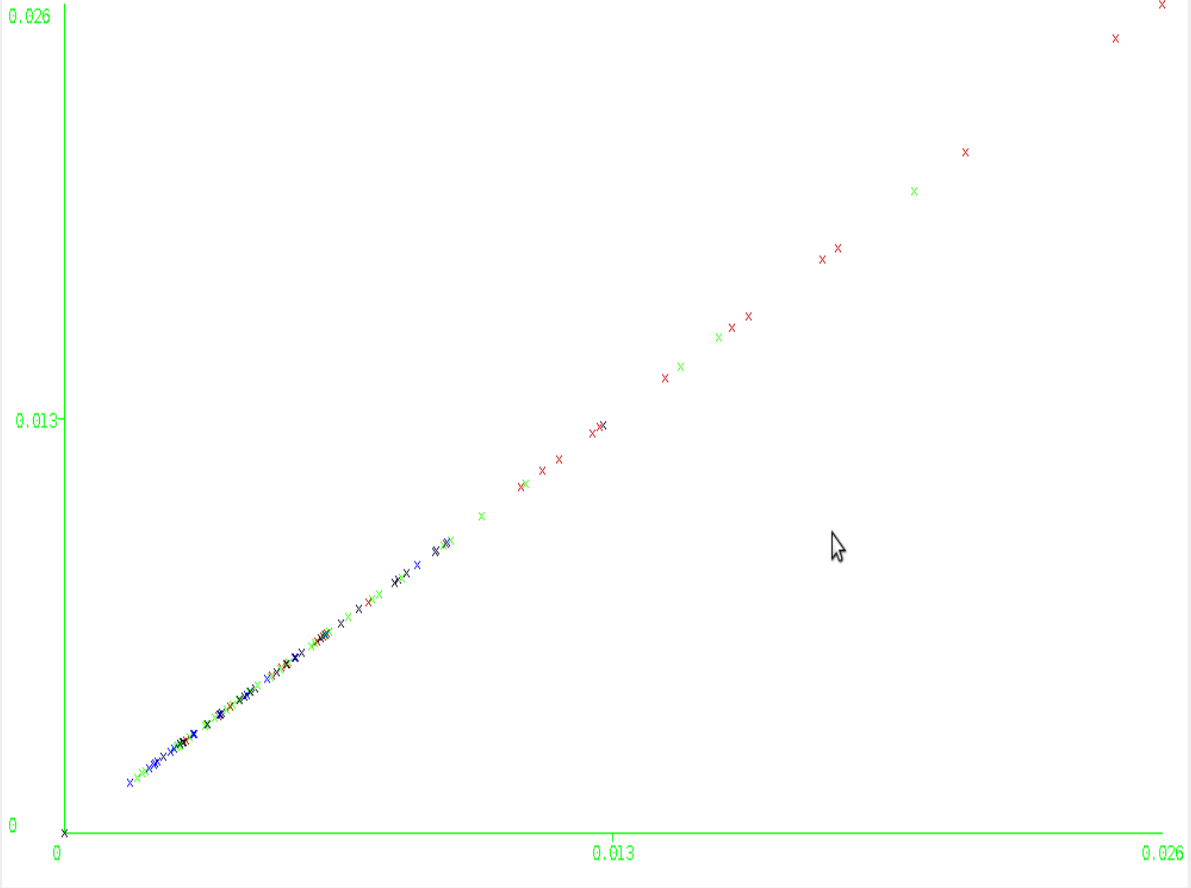


X: m17 (Num) Y: m17 (Num)
Colour: class (Nom) Select Instance

Reset Clear Open Save

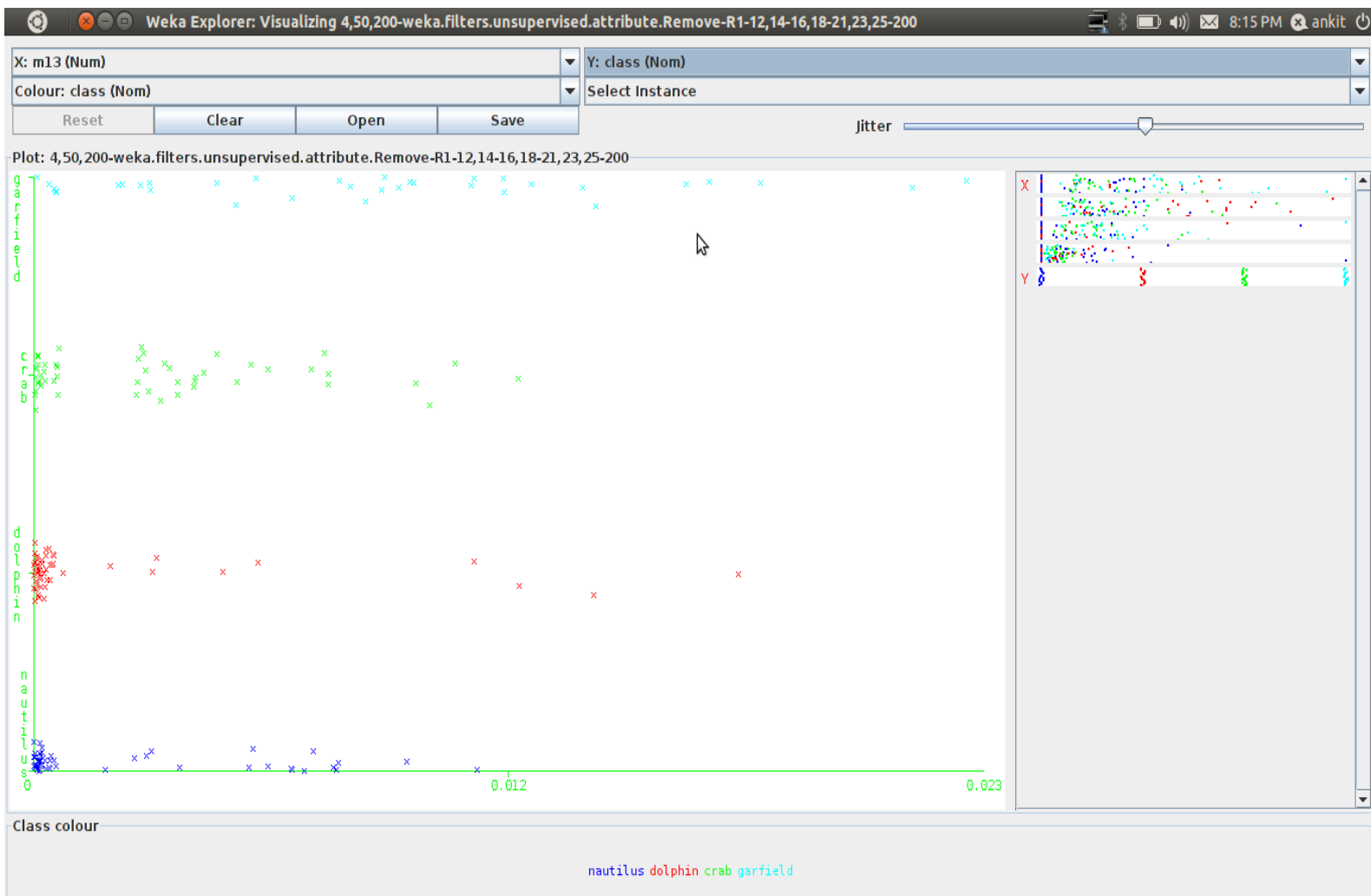
jitter

Plot: 4,50,200



Class colour

nautilus	dolphin	crab	garfield
----------	---------	------	----------



When tested on 5 classes (aeroplane, ant, brain, nautilus, garfield) each with 50 images and vocab size = 200, SVM gave accuracy of 70.9%

for 10 classes ~60% accuracy

BOW limitations :

- It models only the basic appearance of the object in an image. Needs to be implemented with pyramid structure.
- It does not take into account shape of the object which alone is not efficient but when combined with appearance proves useful

THANK YOU