Machine Learning for Visual Data Analysis – IMAGE CLASSIFICATION USING BOW

Description

Background:

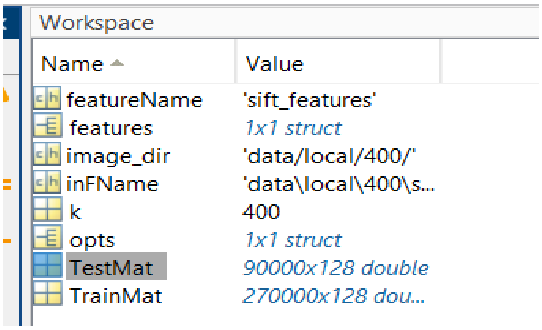
The term "bag of visual words" (BOVW) is frequently used in image classification. Its idea is inspired by information retrieval and NLP's bag of words (BOW). We think of a document as a collection of words (BOW). In bag of visual words (BOVW), we have the same concept, but instead of words, we use image features as the "words." Image features are distinct patterns that can be found in images.

Dictionary creation-feature quantization:

Section 2.1

When the given code section 2.1 is executed successfully, variables TrainMat(270000 X 128) and TestMat((90000X128) are created and loaded.

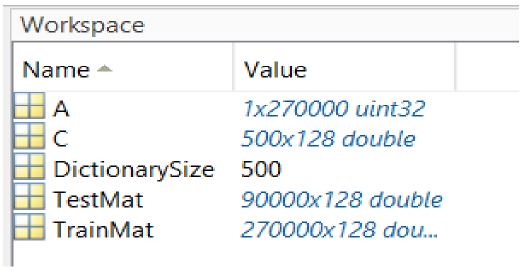
```
%{%
load('data/global/all_features');
```



Section 2.2

After running the code in section 2.2. We can observer that a dictionary of 500 words is generated successfully. The dictionary's size id determined by the number of clusters. The following observed dictionary is saved in dictionary.mat as C(500x128).

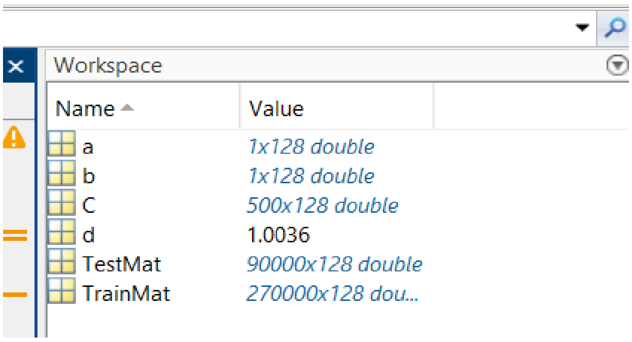
```
Command Window
Elapsed time is 110.381251 seconds.
```



Section 2.3

In this section, the distance between image descriptors and codewords(single codeword is taken from 'C') is computed by using the code definition in file EuclideanDistance.m. From the figure below it can be stated as follows:

'a'	single sample form TestMat
'b'	single cluster centre from the dictionary of cluster centres
'C'	single codeword
'd'	Euclidean distance between the given sample and the cluster centre.



Section 2.4

With the help of below provided code each descriptor in the training and test images is assigned to the closest codeword cluster. In the workspace it can be observed that an index_train(1x270000) and an index_test(1x90000) is created which contains indices of assigned codewords.

Code

```
%% 2.4 Assign each descriptor to the nearest codeword

%{%
clear;
load('data/global/all_features');
load('data/global/dictionary');

% The following 3 lines is an example to on how to assign the descriptor
discrptor_test1 to the nearest codeword in C
discrptor_test1 = TestMat(1,:);
d = EuclideanDistance(discrptor_test1,C);
[minv,index] = min(d);% index will be the nearest codeword cluster

%--------------Write Your Own Code here that assigns all descriptors -------------
%Jahnvi's code____210538601
index_train = zeros(size(TrainMat,1),1); % initializing size of training samples
index_test = zeros(size(TestMat,1),1); % initializing size of test samples

for k=1:size(TrainMat,1) % iterating over and computing for each training sample
    [minv,index] = min(EuclideanDistance(TrainMat(k,:),C)); % computing codeword
for each training sample
    index_train(k,1) = index; % store the index only for the nearest code word
end

for k=1:size(TestMat,1) % iterate over and compute for each test sample
    [minv,index] = min(EuclideanDistance(TestMat(k,:),C)); % compute codeword for
each test sample
    index_test(k,1) = index;
end
%--------------Write Your Own Code here that assigns all descriptors -------------
%--------------Write Your Own Code here that assigns all descriptors -------------
save('data/global/assignd_discriptor','index_train','index_test');
%}
%----------------------------end of code---------------------------------
```

Workspace output:

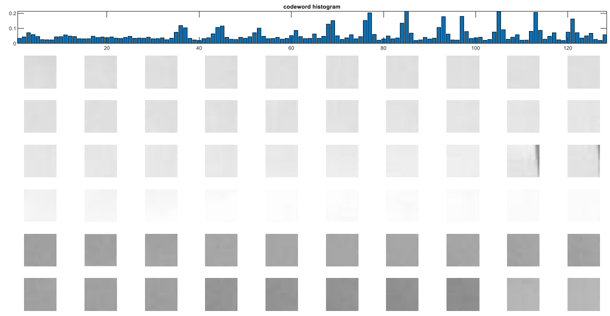
Workspace

Name	Value
C	500x128 double
d	1x500 double
discrptor_test1	1x128 double
index	158
index_test	90000x1 double
index_train	270000x1 double
k	90000
minv	0.5444
TestMat	90000x128 double
TrainMat	270000x128 dou...

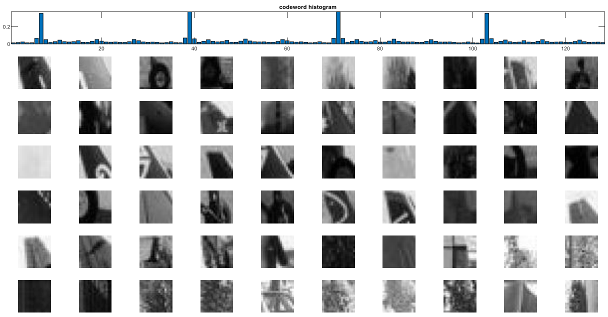
Section 2.5

The patch visualisations for the mentioned inputs(78,37,397) are listed below:

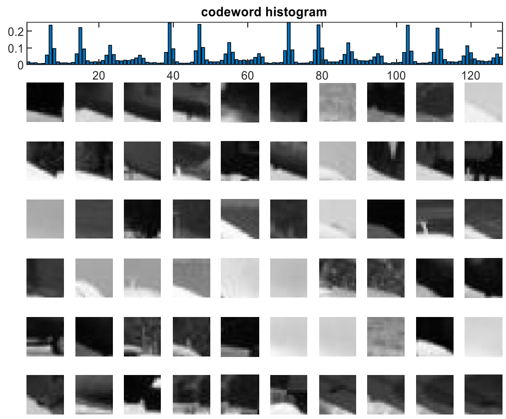
Outputs:



Output of ‘wordid’ 78



Output of ‘wordid’ 37



Output of ‘wordid’ 397

Image representation using bag of words:

Section 3.1

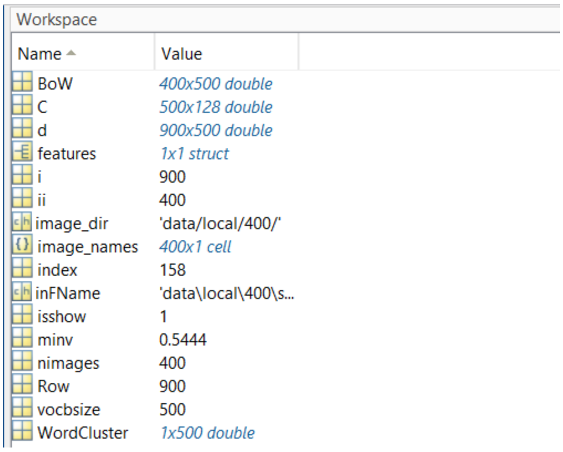
Each image is a Bag of Words histogram that has been normalised using the predefined do normalise() function. The training images range from 1 to 300 in the dataset, while the testing images range from 301 to 400. This is the code that was used to generate this output, as well as the output.

Code:

```
%% 3.1 represent each image using BoW
%{%
clear;
BoW =[]; %initialization
isshow =1 ;%0; % show image and histogram or not
load('data/global/image_names');
load('data/global/dictionary','C');
%load('data/global/all_features');
nimages = 400;
vocbsize = 500;
for ii = 1:nimages
      image_dir=sprintf('%s/%s/','data/local',num2string(ii,3));
% location where detector is saved
      inFName = fullfile(image_dir, sprintf('%s', 'sift_features'));
      load(inFName, 'features');
      %------------- write your own code here--------------------------------
```

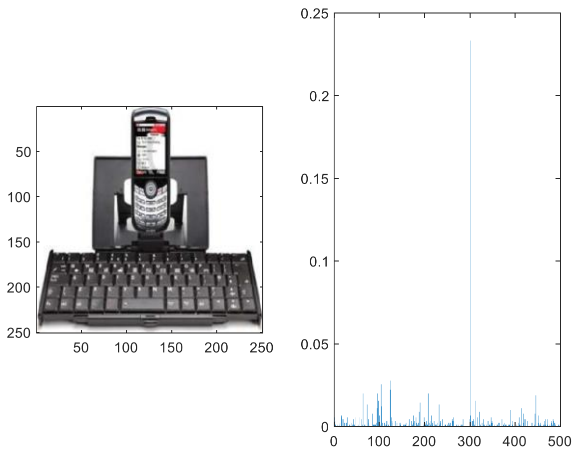
```
      % Jahnvi's code___210538601
      % we get 900 indices as for each image there are 900 features
      % by counting occurences you construct 500 columns feature vector BoW
      % 1 2 1 3 1 (900) ->   3 1 1(500)
      d = EuclideanDistance(features.data, C);
      Row = size(d,1);
      WordCluster = zeros(1,vocbsize);
      for i =1:Row
        [minv,index] = min(d(i,:));
        WordCluster(1,index) = WordCluster(1,index)+1;
      end
      BoW(ii,:) = do_normalize(WordCluster);
      %------------------- write your own code here--------------------------
      if isshow == 1
        close all; figure;
        subplot(1,2,1),subimage(imread(strcat('image/',image_names{ii})));
        subplot(1,2,2),bar(BoW(:,ii)),xlim([0 500]);
      end
end
%
save('data/global/bow','BoW')
%}
%------------------------end of code-----------------------------------
%==========================================================================
%==========================================================================
```

Outputs:



Workspace

Name	Value
BoW	400x500 double
C	500x128 double
d	900x500 double
features	1x1 struct
i	900
ii	400
image_dir	'data/local/400/'
image_names	400x1 cell
index	158
inFName	'data\local\400\s...
isshow	1
minv	0.5444
nimages	400
Row	900
vocbsize	500
WordCluster	1x500 double



NN classifier:

Section 4.1

Usage of BoW variable form section 3.1(code provided in the lab1.m). By default, this section employs a 1-NN search algorithm with L2 normalisation. "knnsearch.m" employs the arguments specified in the lab assignment document, where "T" equals 100, "D" equals 500, and "N" equals 300. The letter "k" is chosen as 1. In this example, the data is searched between rows using the method "knnsearch.m" and the training data is contained within the closest neighbour (the search is performed between rows as defined by the method's documentation).

Workspace output

Workspace

Name	Value
k	1
method	1
NNresult	*100x1 double*
predict_label	*100x1 double*
tem	*20x1 double*
test_data	*100x500 double*
test_labels	*100x1 double*
train_data	*300x500 double*
train_labels	*300x1 double*

Section 4.2

According to the workspace, 24 percent of the samples are classified incorrectly. Class 1 has 18 correct classifications and 2 incorrect classifications in the outputs. The number of misclassifications for each class is 1, 4, 1, and 16.

Workspace

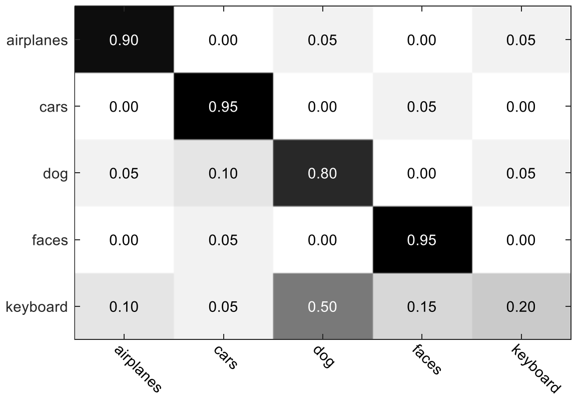
Name	Value
err	[0.1000;0.0500;0....
err_all	0.2400
error_n	[2;1;4;1;16]
k	5
method	1
NNresult	*100x1 double*
num_c	5
num_pc	20
predict_label	*100x1 double*
tem	*20x1 double*
test_data	*100x500 double*
test_labels	*100x1 double*
train_data	*300x500 double*
train_labels	*300x1 double*

Workspace output

Section 4.3

In this step a confusion matrix is generated by executing the code provided in the lab assignment. Here airplanes, cars, dogs, faces, and keyboard are the classes in Parts 4.2 and 4.3, respectively. The confusion matrix also depicts how misclassifications per class are distributed to other classes as well as general information obtained from Section 4.2.

Outputs



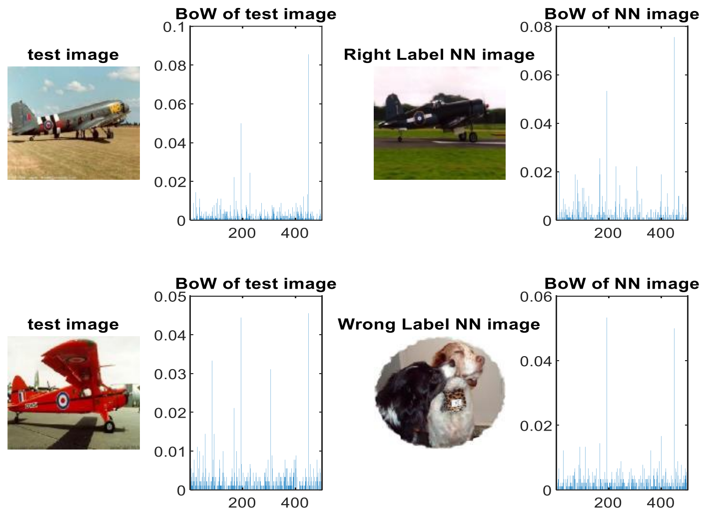
Workspace

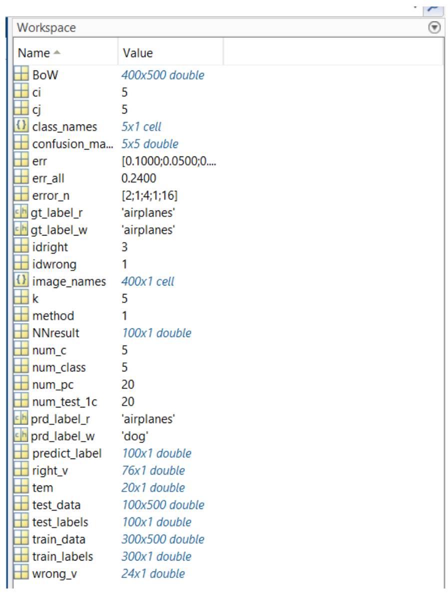
Name	Value
ci	5
cj	5
class_names	*5x1 cell*
confusion_ma...	*5x5 double*
err	[0.1000;0.0500;0....
err_all	0.2400
error_n	[2;1;4;1;16]
k	5
method	1
NNresult	*100x1 double*
num_c	5
num_class	5
num_pc	20
num_test_1c	20
predict_label	*100x1 double*
tem	*20x1 double*
test_data	*100x500 double*
test_labels	*100x1 double*
train_data	*300x500 double*
train_labels	*300x1 double*

Section 4.4

With the code provided in the assignment, there is generation of outputs where some correctly and incorrectly classified images are being depicted below in the output section.

Output





Section 4.5

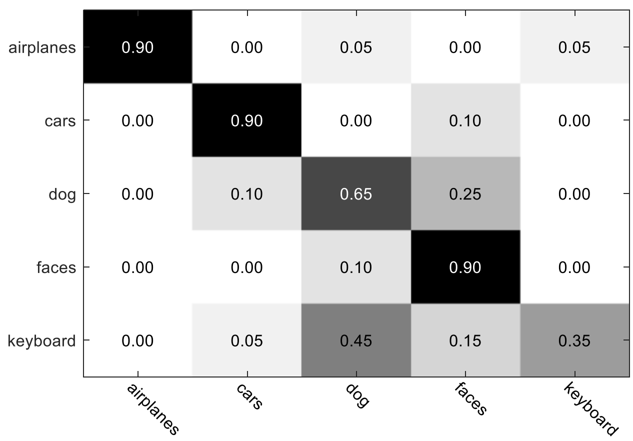
For section 4.5 the steps 4.1 and 4.4 are executed by using the method histogram intersection in the 'knnsearch.m' function. The below mentioned code for 'knnsearch.m' is successfully implemented with the results mentioned in 'Output' section.

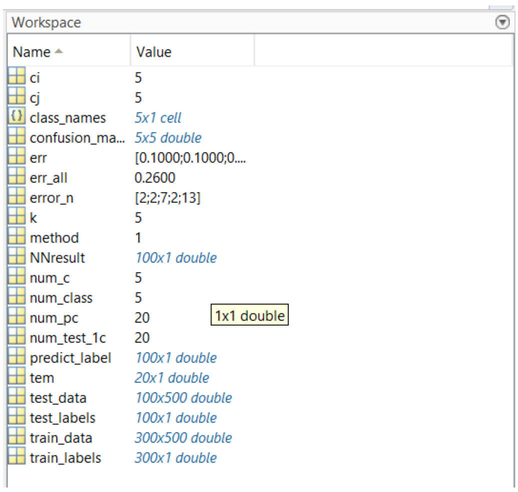
Code:

```
% -------------- write your own code here ---------------
  %Jahnvi's code____210538601
  d = 1;

  for i = 1:p
    d = d - min(a(1,index),b(1,index));
  end
  % -------------- write your own code here ---------------
```

Outputs:





Dictionary size:

In this part the dictionary size is altered from 500 to 20 and necessary observations and steps are taken with it.

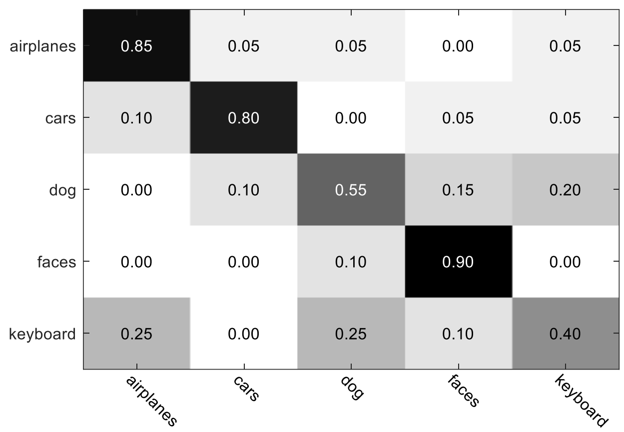
Section 5.1:

Results obtained with Method 1-L2

Executing step 4.2 the workspace looked like the one shown below. Starting from left is the *Step 4.2 – classification of error* and on the right there is *Step 4.3 confusion matrix generated* for L2 method.

Workspace

Name	Value
err	[0.1500;0.2000;0....
err_all	0.3000
error_n	[3;4;9;2;12]
k	5
method	1
NNresult	100x1 double
num_c	5
num_pc	20
predict_label	100x1 double
tem	20x1 double
test_data	10 20x1 double
test_labels	100x1 double
train_data	300x500 double
train_labels	300x1 double



Results obtained with Method 2-Histogram with intersection

Executing step 4.2 and 4.3 the workspace looked like the one shown below. Starting from left is the *Step 4.2 – classification of error* and on the right there is *Step 4.3 confusion matrix generated* for Histogram with intersection method.

Workspace

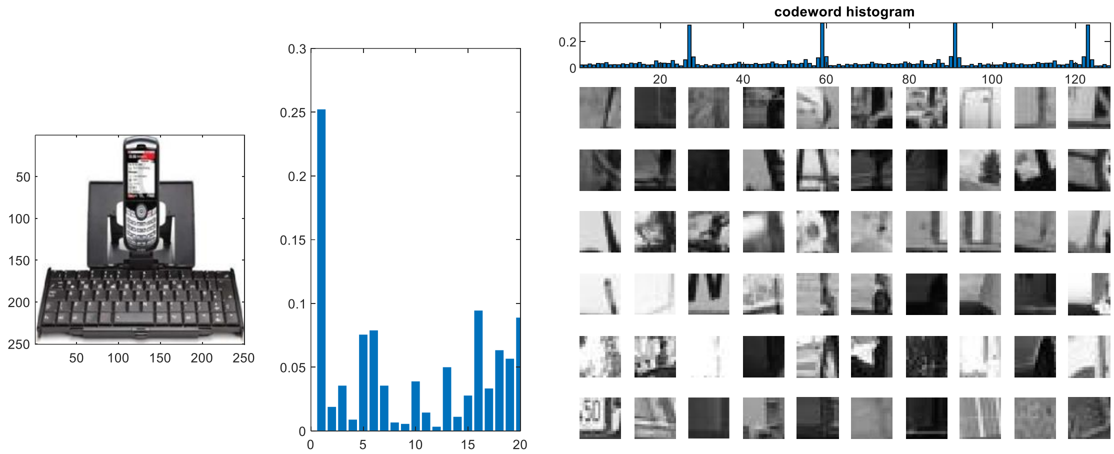
Name	Value
error_n	[0;0;0;0;0]
k	1
method	2
num_c	5
num_pc	20
tem	20x1 double
test_data	100x500 double
test_labels	100x1 double
train_data	300x500 double
train_labels	300x1 double

Name	Value
error_n	[0;0;0;0;0]
k	1
method	2
num_c	5
num_class	5
num_pc	20
tem	20x1 double
test_data	100x500 double
test_labels	100x1 double
train_data	300x500 double
train_labels	300x1 double

Section 5.2:

After the execution there was a drop in the performance due to the decrease in size of dictionary. As it is too small for train and test dataset which is presented. This creates a classic machine learning underfitting problem. From the histogram it can be seen that there are multiple image patches that are repetitive and of same kind in complete codewords set.

Output



SVM classifier:

Section 6.1:

Predefined code from 6.1 was executed and the results were observed to train the model with a linear multiclass SVM algorithm for each class of image. Using the commented code, cross validation is used to calculate the optimal values for C. With each iteration, the accuracy improves, and a total of 168 iterations were completed.

Output:

Command Window

```
optimization finished, #iter = 156
nu = 0.008126
obj = -399.403181, rho = 2.249448
nSV = 39, nBSV = 0
Total nSV = 183
Cross Validation Accuracy = 85.3333%
10 1.5 85.3333 (best c=256, g=1.31951, rate=86)
Elapsed time is 207.550484 seconds.
fx >>
```

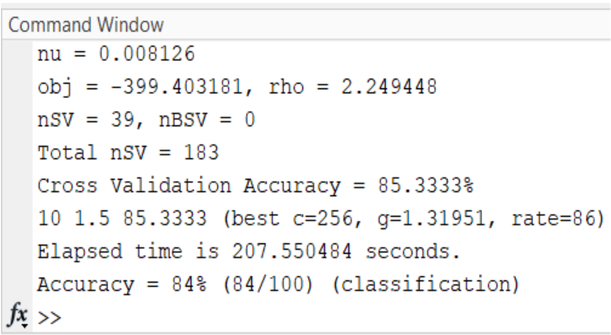
Workspace

Name	Value
bestc	256
bestcv	86
bestg	1.3195
cmd	'-v 5 -t 2 -c 1024 ...
cv	85.3333
log2c	10
log2g	1.5000
model	1x1 struct
options	'-s 0 -t 2 -c 256.0...
tem	20x1 double
test_data	100x500 double
test_labels	100x1 double
train_data	300x500 double
train_labels	300x1 double

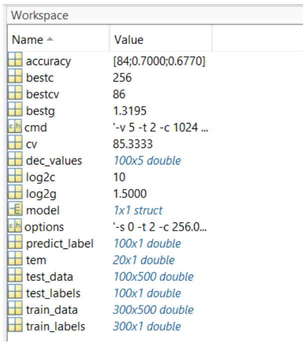
Section 6.2:

The overall accuracy after classifying all of the test images was 84%.

Output:



```
Command Window
nu = 0.008126
obj = -399.403181, rho = 2.249448
nSV = 39, nBSV = 0
Total nSV = 183
Cross Validation Accuracy = 85.3333%
10 1.5 85.3333 (best c=256, g=1.31951, rate=86)
Elapsed time is 207.550484 seconds.
Accuracy = 84% (84/100) (classification)
fx >>
```



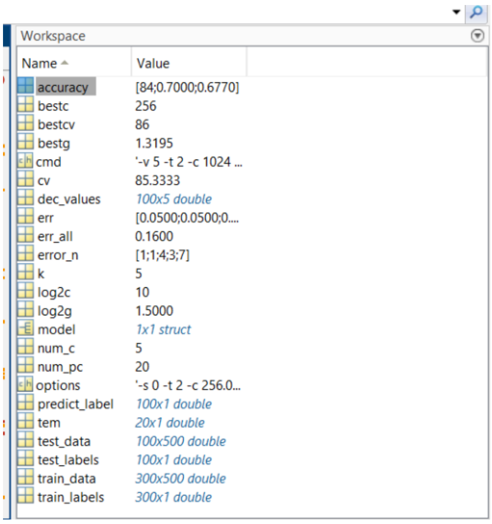
Workspace

Name	Value
accuracy	[84;0.7000;0.6770]
bestc	256
bestcv	86
bestg	1.3195
cmd	'-v 5 -t 2 -c 1024 ...
cv	85.3333
dec_values	100x5 double
log2c	10
log2g	1.5000
model	1x1 struct
options	'-s 0 -t 2 -c 256.0...
predict_label	100x1 double
tem	20x1 double
test_data	100x500 double
test_labels	100x1 double
train_data	300x500 double
train_labels	300x1 double

Section 6.3:

Overall classification accuracy is 84% with 1,1,4,3 & 7 as the misclassifications per class respectively. The observed output is for dictionary size of 500.

Output:



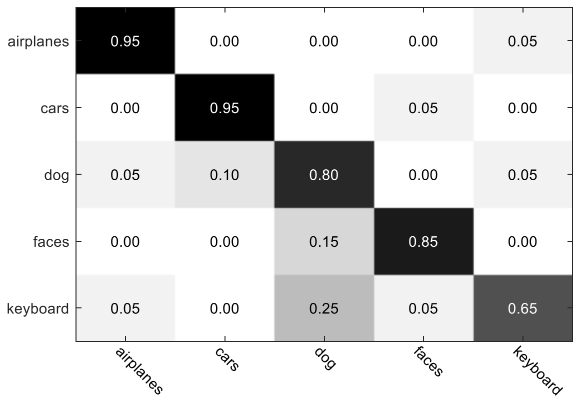
Workspace

Name	Value
accuracy	[84;0.7000;0.6770]
bestc	256
bestcv	86
bestg	1.3195
cmd	'-v 5 -t 2 -c 1024 ...
cv	85.3333
dec_values	100x5 double
err	[0.0500;0.0500;0....
err_all	0.1600
error_n	[1;1;4;3;7]
k	5
log2c	10
log2g	1.5000
model	1x1 struct
num_c	5
num_pc	20
options	'-s 0 -t 2 -c 256.0...
predict_label	100x1 double
tem	20x1 double
test_data	100x500 double
test_labels	100x1 double
train_data	300x500 double
train_labels	300x1 double

Section 6.4

The confusion matrix obtained for the code provided in section 6.4 is as shown in 'Output' section below.

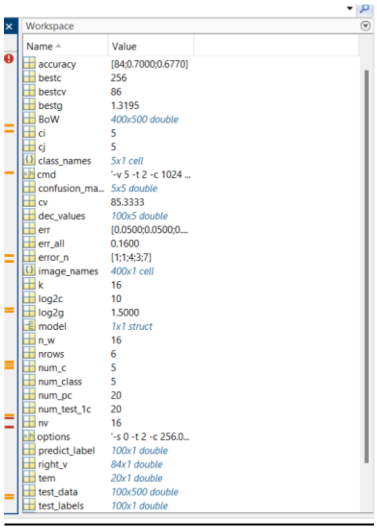
Output



Section 6.5

SVM performs well when the data points are well separated, but poorly when they are not. Because there are many overlapping data points within the same dataset, SVM cannot create a proper boundary between the datasets, resulting in incorrect classifications and misclassifications.

Output:



References:

https://towardsdatascience.com/bag-of-visual-words-in-a-nutshell-9ceea97ce0fb