Universidad Complutense de Madrid

# Visual object detection in live video streams

Apical LTD, United Kingdom

Elena Castilla, Pedro Chocano, Ander Irastorza, Marta López and Fernando Sebastián
05/07/2016

# ÍNDEX

# Introduction

Nowadays, technology is not an option, it is a need. The detection in Live Video Streams is everywhere, in our smart phones (detecting faces and also age), in security environments (streets, preventing riots) and many important companies like Google and Tesla are working on autonomous vehicles.

There are different mathematical models to resolve this problem but because of the dimensionality of the data set, there are problems like:

1. Computational costs
2. Autonomy and energy consumption
3. The need of large data set of samples.

The objective of this problem is to train the system to detect a new object (we choice apples). The system already knows hands and faces.

# Approach and work plan

On the first day we attend a lecture about the basics of Machine Learning Theory that is relevant for the Project. Our first task was to collect a lot of pictures from apples (the positives ones) and the same amount of pictures from others objects (like hands, faces, etc.).

On the second day we used the fisher linear discriminant model to separate the two classes of data. We had dimension problems of data, so we used principal components analysis (PCA) to reduce the data dimension and avoid collinearity problems. We were working with two sets, training and test. We tried to get the value of the coefficient b which represents the translation of the hyperplane w. We calculated it using an entropy function, which tells us how much information we have of the data set. We chose b subject to maximize entropy function.

After that we used the linear support vector machine model, which improves our results. And finally we implemented our final model in the system.

# 1. Working with the data

We collected 1000 pictures negatives and 1000 pictures positives. We use MATLAB to work with this data. First we use Histogram Oriented Gradient to transform pictures into a matrix, where each row is a picture (2400 variables).

## 1.1 HOG: Histogram of oriented gradients

The main goal of HOG is to try to get a vector associated to a picture to characterize it. This characterization is based on gradient orientation in localized portions of an image.

The first step is to divide the image into small adjacent, non-overlapping regions, called “cells”. Cells can be either rectangle or radial (in this case square).

Next, we compute the gradient directions over the pixels of the cell. This gradient is expressed on polar coordinates, with the angle constrained to be between 0 and 180 degree. More specifically, if the input is a window I from a gray-scale image, the two components $I_x$ and $I_y$ of the gradient of I are computed by central differences:

$$I_x(r,c) = I(r,c+1) - I(r,c-1) \quad and \quad I_y(r,c) = I(r-1,c) - I(r+1,c).$$

The gradient is then transformed to polar coordinates

$$\mu = \sqrt{I_x^2 + I_y^2} \quad and \quad \theta = 180/\pi(\tan_2^{-1}(I_x, I_y) \ mod \ \pi).$$

Once these gradients are calculated, we discretize their values in the so called Cell Orientation Histograms.

Cells can also be grouped in “blocks”, in which accumulate histograms are normalized.

The HOG descriptor is then the concatenated vector of the components of the normalized cell histograms from all of the block regions.

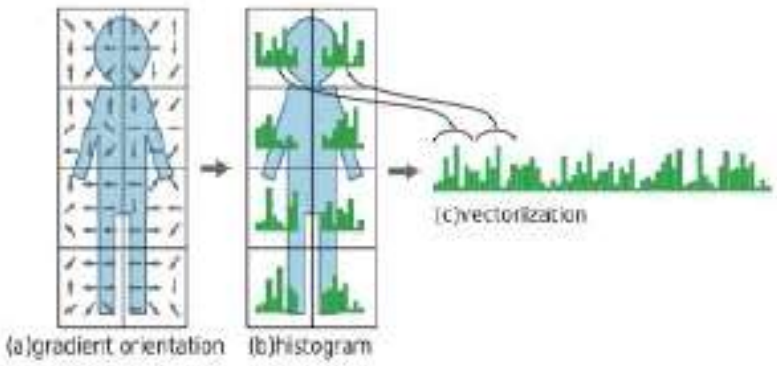



*Figure 3.5 (a) (b) Overview of HOG calculation*

Then we split randomly the data in two: training and test.

# 2. Fisher linear discriminant

The fisher linear discriminant uses covariances ($\Sigma_1 \; and \; \Sigma_2$) and means ($m_1, m_2$) for each class. We search a hyperplane that separate the two classes the "best" possible.

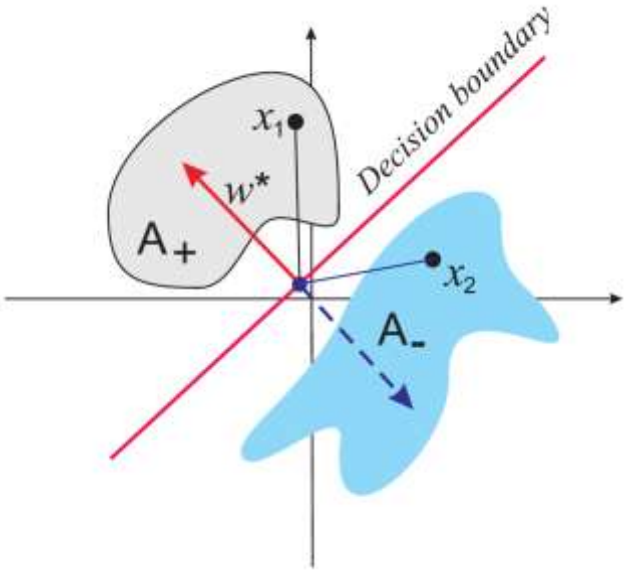
We calculate a direction (vector) w* so that

- ✓ The distances between the centroids of projected points on w* are maximized
- ✓ The "variance" is minimized

This leads to this problem

$$\max_{\mathrm{w}} \mathrm{w}^{\mathrm{t}}(\mathrm{m}_1 - \mathrm{m}_2)(\mathrm{m}_1 - \mathrm{m}_2)^{\mathrm{t}}\mathrm{w}$$

$$\mathrm{s.t.}\, \mathrm{w}^{\mathrm{t}}(\Sigma_1 + \Sigma_2)\mathrm{w} = 1$$




The optimal solution of this problem is:

$$\mathrm{w} = (\Sigma_1 + \Sigma_2)^{-1}(m_1 - m_2).$$

So, if $x^T w + b > 0$ then decide class $A_+$. Where x is an observation from the data set, w the coefficients of the vector and b the thresold. We chose b such that it maximizes Relative Information Gain. We solved this problem over the training set.

We faced with the fact that $(\Sigma_1 + \Sigma_2)$ is singular. In order to solve this we used Principal Components Analysis (PCA).

# 3. Principal Components Analysis

## 3.1 What is Principal Components Analysis?

Principal component analysis (PCA) is a statistical method that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. These new variables are a lineal combination of the original variables.

PCA is usually used to reduce the dimension of a problem. However, the main idea of Principal Components Analysis, according to Pearson, is to explain the data set through a coordinate transformation. This transformation is defined in such a way that the first principal component has the largest possible variance and the rest of components are ordered so that they explain most of the variance.

In addition, PCA is closely related to factor analysis. Factor analysis typically incorporates more domain specific assumptions about the underlying structure and solves eigenvectors of a slightly different matrix. Also, it is also related to canonical correlation analysis (CCA). CCA defines coordinate systems that optimally describe the cross-covariance between two datasets while PCA defines a new orthogonal coordinate system that optimally describes variance in a single dataset.

## 3.2 How to calculate Principal Components?

Let`s consider a data matrix, $X$ with column-wise zero empirical mean (the sample mean of each column has been shifted to zero), where each of the *n* rows represents a different repetition of the experiment, and each of the *p* columns gives a particular kind of feature.

$$X = \begin{pmatrix} x_{11} & \cdots & x_{1p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{np} \end{pmatrix}$$

The main idea is to calculate a new set of uncorrelated variables $\{y_1, \dots, y_p\}$. In order to calculate these variables, we need define a lineal transformation.

Mathematically, this transformation is defined by a set of *p*-dimensional vectors of weights or *loadings* $a_k = (a_{1k}, a_{2k}, \dots, a_{pk}),\ k = 1, \dots, p$ that map each row vector $x_j$ of $X$ to a new vector of principal component *scores* $y_k = (y_{1k}, y_{2k}, \dots\dots, y_{pk})$, given by

$$\begin{pmatrix} y_{1k} \\ \vdots \\ y_{pk} \end{pmatrix} = \begin{pmatrix} x_{11} & \cdots & x_{1p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{np} \end{pmatrix} \begin{pmatrix} a_{1k} \\ \vdots \\ a_{pk} \end{pmatrix}$$

The vectors of components principals $(y_1, \dots, y_k, \dots, y_p)$ of this lineal transformation are orthogonal because they are eigenvector of the covariance matrix, which is symmetric.

As we see, we need to calculate vectors of weight to get the principal components. So, we need to resolve the next problem,

$$\max_{a_k} \; var(y_k) = \max_{a_k} \; a_k' \Sigma a_k$$

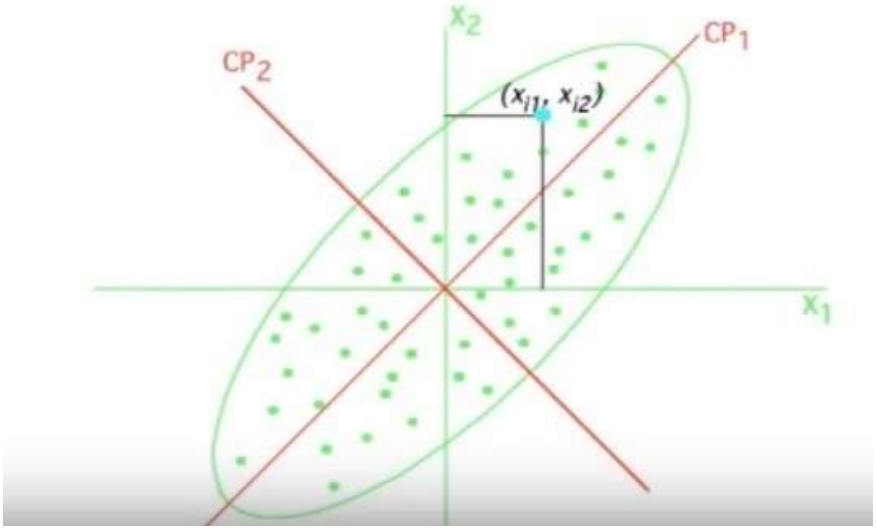
$$a_k' a_k = 1$$

$$cov(y_k, y_j) = 0 \quad j = 1, \dots, k-1$$

From this problem, it is easy to realize that principal components are a projection of the original variables on the directions defined by the eigenvectors of the covariance matrix

## 3.3 Geometric Interpretation of PCA

PCA can be thought of as fitting an *n*-dimensional ellipsoid to the data, where each axis of the ellipsoid represents a principal component. If some axis of the ellipse is small, then the variance along that axis is also small, and by omitting that axis and its corresponding principal component from our representation of the dataset, we lose only a commensurately small amount of information.

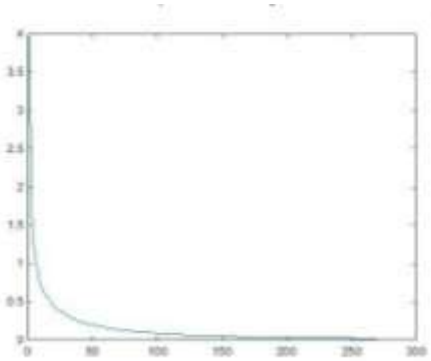


## 3.4 Criteria for determining the number of components

As we saw before, the number of principal components is the same than the number of original variables and we are interested in reducing the dimension of our problem. For that reason it is very important to choose a good number of principal components. I mean a number of components which explain the most of the data but it uses a few principal components. There is no a perfect number of principal components but there are some rules that can help to choose it.

For this problem, we used the Kaiser Rule. This criterion chooses all the principal components whose eigenvalues are bigger than the average of all of them.

As we can see in the graphic, only a few principal components explain the most of the variance. According to the Kaiser Rule, the “magic” number of principal components is 250.



# 4. Results

After applying PCA to our problem, we could solve the singularity matrix problem. In this way, we got 90% hit rate. It seems a good percentage, but for our problem that is not true.

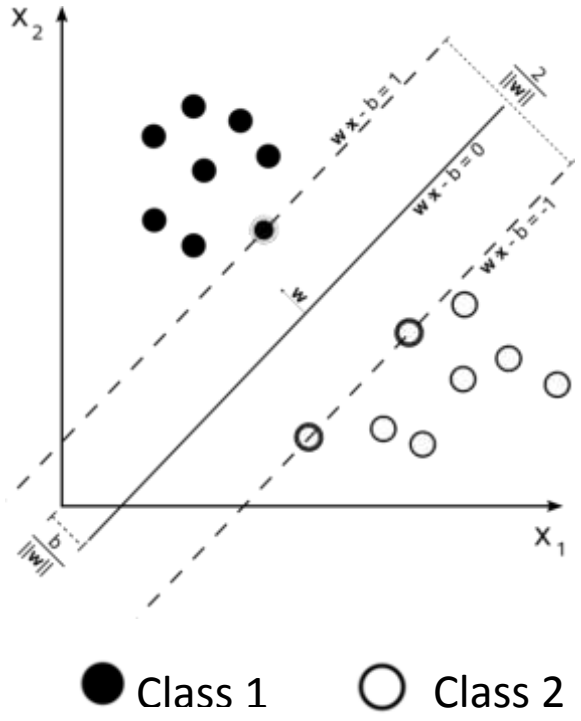
This is not a traditional problem; we are working with streaming video. As consequence of this, we are working with more than 150000 detections windows per frame.

If we do some math, we can see that 90% hit rate means that we have 15000 false positives per frame, which are 375000 false positives per second. That`s something you can see with your naked eye!

These results were expected for us because we were working with a simple model which only uses covariance and means.

# 5. Support Vector Machine

This machine learning technique is also useful for our problem. The objective of this method is find a hyperplane that separate the two class of data, maximizing the distance between the hyperplane and the nearest point from either group. In other words, SVM maximize the margin around the separating hyperplane.




Unlike the **linear fisher discriminant**, this method does not use any statistical information.

## 5.1 Mathematical formulation (Linear SVM)

A hyperplane can be represented by a normal vector w (which is perpendicular to the hyperplane) and the intercept **b**. So, all the points in the hyperplane satisfy $\mathrm{w^T x} = \mathrm{b}$.

We have the following optimization problem:

$$Maximize \quad \frac{2}{\|w\|}$$

$$subject\ to$$

$$w^T x_i - b \geq 1$$

$$w^T x_j - b \leq -1$$

Where $x_i$ are the observations corresponding to the class 1 and $x_j$ to the other class. Now, we transform the problem into a nicer one (quadratic optimization problem, computationally cheap):

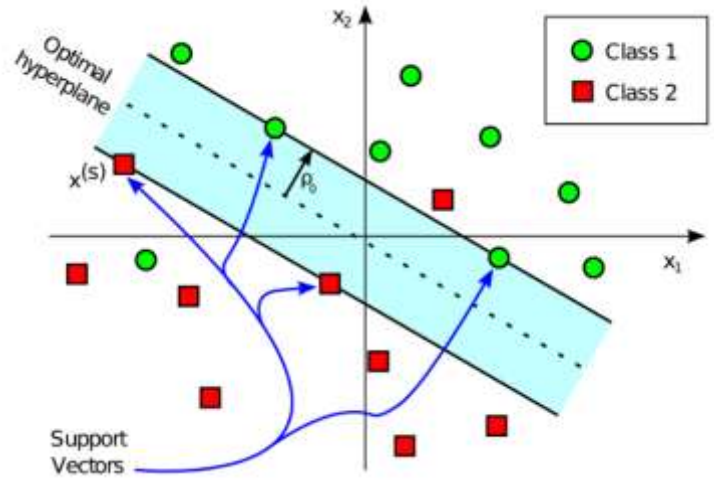
$$Minimize \qquad \|w\|^2$$

$$subject\ to$$

$$w^T x_i - b \geq 1$$

$$w^T x_j - b \leq -1$$

However, this formulation is not valid for non-linearly separable data.




We introduce a penalized vector $\gamma$ in order to solve this problem.

$$Minimize \qquad \|w\|^2 + c\gamma$$

$$subject\ to$$

$$w^T x_i - b \geq 1 - \gamma$$

$$w^T x_j - b \leq -1 - \gamma$$

$$\gamma > 0, c > 0.$$

Solving this problem using the training data, we get a classificator that only depends on **w** and **b**. So, in our case the classifier has the parameters $\alpha = (w, b)$,

$$f\big(x, (w, b)\big) = w^T x - b.$$

And if $f\big(x, (w, b)\big) > 0\ then\ x \in Class\ 1, either\ x \in Class2.$

## 5.2 Remarks

- ✓ The maximum points that a hyperplane in a real n dimensional space will separate with probability 1 is n+1.

- ✓ In order to avoid trivial classification, we introduce a cost function.
  We penalize more the largest class (which is easy to classify)

- ✓ We get an approximately 98 % hit rate in cross-validation in the original dimension (2400) and 85% in the reduced dimension (250)

- ✓ In this method, like Fisher Linear Discriminant, the belonging to a class does not depend on the distance to the hyperplane

# 6. Results

## 6.1 Hit Rate of 98%

As we have seen before, a *high* hit rate could not be enough to detect an object in live video streams, because of the large amount of false positives or miss detections. In fact, for a 98% hit rate, we have a 2% of error (either false positives or miss detections).

In a pessimistic estimation (where all the error were false positives), for our problem, if we have approximately 150 000 detection windows for a picture, we could have 3 000 false positives, which means (as we have 25 frames per second) that we could have 75 000 false positives per second. This number of false positives will be easily detected by human eye.

Again, as we proceeded before, our goal is improving our hit rate. But a 98% hit rate would be hardly improved changing our model, almost without a high computational cost. So, why are not we able to improve this hit rate?

Now, we have to look into the data. First of all, an apple has a very common shape. This is a remarkable problem, because the apple's shape could be easily confused with other objects like other rounded fruits or a lot of rounded objects.

Also, using PCA has not helped us to solve this problem, because using it we have lost valuable information by reducing variables, which may have help us to distinguish apples.

So, once we have rejected improving our hit rate improving our models, we have to improve our data.

## 6.2 Improving Hit Rate

There is one thing left we can do to improve our model before we change our data. We can add a *colour classification*. We have been using samples in grayscale, but colour is a very particular specification of an object. So, if we add this criterion, we can improve our model.

This increases the computational cost, but as we have seen in out tests, also improves the accuracy of the model making it worthwhile.

In our tests, this allowed us to detect our apple, but also we got a lot of false positives.

Also, for our very first tests, we tried reducing the size of the sample. We took only 20 positives. In fact, we took 20 pictures of the same apple. This helped us to distinguish one simple apple. The reason this works is because in high dimensional spaces, one object is close to the border. So then, is easy to split one object from the rest of the universe.

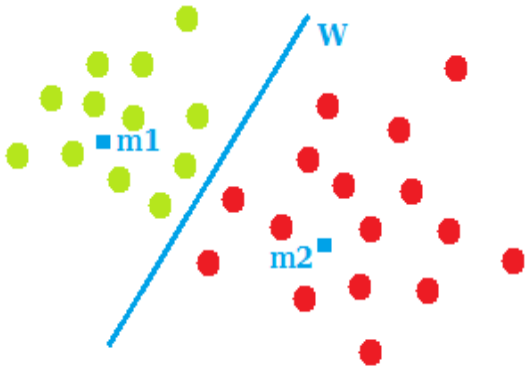
This technique helped us to clearly distinguish one single apple in a live video stream with a hit rate of 100%.

However, we want to establish a general method, so we need other techniques.

The key for generalizing the method is to increase our samples. Also, we do not want just more samples but *quality* samples.

To obtain these new samples, we use a technique called *prototyping*. Our goal is to clearly separate positives and negatives. That is, to find a hyperplane (or a **band**) which separates our two sets.

As we have seen before, this hyperplane is closely related to the distance between the mass centres of both sets. Also, our problem with the false positives remains on the fact that there are a lot of points in the border of the two sets.




*Hyperplane which separates the two sets and the mass centres*

So, we want now to take samples close to the border. These will be our quality samples. If we take samples this way, we could accumulate points in both sides of the border. This makes the

mass centres to get close to the border, so the distances between them is too small, and we will find a hyperplane that splits both sets –which will be almost the border.

But first we have to get the samples close to the border. This is achieved by the *prototyping* technique. It is based on taking all the false positives given by the method, and putting them into the negative's set. Then you retrain your model with the new samples. If you repeat it again (adding samples), you get the quality samples you need, and this approaches the model's hit rate to 100%.

**NOTE:** In spite of this technique improves a lot the model, especially to generalize it, we were running out of time in our test, so we did not apply this method to our experiments. Our last change was to reduce the sample to one single apple and add colour classification.

# 7. Conclusions

- ✓ We had obtained first-hands experience in training a real functional object detection system. We also acquire practical data mining and data analysis skills and experimented with measure concentration effects in high dimensions.

- ✓ Our purpose was detecting a specific object in live video streaming. In order to get it, we have learnt HOG, Fisher Linear Discriminant, SVM, PCA and some concepts of Machine Learning.

- ✓ We have dealt with problems related to Big Data. Maybe the most difficult of this project was to work with a big dimensionality of data set.

# 8. Further Work

- ✓ Collecting more negatives and positives for our data set. With more pictures we can distinguish a specific object of the others which are in the real world.

- ✓ Using quadratic Kernel in the model Support Vector Machine. Nowadays, this method is not available in the system that we used to detect object in live streaming video.

# Appendix

Code to Split the data:

```
% Separa una matriz en test y training

function [train,test] = split(matrix,perc)
    seq = randperm(size(matrix,1));
    point = floor(size(matrix,1)*perc);
    train = matrix(seq(1:point),:);
    test = matrix(seq(point+1:end),:);
end
```

Code to Fisher Linear Discriminant:

```
function [r,r1,r2]=Fisher_lim(test,train,lim)

A=train(train(:,end)==1,:);
B=train(train(:,end)==0,:);
A(:,end)=[];
B(:,end)=[];
covA=cov(A);
covB=cov(B);

% means
m1=sum(A)/size(A,1);
m2=sum(B)/size(B,1);

mean_all=m1-m2;

%% PCA

train2=train;
train2(:,end)=[];
[COEFF,SCORE,latent] = princomp(train2);
ind = train(:,end);
PCA=SCORE(:,1:lim);
PCA = [PCA ind];
A=PCA(PCA(:,end)==1,:);
B=PCA(PCA(:,end)==0,:);
A(:,end)=[];
B(:,end)=[];

covA=cov(A);
covB=cov(B);

mean_all = sum(A)/size(A,1) - sum(B)/size(B,1);
sigm=covA+covB;

mean_all = mean_all(:);
w=sigm\mean_all;
H = COEFF(:,1:lim);
wt = w*H';

%% Training
Results_P=A*w';
Results_N=B*w';

Rpos=[Results_P>0];
Rneg=[Results_N<0];

%% Test

%[COEFF,SCORE,latent] = princomp(test2);

ATEST=test(test(:,end)==1,:);
BTEST=test(test(:,end)==0,:);
ATEST(:,end)=[];
BTEST(:,end)=[];

% RESULTS
```

```
Results_P=ATEST*wt';
Results_N=BTEST*wt';

Rpos=[Results_P>0];
Rneg=[Results_N<0];

r1=sum(Rneg)/size(Rneg,1);
r2=sum(Rpos)/size(Rpos,1);
r=(sum(Rpos)/size(Rpos,1)*size(A,1)+sum(Rneg)/size(Rneg,1)*size(B,1))/
(size(A,1)+size(B,1));

% %% Using b
% Results_P=A*w';
% Results_N=B*w';
%
% % means
% mA=1/size(A,1)*sum(A);mA=mA*w';
% mB=1/size(B,1)*sum(B);mB=mB*w';
% S=[Results_P;Results_N];
%
% if (mB<mA)
%     grid = linspace(mB,mA,100);
% else
%     grid = linspace(mA,mB,100);
% end
% PA=size(Results_P,1)/size(S,1);
% PB=size(Results_N,1)/size(S,1);
% H_S = -(PA*log(PA)+PB*log(PB));
%
% P = Results_P;
% N = Results_N;
% RIG = 0;
% for i=1:length(grid)
%     b = grid(i);
%     % S_RIGHT
%     S_right = S(S>=b);
%     Size_S_right = size(S_right,1);
%     Size_P_right = size(P(P>=b),1);
%     Size_N_right = size(N(N>=b),1);
%
%     % S_LEFT
%     S_left = S(S<b);
%     Size_S_left = size(S_left,1);
%     Size_P_left = size(P(P<b),1);
%     Size_N_left = size(N(N<b),1);
%
%     % Probabilities
%     Prob_P_right = Size_P_right/Size_S_right;
%     Prob_N_right = Size_N_right/Size_S_right;
%
%     Prob_P_left = Size_P_left/Size_S_left;
%     Prob_N_left = Size_N_left/Size_S_left;
%
%     H_right = -(Prob_P_right*log(Prob_P_right) +
Prob_N_right*log(Prob_N_right));
%     H_left = -(Prob_P_left*log(Prob_P_left) +
Prob_N_left*log(Prob_N_left));
%
%     Size_S = size(S,1);
%
```

```
%     H_cond = Size_S_right/Size_S*H_right +
Size_S_left/Size_S*H_left;
%
%     RIG_aux = (H_S-H_cond)/H_S;
%     if (RIG_aux>RIG)
%         RIG = RIG_aux;
%         b_best = b;
%     end
%
% end
%
% b = b_best;
% Results_P=ATEST*w'+b;
% Results_N=BTEST*w'+b;
%
% Rpos=[Results_P>0];
% Rneg=[Results_N<0];
%
% %display('-----------------------------------------------------------
');
% %display('Negatives hit rate (using b)');
% r1=sum(Rneg)/size(Rneg,1);
% %display('Positives hit rate (using b)');
% r2=sum(Rpos)/size(Rpos,1);
% %display('Total hit rate (using b)');
%
r=(sum(Rpos)/size(Rpos,1)*size(A,1)+sum(Rneg)/size(Rneg,1)*size(B,1))/
(size(A,1)+size(B,1));
```

Code to Crossvalidation:

```
clear all

% Data
%load('E:\problem 5\Data\pos_neg_hogs.mat')

positives = ts_p_hog;
negatives = ts_n_hog;
positives(:,end+1) = 1;
negatives(:,end+1) = 0;

ts_total_hog_ind = [positives; negatives];
matrix=ts_total_hog_ind;

% Parameters
perc=0.8;
n=10;
lim=250;
delta_lim=100;
t=20;
L=linspace(lim-delta_lim,lim + delta_lim,t);

% Training & Test tables
[train,test] = split(matrix,perc);

%PCA Study
for j=1:length(L)

    % Cross Validation
    for i=1:n
        [train,~] = split(matrix,perc);
        [r(i),~,~]=Fisher_lim(test,train,floor(L(i)));
    end

    R(j)=mean(r);
end

plot(L,R)
```

Code to SVM:

```
load('pos_neg_hogs.mat')
load('8500_negs_extra.mat')

perc = 0.8;

positives = ts_p_hog;
negatives = ts_n_hog;
positives(:,end+1) = 1;
negatives(:,end+1) = 0;

ts_total_hog_ind = [positives; negatives];
matrix=ts_total_hog_ind;

[train,test] = Separa(matrix,perc);



% PCA
train2=train;
ind = train(:,end);
train2(:,end)=[];
[COEFF,SCORE,latent]=princomp(train2);
lim=size(latent(latent>=mean(latent)),1);
PCA=SCORE(:,1:lim);

numPos = size(ind(ind==1),1);
numNeg = size(ind(ind==0),1);

SVMModel = fitcsvm(PCA,ind,'Cost',[0,numNeg;numPos,0]);


w = SVMModel.Beta;
b = SVMModel.Bias;

H = COEFF(:,1:lim);
wt = w'*H';

test2=test;
ATEST=test2(test2(:,end)==1,:);
BTEST=test2(test2(:,end)==0,:);
test2(:,end)=[];
ind = test(:,end);


ATEST(:,end)=[];
BTEST(:,end)=[];

Results_P=ATEST*wt'+b;
Results_N=BTEST*wt'+b;

Rpos=[Results_P>0];
Rneg=[Results_N<0];

display('------------------------------------------------------------');
display('Negatives hit rate wt (Test)');
```

```
sum(Rneg)/size(Rneg,1)
display('Positives hit rate wt (Test)');
sum(Rpos)/size(Rpos,1)

display('Total hit rate  wt (Test)');
(sum(Rpos)/size(Rpos,1)*size(ATEST,1)+sum(Rneg)/size(Rneg,1)*size(BTES
T,1))/(size(ATEST,1)+size(BTEST,1))
```

**References**


1. Lectures' Teacher, Ivan Tyukin, Leicester University, United Kingdom
2. https://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf
3. http://www.mdpi.com/1424-8220/14/11/20713/htm
4. https://en.wikipedia.org/wiki/Support_vector_machine
5. https://en.wikipedia.org/wiki/Principal_component_analysis