Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques

Universitat de Barcelona

Automatic Labeling Application Applied to Food-Related Object Recognition

---

Marc Bolaños Solà

Directora: Petia Ivanova Radeva

Realitzat a: Departament de
Matemàtica
Aplicada i Anàlisi.
UB

Barcelona, 20 de juny de 2013

1. Introduction and State of the Art

1.1 Project Scope

It is clear that every day that goes by, technology is a little bit more present in our everyday life. Pervasive Computing (Ubiquitous Computing) is a fact that every one must assume, and even though ethical and moral topics will always arise. There are unlimited aspects and usual tasks in which Pervasive Computing can improve our quality of life.

One of the ways in which this recently emerging field can help us the most on making the quality of life of people a lot better, is on our feeding habits and all their related aspects: nutrition, physical activities, emotions and social interaction. Some of these environments are what this project intends to improve.

One of the most evident problems for which we could be interested in logging every bit of the diet of one person would be overweight. People who clearly need help on their nutrition and all the habits related to it (like physical activities, emotions and social interaction), could get an incredible benefit. Using a device or some interconnected devices which are able to monitor and record different kind of information, could help them overcome their habits, and solve their weight problems. But, ultimately, every person, even without any evident nutrition problem, could also take a great advantage from a device like that.

Although the advantages of keeping a record of feeding habits are crystal clear, we could wonder if it is necessary to do an automatic analysis, using some sort of device. Or is there any way for doing it manually? Writing down everything that we eat, the exercise that we perform and any other relevant information that would be needed? There is indeed, even though, some studies prove that we tend to underestimate our food intake and overestimate our physical activities, which means that doing it automatically would be even more meaningful.

Even though this is set into a larger work that we will overview in the following subsections, in general terms, in our final degree project we have focused on finding a way to use lifelogging technologies and deal with the high amounts of data that they produce to recognise common objects closely related to nutrition and eating habits, more precisely dishes [11].

1.2 Lifelogging

There are many different devices that help us logging any kind of information related to our daily routine, photo and video cameras; bracelets that record our vital signs; devices that read our GPS coordinates; or any other type of sensor that could "sense" what we are doing, where we are, what are we looking at, etc. And even our smart phones, which we wear with us every moment of the day, can become a powerful lifelogging machine that can feed us with a wide range of useful information.

Talking precisely about the benefits of not pervasive computing in general but lifelogging in particular, A. Sellen and S. Whittaker in [1] summarized by the Five Rs: recollecting, reminiscing, retrieving, reflecting and remembering intentions.

Recollecting can help us re-live past experiences for us to be able to recall where a particular object was situated or at which precise date and time we met someone.

Reminiscing, which would involve remembering emotions that any kind of situation could arise on us, like watching a film or a pleasant smell.

Retrieving would allow us to find in the present specific information that we saw or possessed in our past, like a document that we forgot where it is stored.

Reflecting, meaning to recognise and learn some patterns that we experienced in the past and that could have lead to some consequence and could benefit us in the future helping us to predict the same result, for example a series of situations that made us be stressed.

Remembering intentions, which involves recording information about future actions or plans that we want to accomplish like our appointments or schedule.

If we analyse the main objectives of this project, their most related "Rs", and therefore the devices that we choose must accomplish, would be Recollecting relevant information to analyse our 4 targets (nutrition, physical activities, emotions and social interaction); Reminiscing the emotions that certain situations could arise on us and Reflect some possible patterns that should be changed to improve our nutrition and physical habits and therefore our quality of life.

1.3 Nutrition, Physical Activities, Emotions and Social Interaction

There are 4 main goals that we want to accomplish, all of them tightly related to our quality of life and nutrition habits. And even though not all of them are obviously related to this fact, we will try to explain some of the implications that each of them can have on our health.

Nutrition, precisely an adequate and rich nutrition is clearly an important issue to take into account for anyone who wants to be healthy. Nutrition problems are widely known in our society, although not everyone is concerned about it and does much to solve them. Obesity and anorexia are diseases that all of us have heard about and that an application based on lifelogging could make a big leap to solve. Starting by recognising objects related to feeding and ending by being able to analyse the components of every meal that we have are important goals for the success.

Physical Activities are another one of the widely known aspects in our modern society, which influence directly to our physical health. Even though we all tend to overestimate the amount of exercise that we do, or do not even bother doing any. In this field, lifelogging would monitor every second that we are exercising, every beat of our heart for us to know if we can push harder or we are reaching our limit, and even create an automatic planning for us to improve our physical condition and therefore our health.

Emotions are more related to the diet and the nutrition health of a person that we could think at first. Some studies indicate that important emotional changes make us change our eating habits in the short and the long term, and situations like eating chocolate when you are depressed are in fact close related to this behaviour. For this reason, recording possible emotional changes along the day or during a larger period of time could show us patterns that helped us improve the nutrition of the subjects.

Social Interaction is closely related to emotions too. If someone tends to eat alone or with company, or even if we take into account any possible humour change that could provoke on us any person in our surroundings, we must know that it can make us having a less regular or lead us to have an unhealthier diet.

1.4 SenseCam and Other Lifelogging Devices

If we want to accomplish all our goals, we have to find a device that gives us enough information and from all the senses and locations that could solve our different needs. A device being able to capture as much information about our feelings and surroundings as we can. And even though maybe in a not very distant future ubiquitous computing will reach all this capacities, there are some feelings that can not be recorded by any device on the market, and even less anyone that can perform all those tasks at the same time.

Having analysed all the "senses" that would help us reach our objectives, we must choose which of them are essential and which ones are only complementary. And then find a combination of devices that helped us performing those tasks.

For logging all the nutritional aspects and social interactions, we could use one of the multiple wearable cameras that are available in the market for general users or for researchers right now. One of which is the Microsoft's SenseCam (Figure 1).

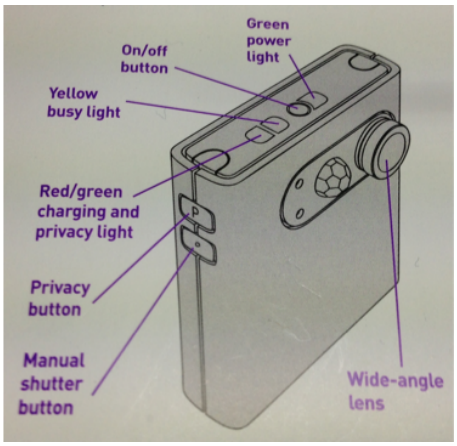

Fig 1. SenseCam scheme. Lifelogging device used for the Project.

SenseCam is a wearable camera designed for any lifelogging issues, even though, its main research goal was improving the memory retention of Alzheimer's patients. For this case of diseases, S. Hodges et al. [2] proved to make a difference. Its battery lasts for about 14h during which it takes a photo aproximately every 12 seconds. That means that at the end of the day, we will have on average about 4200 images. Furthermore it has a motion sensor that can tell if the user is moving too much and wait until the toss has stopped to take another photo. Although the motion sensor information is stored in the internal memory too, for this project we only used its images and their date and time log.

Fig 2. A SenseCame user wearing the camera.

As we can see in Figure 2, the user can wear the camera hanging on his/her neck without having to worry about taking any picture because it works on its own. Although, if you want to take one at a specific

moment, you only have to push the “Manual Shutter Button” once. Pressing during some seconds, the “Privacy Button” will turn it to standby and it will not take any photo for 15 seconds. Apart from that, it is able to indicate whether it is busy (taking a picture or processing the motion sensor signal), if its battery is charging or the privacy button has been recently pressed and posseses a wide angle lense in order to have the largest vision possible.

As we can see in [2], [8], [9], [10], [11], a lot of people have already been working for multiple purposes with wearable lifelogging devices like SenseCam, so we will be able to get a lot of useful information about their research.

And finally, for logging emotions, or a sign that indicated us that the subject's emotions have shifted, in a future extension of the project, we can use an Electronic Pulse Bracelets (Figure 3). They are more common everyday, and we could use the quick rise or the stabilization of the pulse signal as indicators to detect if the person is excited by some kind of stimulation (unknown by this device) or that he/she is calmed.

Fig 3. Example of Electronic Pulse Bracelet.

The same bracelet combined with some traking device, like a common smart phone with GPS, or a more advanced bracelet that is able to keep record of the distance, velocity, or movement in general of the user, could be used for monitoring the physical activities.

1.5 Machine Learning and Computer Vision

Our main goal at this point is recognising objects related to nutrition (e.g. dishes) in the surroundings of our subject analysing the photos taken by the SenseCam along the day. We can imply that if we detect a plate the user will be close enough to see it and therefore it will produce an effect on him.

The problem is: how can we automatically detect an object and every one of its instances in all of its variants, shapes and positions and in such a large number of images?

And our solution is by means of Computer Vision and Machine Learning techniques. This kind of algorithms (Decision Tree, Genetic, Neural Networks, Clustering, Support Vector Machine, etc.) use large amounts of input data to learn the possible patterns that describe the objects (distinguishing them from input data different of them) in order to be able to automatically determine if a new information tested by it after the training process is or not an instance of the element that we are detecting/trained.

Some of the most popular Machine Learning algorithms are: Single Linkage, LDA, LMNN, KNN, AdaBoost, K-means, etc. Given the "No free lunch theorem", we applied several of them in order to compare their performance and decide which one can be more beneficial to us in this specific application. Since it depends on the type of data and a wide range of variables our question is which of them can make a difference in any particular usage. We will discuss the algorithms, their differences and similarities and their complementarity in further sections.

In addition to the Machine Learning algorithms, we must establish some pre-processing methods and techniques to our images, Computer Vision algorithms, in order to standardise them and extract the most discriminating possible information (features) for the algorithms to have an efficient way of distinguishing the objects of interest: dishes.

The main problem about the combination of these techniques is that even if we try to normalise as much as possible the data, working with information as continuous as an image of the real world, it leads to the fact that the possible variables related to our objects are too much complex in order to develop a highly reliable unsupervised learner. For that reason, we chose to combine the supervised and the unsupervised learning (semi-supervised) and develop a supervised Active Learner combined with unsupervised algorithms [3], [5]. Moreover, it allows creating huge amounts of manually labelled images, in our case more than 90.000. Active labelling has the

challenging advantage that it allows to guide the process of labelling and assure that manual labelling will be done with the minimal manual effort of the labelling experts.

1.6 Active Learning Approaches

First of all, we must take into account that the basic utility and the most usual necessity of using an active learning are: the environments on which we have large numbers of samples to label, and the need for a user or oracle to label and validate the samples.

Roughly speaking, we can divide the approaches on Active Learning classification in 2 types [6]:

i) Classifier-Based Active Learning.

   Having a distribution of samples and a set of them already labeled by a master (the user that answers the labels of the samples queried by the software), determines the region of its distribution (based on the classifier that we are using) in which the uncertainty of the not labeled yet samples is higher [4].

ii) Data Distribution-Based Active Learning.

   Initially, we create a hierarchical distribution of our samples using their features as guidelines for their partition. Once the master has answered the labels of an iteration, the division of the samples in the cluster corresponding to those queries depends on the answer of the user and the previously constructed hierarchical tree [5].

The data distribution-based approach is classifier independent. Even though, as we did during the development of this project, both techniques can be combined.

Dasgupta's algorithm [5], [6], which we used as the main skeleton of this project, lies on the second approach, a data distribution-based one. Its main advantages comparing it with the first option are:

i) Guides the labelling process through a previously calculated clusterised tree depending on the impurity (or uncertainty) of the set of labels that reside in that particular level of the tree, travelling downwards dividing the sets but never upwards.
ii) Presents a method to calculate the error bounds or impurity of each of those clusters after each query step.

1.7 Project's Proposal Overview

Now that we have finally explored the different aspects of the project, due to the high range of possibilities and aspects in which we could do research and could develop technological solutions to the problem of healthy habits follow-up, we will summarize the main aspects on which we focused to accomplish:

a) Use a lifelogging device (SenseCam) in order to record the nutritional habits of a person and set a basis to improve them.
b) Find a method to extract, from the taken images, the objects of interest the cleanest way possible (dishes).
c) Determine the features that best describe and discriminate the dishes from any other object.
d) Develop an automatic learning method applicable to the wide range of objects related to the nutrition habits (Dasgupta's).
e) Propose forest-based active learning approach to additionally decrease the user involvement (clicks and time spent) on the active learning process.
f) Extend the active learning approach introducing feature extraction methods.
g) Validate the proposed method studying different classifiers and distance learning techniques.

All of them, leading to the final three newly researched contributions:

i) New approach to Active Learning.
ii) New technique applied to food-recognition.
iii) New user-oriented application for labelling huge amounts of samples.

2. Methodology

In this section we will review each of the algorithms we used to develop the Active Learner. We will explain how they work in general, without getting into details about their implementation in the project, since that will be explained in the following section.

First of all, we can see the scheme in Fig. 4. as an overview of all the techniques and related fields to this project. We must have in mind that this is not the only accepted vision of the Object Recognition, but the one that we took for our project.

In it we can find the techniques that we will analyse in this section (green) and all the concepts related to the project in general (object recognition or machine learning) and to more particular aspects that we will analyse further (features extraction, labels definition, Active Learning, dimensionality reduction, etc.).

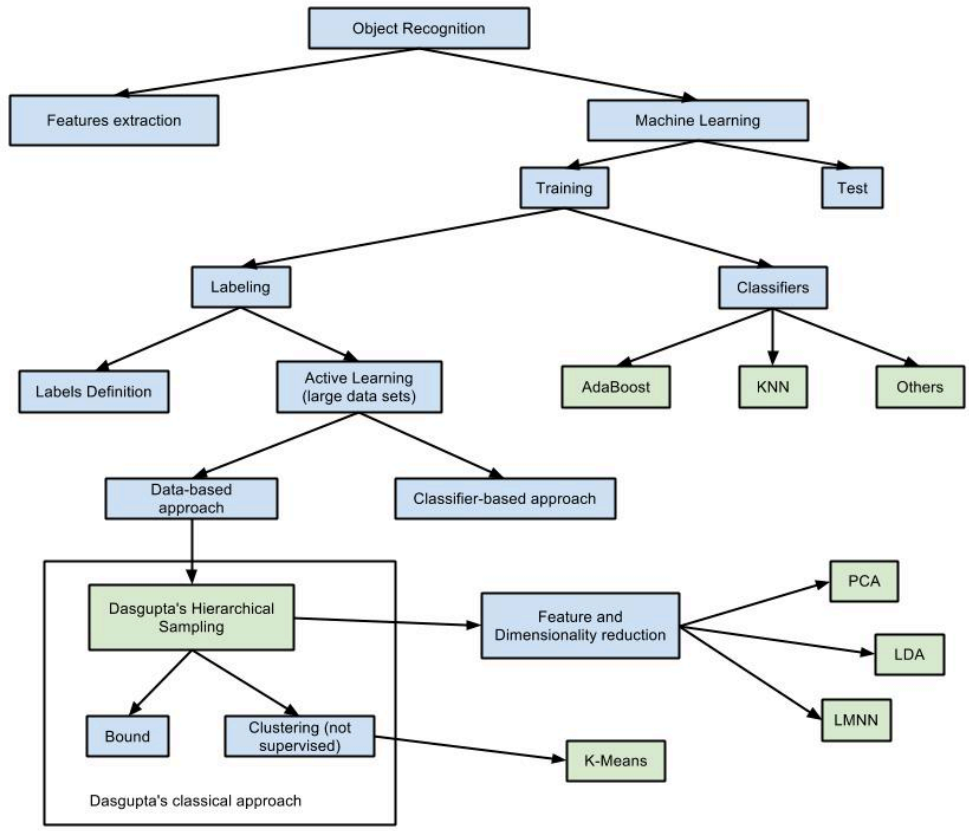

Fig. 4. Scheme overview of the partcular vision we took for this project of the object recognition. We can see all the techniques that we will analyse in this section (green) and other fields and concepts (blue) related to it.

In the following subsections we will start reviewing Dasgupta's approach for Active Learning, then we will continue analysing how the chosen classifier (or Online Learners like) techniques work (AdaBoost and KNN), after that we will explain the

dimensionality reduction (Active Learners like) algorithms used (PCA, LDA and LMNN) and finally we will end with the clustering algorithm K-Means.

2.1 Dasgupta's Cluster-Adaptive Active Learning

Knowing that the active learning techniques can be grouped in two different types (classifier-based [4] and data-based [5]), we propose a new approach that is able to combine a data-based active learner (Dasgupta [5]), and the use of a classifier that we will retrain after each labelling session (see section 1.6 Active Learning Approaches).

Focusing on Dasgupta's algorithm, it has the ability to recognise the distribution of the data and clusterise it using a hierarchical technique. Classically, Dasgupta's hierarchical clustering uses a tree created offline. This means that it is built before knowing any of the labels of our samples, only based on their features, and that is used without any modification during the whole labelling process. Once the tree has been created and the labelling has started, Dasgupta gives us the possibility to lower the level of the tree (and with it focusing in more little clusters and being more specific in their differences) step by step when needed, and therefore managing a better separation of the samples (see Fig. 5).

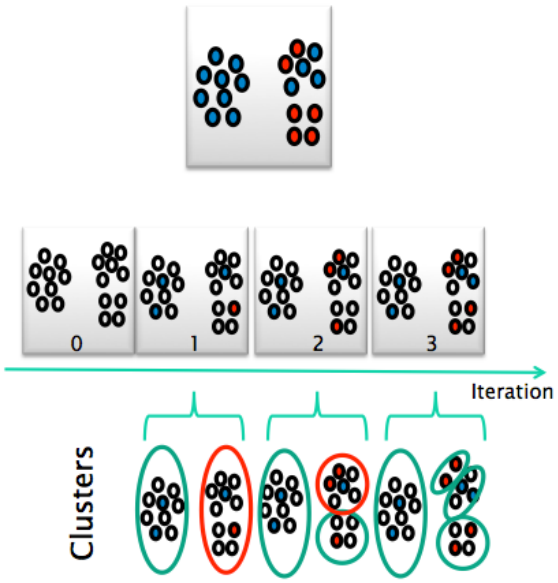

Fig. 5. Scheme of the performance of Dasgupta and its clusters division on each iteration.

This approach lets us be able to guess the majority label in each of the clusters in an instant of time, and therefore having a good approximation of the labels from all the samples without having to label them one by one. Even though, if the label on a cluster is not as clear as it should, we can continue branching the tree and lowering its level. At

this point is where the most valuable advantage of Dasgupta arise, the ability of calculating a bound that can be used as a measure of purity or uncertainty for each cluster.

As we can see in Dasgupta's algorithm (Algorithm 1), which we will analyse deeply as follows, its performance relies in two basic actions that takes on each iteration:

i) Calculation of scores (or bounds) that determine the purity or uncertainty of the set of elements that belong to each of the current clusters.
ii) Query elements for the user to label correctly, which will be used by the algorithm to recalculate the bounds.

Let us introduce the following notions:

- T = the hole hierarchical tree.
- T_v = node v and subtree hanging under it.
- $\mathcal{A}$ = admissible (node, label) pairs.
- w_v = weight of a node v.
- $w(P)$ = sum of weights of the pruning P.
- $p_{v,l}$ = fraction of label l in node v.
- $L(v)$ = majority label in node v.

Input: Hierarchical clustering of n unlabeled points; batch size B
$P \leftarrow \{\text{root}\}$ (current pruning of tree)
$L(\text{root}) \leftarrow 1$ (arbitrary starting label for root)
for time $t = 1, 2, \ldots$ until the budget runs out **do**
  for $i = 1$ to B **do**
    $v \leftarrow \text{select}(P)$
    Pick a random point z from subtree T_v
    Query z's label l
    Update empirical counts and probabilities $(n_u(t), p_{u,l}(t))$ for all nodes u on path from z to v
  end for
  In a bottom-up pass of T, update $\mathcal{A}$ and compute scores $s(u,t)$ for all nodes $u \in T$ (see text)
  for each (selected) $v \in P$ **do**
    Let (P', L') be the pruning and labeling of T_v achieving scores $s(v,t)$
    $P \leftarrow (P \setminus \{v\}) \cup P'$
    $L(u) \leftarrow L'(u)$ for all $u \in P'$
  end for
end for
for each cluster $v \in P$ **do**
  Assign each point in T_v the label $L(v)$
end for

Algorithm 1. Cluster-adaptive active learning.

In order to better understand the calculation of the bounds, we have to know that:

the error on each P,L pair at the iteration t is estimated by:

$$\widetilde{\epsilon}_{v,l}(t) = \begin{cases} 1 - p_{v,l}(t) & \text{if } (v,l) \in \mathcal{A}(t) \\ 1 & \text{if } (v,l) \notin \mathcal{A}(t) \end{cases}$$

$$\widetilde{\epsilon}(P,L,t) \;=\; (1/w(P)) \sum_{v \in P} w_v \widetilde{\epsilon}_{v,L(v)}(t)$$

and the score or bound s for each of the nodes v at the time t can be calculated by:

$$s(v,t) = min(\widetilde{\epsilon}(P,L,t)) \text{ where } (P,L) \text{ must be an admissible pair in } T_v \text{ at time } t$$

Now, after analysing the working of Dasgupta's Hierarchical Sampling, we will continue with the explanation of the classifiers that we used in the project (KNN and AdaBoost).

2.2 K-Nearest Neighbours (KNN)

The KNN algorithm is one of the simplest Machine Learning methods [16], [15]. Once you have given a pool of classified examples, it answers with a corresponding label for each of the new samples that we pass to it, based on its closest neighbours. That decision is simply made calculating the distance between the new specimen and every one of the pool samples and searching for the K-nearest neighbours in that set. Once it has found those K samples, it gets the majority label among them and sets it as the most probable class for the new variable (Fig. 6).

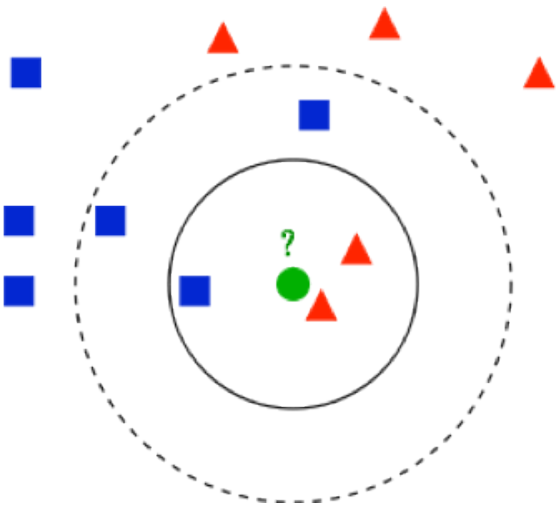

Fig. 6. Green circle represents the sample about to label. If we choose k=3 (solid line) the sample will be labelled as red triangle. If we choose k=5 (dashed line) it will be labelled as a blue square.

The metric used to calculate the distance can be anyone applicable to the sample's features, and its results may vary depending on the type of examples. The parameter K, whose optimal value may vary depending on the data too, must always be an odd number for the disambiguation of the results.

2.3 AdaBoost

AdaBoost, or also called Adaptive Boosting, is a supervised learning algorithm that generates a classifier (Strong Classifier) using the combination of several simpler Weak Classifiers [20].

The behaviour of AdaBoost (Algorithm 2) is based on the refinement of the weights given to each of the features that describe the samples, where all of them will be 1/N at T = 1 and will be changing on the course of the algorithm.

Input : A weak learning algorithm $WeakLearn$, an integer T specifying number of iterations, and N labelled training data $\{(x_1, y_1), \ldots, (x_N, y_N)\}$.

Output : A strong classifier F.

Initialize the weight vector $w_i^1 = \frac{1}{N}$, *for* $i = 1, \ldots, N$.

for $t \leftarrow 1, 2, \ldots, T$ **do**

1. $\mathbf{p}^t \leftarrow \mathbf{w}^t / \sum_{i=1}^{N} w_i^t$.
2. Call $WeakLearn$, providing it with the distribution on $\mathbf{p}^t$; get back a weak learner $h_t : X \rightarrow \pm 1$.
3. Calculate the weight error of h_t: $\epsilon_t = \sum_{i=1}^{N} p_i^t \frac{1}{2} |h_t(x_i) - y_i|$.
4. $\alpha_t \leftarrow \log\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$.
5. $w_i^{t+1} \leftarrow w_i^t \exp\left(\alpha_t \frac{1}{2} |h_t(x_i) - y_i|\right)$, for $i = 1, 2, \ldots, T$.

Output the final strong classifier:

$$F(x) = \begin{cases} 1, & \text{if } \sum_{i=1}^{T} \alpha_t h_t(x) \geq 0, \\ -1, & \text{otherwise.} \end{cases}$$

Algorithm 2. Adaboost supervised learning classifier.

At the first step, a Weak Learner function (defined by the user) decides which is the feature that discriminates the best the different classes.

Then, after calculating the error given precisely by that weak leaner, it recalculates the weights and loops over the process again.

The process finishes when the time T runs out or when the classification error has gotten to 0.

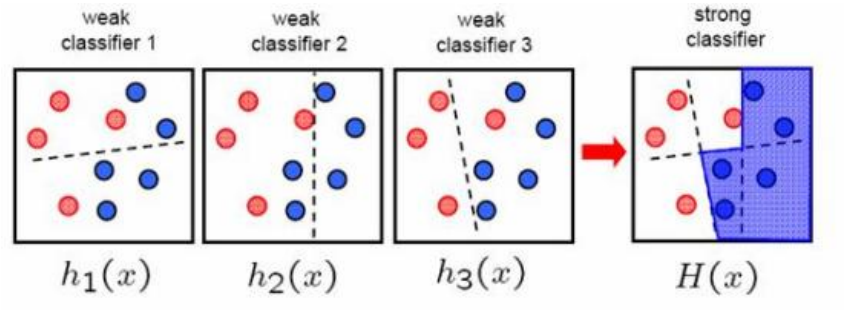

Fig. 7. Example of AdaBoost algorithm applied to a small group of samples. Each of the Weak classifiers is represented by a dashed line. The resulting Strong classifier is represented in the last picture as the combination of all the Weak classifiers.

Now, after the classifiers review, we will continue analysing the dimensionality reduction algorithms (PCA, LDA and LMNN).

2.4 Principal Component Analysis (PCA)

Principal Component Analysis [7] is an unsupervised technique that allows to significantly reduce the dimensionality of a multidimensional set of variables converting the original variables' space into a new one with less information but keeping the relevant part of it.

The main goals of PCA are:

i) Extract the most relevant information of our source data.
ii) Compress and simplify the data dimensionality.
iii) Analyse the information of our samples searching for the projections with higher variance among the data.

The basis of this technique is the calculus and sorting of each of the components of our data, taking as the first column the one with more variability, which will be named Principal Component 1 – PC1. And following with the rest of the data but always taking into account that the PC_i must be orthogonal to PC_{i-1} and the variance of the data in each component must be in decreasing order:

$$PC_{i-1} \perp PC_i \text{ and } var(PC_{i-1}) > var(PC_i)$$

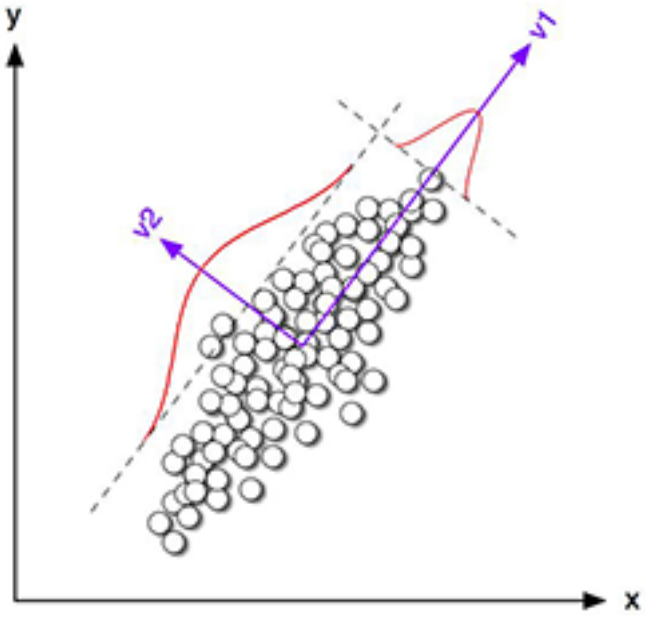

Fig. 8. Image illustrating the choosing of projections for each principal component in PCA. The projection in v1 would be the PC1 and the orthogonal projection v2 would be PC2.

Furthermore, the principal components obtained from PCA will always be linear combinations of the original variables, meaning that we will have to find the eigenvalues and eigenvectors of the covariance matrix of our original data and only keep the eigenvectors with eigenvalues bigger than a given threshold. This way we will accomplish the objective of eliminating the irrelevant data and reduce their dimensionality.

Describing it mathematically, we must find:

$$cov(M) * V = V * D$$

Given that M are our samples. M's rows are observations and M's columns are variables, V are their eigenvectors and D are their eigenvalues.

And after keeping only the most informative eigenvalues V':

$$M' = (M - mean(M)) * V'$$

knowing that M' are our samples in the new PCA space. However, PCA is projecting data in a new feature space without taking into account difference in classes of data. The technique that looks for a projection so that compacts as possible the data of the same class and separates as much as possible the data from different classes is classed Linear Discriminant Analysis.

2.5 Linear Discriminant Analysis (LDA)

LDA, also called Fisher-LDA [12], [13], [14], is a supervised machine learning algorithm that converts the space of some given samples trying to maximise the distance between the different classes. Comparing it with PCA, one of the main utilities of Linear Discriminant Analysis is the class-dependant dimensionality reduction of the input data. Although, there are some important features in LDA that make these methods differ:

i) We must have a set of previously labelled samples.
ii) The number of output dimensions of the data will always be C – 1 given that the number of different classes is C.
iii) LDA always tries to choose the projections in which the distance between the different classes is bigger.

In the following figure we can see an example of how it works:

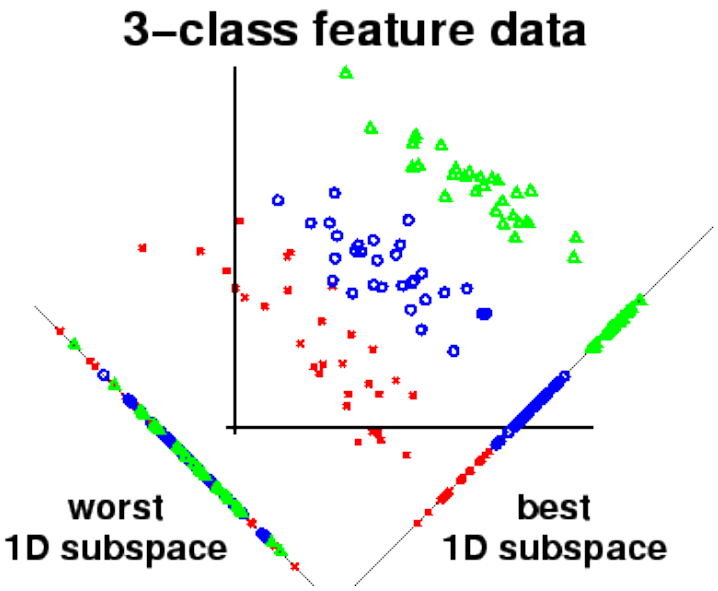

Fig. 9. Example of a good and a bad LDA projection given some samples of 3 different classes.

Apart from that, we must know that Principal Component Analysis is usually applied before using Linear Discriminant Analysis for its better performance working with a previously dimensionality-reduced space.

In order to take into account the already labelled samples in the space transformation process, LDA introduces two specific matrices that must be calculated from the labelled data. The between classes scatter matrix S_B, describes the variance of the samples

between all the different classes, and the within classes scatter matrix S_W, describes the variance of the samples within the same class.

$$S_B = \sum_{i=1}^{c} N_i(\mu_i - \mu)(\mu_i - \mu)^T$$

$$S_W = \sum_{i=1}^{c} \sum_{x_k \in X_i} (x_k - \mu_i)(x_k - \mu_i)^T$$

where:

- C = number of classes.
- N_i = number of elements in class i.
- μ_i = mean of elements in class i.
- μ = mean of all elements.
- X_i = set of elements in class i.
- x_k = element k from class Xi.

One of the environments in which LDA works very well is when the samples are normally distributed [25].

Following with the presentation of the active learning algorithms we used, LMNN is another of them. Even though its performance is based on the operation of KNN as we will explain as follows.

2.6 Large Margin Nearest Neighbour (LMNN)

The K-Nearest Neighbours method has proved to be very beneficial, but when combined with other techniques it has shown a much better performance [17], [18], and that is why some researchers decided to develop the LMNN [15], [19]. LMNN, based on the method used by KNN, offers us an active learning useful tool.

The operation of this algorithm resides in the same principles than the LDA and PCA methods. It introduces a different method for transforming the space of our data so as to separate the samples of the several classes, narrowing the distances between the samples of the population of the same class and enlarging the distance to the others. For the LMNN to be more effective, it is always recommended to previously apply a dimensionality reduction method like PCA to the data [15] (see Fig. 10).

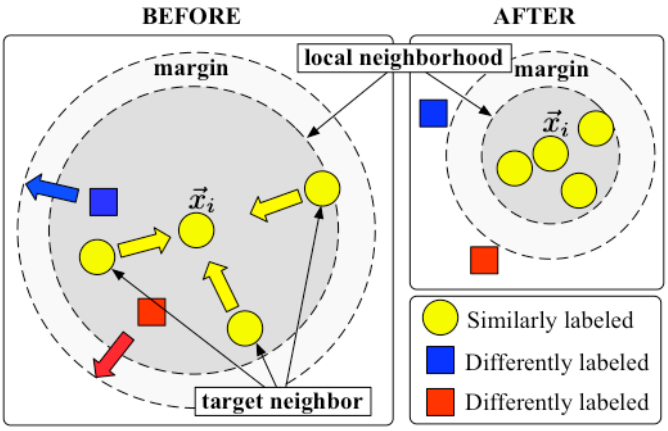

Fig. 10. Example of the space transformation applied by LMNN to a small group of samples.

Being more precise, LMNN uses the Mahalanobis distance to calculate the interval between the samples. And the algorithm linear transformation uses two terms in order to try to minimize a cost function, where the first of the two terms penalizes large distances between examples from the first class and the second one penalizes short distances between examples of different labels:

$$\varepsilon(L) = \sum_{ij} \eta_{ij} D_{ij} + c \sum_{ijl} \eta_{ij}(1 - y_{il}) \left[1 + D_{ij} - D_{il}\right]_+$$

where:

- $y_{ij} \in \{0, 1\}$= binary matrix that determines whether a pair $\vec{y}_i$, $\vec{y}_j$ has the same label.
- $\eta_{ij} \in \{0, 1\}$= binary matrix that determines if $\vec{x}_j$ is one of the K samples chosen as nearest neighbours of $\vec{x}_i$.

- $\text{sample, label training pairs} = \{\vec{x}_i, y_i\}$
- $D_{ij} = ||L(\vec{x}_i - \vec{x}_j)||^2$ = squared distances between samples.
- Term $[z]_+ = max(z, 0)$ determines the standard hinge loss.
- c is a constant bigger that 0 set by cross validation.

Focusing at last in the final algorithm used, we will talk about the clustering technique K-Means.

2.7 K-Means

K-Means (first introduced by MacQueen [21]) is a very simple and non supervised learning algorithm, whose operation lies on the minimization of the distance (or the variance) between K previously set points and the samples around them. These points act as the centre of mass of the population (centroid) around it, and its position and the samples that are attached to it vary on each iteration.

The cluster for each sample on each iteration is calculated by:

$$C_j = \underset{i=1}{\overset{K}{arg\,min}} ||x_j - \mu_i||^2$$

And after all the assignments have been revised, the mean on each cluster is recalculated.

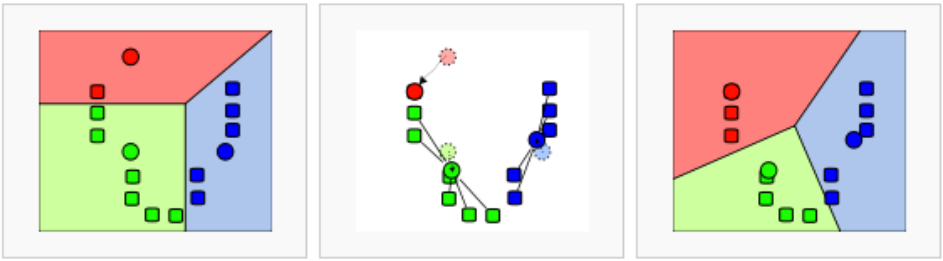

Fig. 11. K-Means example of operation. Squares are samples and circles are centroids. Once the centroids have been initialized and each sample has been assigned to its closes centroid (left), the means are recalculated and the centroids repositioned (center). The operation is repeated until the samples' closes centroid does not change (right).

Once those assignments no longer change, we assume that the algorithm has converged and we have the output (each point assigned to a cluster Ci between all the K clusters) of the algorithm.

The initialization of the "center of masses", which is commonly randomly chosen, is an important point in the method, since it will determine its final result.

Now that we have overviewed all the algorithms that we used, in the following section we will detail how we implemented them and their precise use in this project. Reviewing also the rest of the modules that we needed to implement for the developing of this application.

3. The Problem of Nutritional Object Labelling and Recognition

3.1 Application General Flowchart

To start, as a summary of the whole implementation, in the following flowchart (Fig. 12) we can see a schematic description of how the Active Learner application works. Including from the photo capture to each cycle of training of our Online Learner going through each of the modules that will be explained in this section.

After finishing the pre-processing of all the images taken during the day, we get a group of folders (or sets) that we have to label one by one. For each of them we repeat the same loop. We load the prototypes chosen (and the Single Linkage clusters if that is the Active Learning method chosen), do the initial screening with the previously trained Online Learner and validate all the images selected by the initial screening.

Then starts the Dasgupta's main structure (adapted using the Query and Cluster boxes), labelling groups of images chosen by the program depending on their bound. Each iteration recalculating the bound and dividing the clusters using the Active Learner chosen by the user if we selected the random images (Queries box) or simply recalculating the bound of the images (Cluster box).

Finally, recalculating the prototypes and the Online Labeller (classifier) before starting with the next set of images.

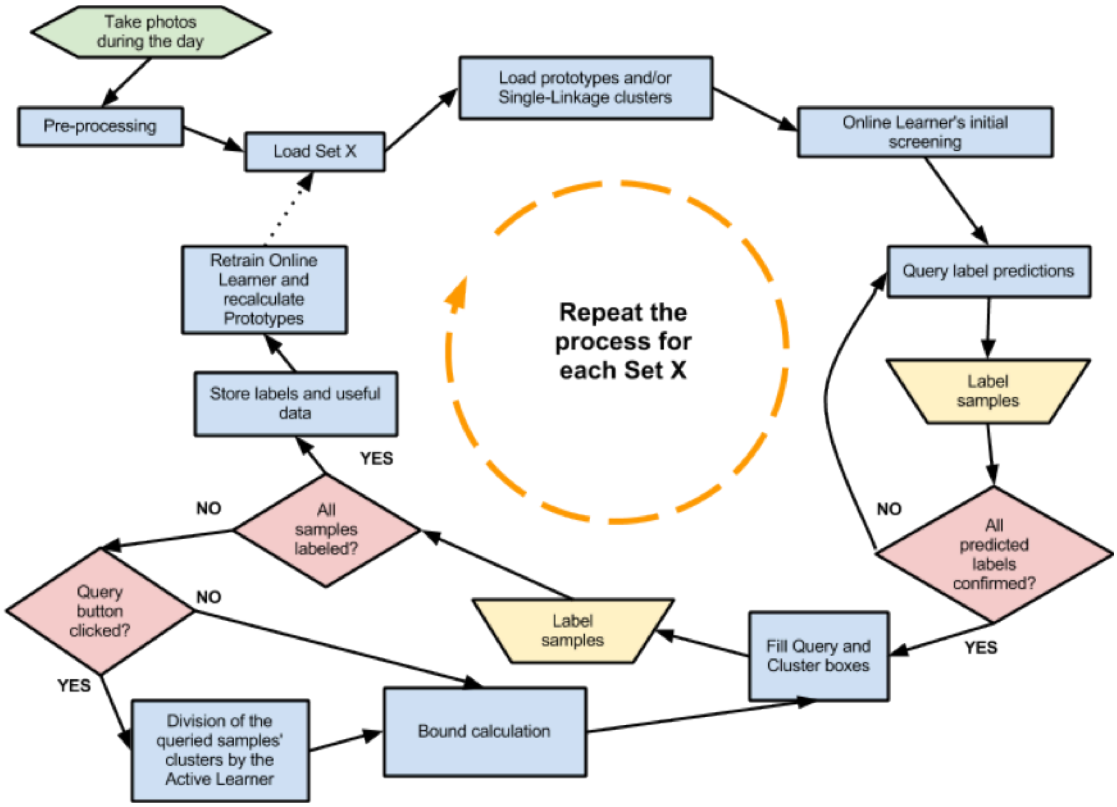

Fig. 12. Flowchart representing the whole Active Learner's labeling process.

3.2 Dishes Images Compilation

Before any initial programming decision or features definition directly related to the pictures was done, we had to try the SenseCam (examples of images in Fig. 13), see how it worked and take as much images of dishes as we can. Apart from that, the development of the skeleton of the interface and the Dasgupta's algorithm could be made in parallel.

Fig. 13. Examples of images taken by the SenseCam in different situations and locations.

Once we had the first sets of images, we decided that as a good way of increasing the number and variety of dishes, we could use some pictures from any online free database. And finally, we added about 1.200 extra images from www.image-net.org on which all of them appeared at least one dish (examples of ImageNet images on Fig. 14).

Fig. 14. Examples of images from ImageNet, all of them containing dishes.

3.3 Sliding Window and Pre-processing

Now that we had a huge set of images to work with, we had to find a way of processing them to accomplish our aim: finding the regions in the photos taken by the SenseCam and ImageNet on which there was any kind of dish.

To do so, we developed a pre-processing function (mainPreprocessing.m), where we would use to do all the necessary tasks to have the image ready for our active labeller, which main tasks are, in the following order:

1. Select a folder and range of images to pre-process (mainPreprocessing.m).
2. Rename images to have the format (00001.jpg, 00002.jpg, etc.) (renameImages.m).
3. Crop the images in smaller sub-images in order to have as many images of dishes as possible for the algorithm development, and to have all the dishes of different sizes and positions correctly centered (slidingWindow.m). We have to establish the sub-images proportions, rescaling factor and slide's overlap, for which we set on most images (resulting on 200 sub-images per image):
    - Initial sub-image scale: 75%
    - Final sub-image scale: 25%
    - Jump between scales: 10%
    - Difference (100 - overlap) between images: 10%
4. Divide the great number of pictures resulting from step 3 in several folders (as maximum about 4.000 images per folder) to avoid overwhelming the RAM memory when calculating the Hierarchical Clustering Tree with Single Linkage (see section 3.5 Dasgupta's Active Learner) of each active labelling session (dividePhotoGroup.m).
5. Rescale the definition of the already cropped images for the HOG to work better when dealing with the image gradients (see section 3.4 Sample Features and Classes Selection) (rescaleImagesCropped.m).
6. Extract the corresponding features of each image and save them all together on each set in the file features.mat (extractFeatures.m).
7. Extract the Euclidean distances between all the image features for the Single Linkage algorithm applied when calculating the Dasgupta's Hierarchical Clustering Tree (extractEuclidDist.m).

On Figure 15 we can see some examples of SenseCam images after the pre-processing procedure.

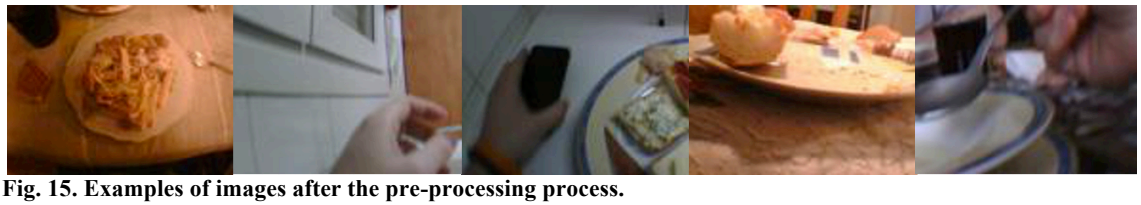

Fig. 15. Examples of images after the pre-processing process.

3.4 Sample Features and Classes Selection

One of the most important decisions we had to take for us to have a good definition of our classifier, was the number of classes and distinctions between them. Knowing that what we are looking for are the dishes, we could establish a "No Dish" and a "Dish" label and that would be enough. Although, seeing the images that we obtained from the crop pre-processing, it could be logical, due to the closer similarity to a Dish rather than a No dish, to add a "Semidish" label for all the images on which a dish is being shown only partially or from a distant point.

Now that we had decided the different classes to use, we had to narrow the boundaries of the different labels, so we established that **approximately** each of the different classes had to accomplish the following:

Fig. 16. Sample labeled as a Dish due to its maximum 5% out of sight.

- **Dish**: A dish is on the centre of the image, its elliptical (or rounded) shape is completely visible or has as much as a 5% of it out of visibility (see Fig. 16).
- **Semidish**: At least a 20% of the elliptical shape of a dish is visible on the image and its most centred point must be close to the centre of the image. Or a distant but clearly visible dish appears in the picture and is near the centre of it.
- **No Dish**: any other image, including a close dish whose shape is indistinct or we only see the food.

Another crucial step to always take into account when someone is developing an active learner or any Machine Learning method is the choice and processing of the features that will describe and distinguish the objects of each of our classes. So as to have the most informative and not redundant information that we had within our reach, we chose the following ones:

- **Time (6 components)**: second, minute, hour, day, month and year when the photo was taken. This information can be helpful because considering that all the photos processed by the active labeller will be from the daily routine of a very same person, he/she will have a certain hours on which the probability that he/she will be eating will be higher than others. Due to the unrelated time information of the ImageNet images, we decided to set to 0 their Time features and do not use them with the Online Labelers/Learners (see section 3.6 Online Learner).
- **RGB (3 components)**: the average label of grey on each of the channels: red, green and blue. It is useful for example, knowing that the most part of the dishes are white.
- **HOG (81 components)**: histogram of gradients of the image. Due to the clearly distinctive shape of the dishes (similarly as when creating a face recognition application), this data can provide us the most powerful information about the structure of the object. The HOGs are calculated dividing an image in different rectangles (3 by 3 in our case) and calculating the intensity of the gradients in a predefined number of different angles (9 in our case).
- **Position (4 components)**: x, y, width and height of the cropped image in relation with the original one. Considering that when we are eating we normally have the dish in front of us, it is very important to take into account an object that seems a dish and is on that position.

To sum up we have 94 different features that will try describe as good as possible our different labels, but before finishing the features extraction we must be sure that our algorithms will not overestimate any of them because of a usual higher value than the rest of the characteristics, and to do so we normalized all the features setting them into a range of 0 and 1.

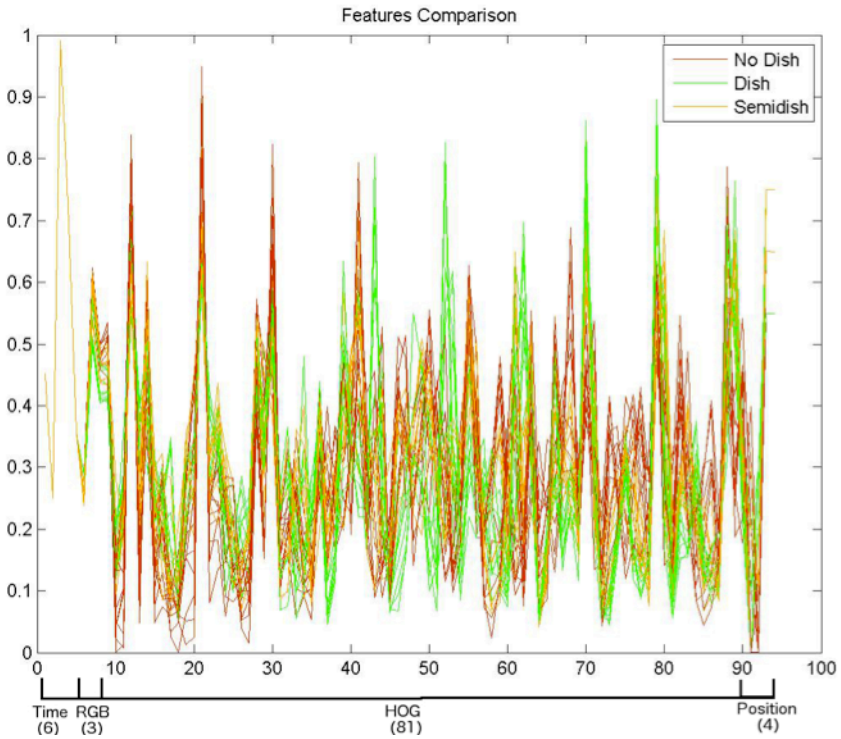

Fig. 17. Example of the chosen features with 10 samples from each class.

In the following figure we can see an example of the features applied to each of the classes (10 samples from each class).

Once we have our data prepared for their usage and classification, we have to continue with the implementation of all the algorithms used for that labelling. Starting by the skeleton of our application, Dasgupta.

3.5 Dasgupta's Active Learner

The basic implementation in this project of the active learner algorithm based on Dasgupta's method (see section 2.1 Dasgupta's Cluster-Adaptive Active Learning) comes from the authors of [22], [23] and [26], which implemented this algorithm for an Active Learning application used for labelling Wireless Capsule Endoscopy (WCE) images. After that we adapted that implementation into a new interface (see section 4. Final Application for Active Labelling of Food-Related Objects) for being able to label a wider variety of image types, different number of classes (initially it was restricted to a binary classifier) and the rest of the features of this project.

The implementation provides two different approaches for the user to proceed with the labelling process on each iteration: 1) Random samples from all clusters (named Query). 2) Samples from the purest cluster (named Cluster).

The combination of the scheme presented by Dasgupta (see section 2.1 Dasgupta's Cluster-Adaptive Active Learning) and the two previously named methods of queried samples can be outlined with the following algorithm:

Input: Set of unlabelled images and their corresponding features.
1: Calculate Hierarchical Cluster Binary Tree (HCT) using Single Linkage clustering based on the Euclidian distance between the samples' features.
2: Initialize an empty tree structure T for keeping track the pruning followed, the labelled and unlabelled samples that are in each cluster and their purity measure.
3: Query the first N random samples. (1) Query).
4: Save labels and set samples to "labelled" and increment clicks.
5: Pull apart the first cluster based on the HCT.
6: **While** there is any unlabelled sample.
7: Calculate bounds (purity of each cluster) and the most probable class for each one (which will be temporary set until user's approval).
8: Query N random samples on a window (1) Query) and the first M samples from the purest cluster on another (2) Cluster) with their guessed labels.
9: Save labels, set samples to "labelled" and increment clicks (only from the chosen window's samples).
10: **If** user selected 1) Query
11: Pull apart the clusters where each of the queried samples belonged to based on the pre-calculated HCT.
12: **End**
Output: Labelled images.

In order to measure the effort of the user for labelling all the samples, every time that the active labeler master changes one of the guessed labels, either he/she is using the Query or the Cluster window, we increment the number of clicks by one.

Now, in order to extract the needed information from the images compiled in 3.1 Dishes images Compilation, and being able to feed the Active Learning algorithm with them, we have to apply some pre-processing techniques.

3.6 Online Learner

The main target to accomplish on the project was the implementation of an Active Learner for the labelling of the images. Even though, we realised that if we wanted to exploit all the information acquired on previously labelled sets (on previous days or previously labelled sets from the same day), it was important to add an "online learner" to the active learner.

Which means that we should build some kind of classifier that could take all the already labelled images to automatically (without any important help from the user) guess a wide part of the labels of the next set. This way, adding a likelihood measurement to the classifier in order to have a high level of success (precision), we could present the automatically guessed images to the user at the beginning of the following labelling for him/her to spend less time and less clicks to correctly classify them.

Considering the time left for the project and the results that we could obtain, we decided to implement two different online learners, a KNN-based (see section 2.2 K-Nearest Neighbours (KNN)) and an AdaBoost based (see section 2.3 AdaBoost) one. To this aim, we developed two different methods of classification (see section 5. Results for information about performance):

A single classifier that distinguishes between our three classes (Single Classifier):

a. NP vs P vs SP*

A simple cascade classifier that does the classification in two steps (Combined Classifiers):

a) NP vs (P + SP)*
b) P vs SP*

The comparisons (using different testing modules) on this different approaches and any other possible parameter like the balancing of the samples, the K or the number of rounds in AdaBoost values will be further analysed in section 5. Results.

*NP = No Dish, P = Dish, SP = Semidish.

At this point a natural question arises: how to use information from one labelled tree to the next one in order to complement the help from the online learners? And our proposed strategy is the prototypes.

3.7 Prototypes

We introduced a novel strategy to improve the performance of the active labeller by the usage of "prototypes". The prototypes introduction would be another way besides the online learner to take advantage of the samples previously labelled to label the next sets. But this way we can use them to help the labelling along the process and not only at the beginning of it like the online learner does.

The basis of the prototypes introduction is simple, each of them should represent a different set of all the labelled population (not necessarily the same number of samples) and must have the same features format.

Based on these simple requirements we developed two different types of prototypes (see Fig. 18):

- **OnePerClass** prototypes: they are built at the same time that the Single Classifier (see section 3.6 Online Learners), and act as the representatives of the same samples that are used to train it. There is only one prototype for each of the classes of the application (in this case NP, P and SP).
- **OnePerLeaf** prototypes: they are built at the ending of each labelling session. Each of them represents one of the leaves that are generated as a result of the tree used by the Active Learner. These leaves must always be purified so as to have only samples from one of the classes.

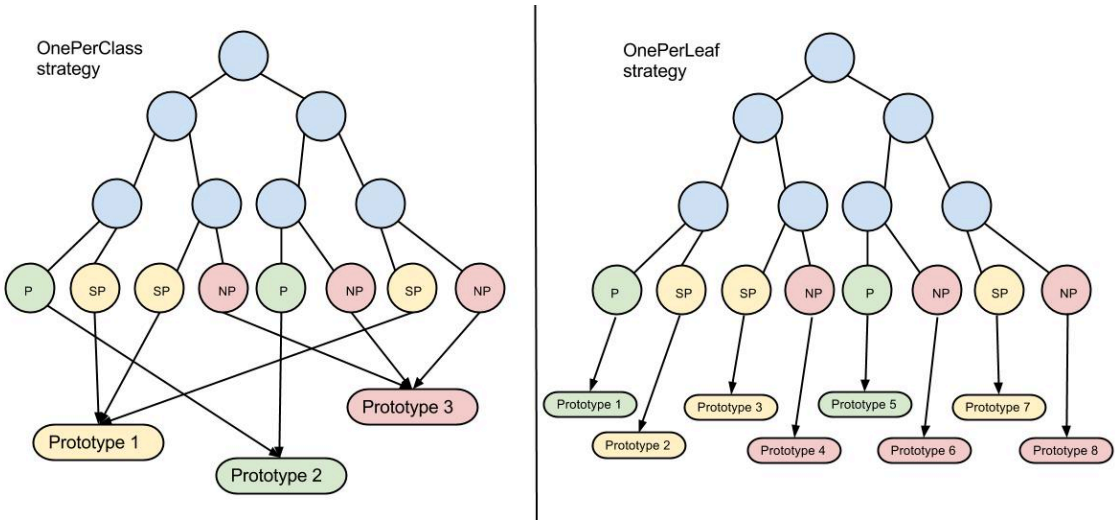

Fig. 18. This figure shows how each type of prototypes is created after the complete labelling of a set. Once we have opened the whole tree and the leafs have been purified (all samples are from a single label) we can create the OnePerClass (left) prototypes or the OnePerLeaft (right) prototypes.

These prototypes, regardless of how they are created, store the same type of information and are calculated in the same way. Their attributes are:

- Label: it identifies the label of its representative samples.
- Features: a set of features like any other sample but calculated as the median of its representative samples.
- Number of samples: number of samples (or weight) that the prototype represents.

Once we have the prototypes created, they always behave in the same way, during the labelling process they work like any other sample except for three reasons. First of all they are marked as "labelled" (see steps 4 and 9 from the algorithm of the section 3.5 Dasgupta's Active Learner) right from the beginning of the labelling, they are never stored on the result of the labelled set, and what is the most important point and helps us in the process: the number of their representative samples acts as a weight on the calculus of the bound (cluster purity).

We introduce the following weights for the new samples:

$$W^{i}_{sample} = \frac{1}{(N + \sum_{j=1}^{M} N_{pj})}$$

and a higher weight for the prototypes:

$$W^{j}_{prototype} = \frac{N_{pj}}{(N + \sum_{j=1}^{M} N_{pj})}$$

where:

- N = number of samples in this set.
- M = number of prototypes that we are using.
- N_{pj} = number of samples that are represented by the prototype j.

This procedure allows the algorithm to give more importance to the group of samples that the prototypes represent rather than any single new example.

The last, but not the least, strategy that we implemented in order to improve the performance of our active labeller was, as we can see in the following section: the use of different active learners apart from the Single Linkage used in the basic Dasgupta's method.

3.8 Active Learners

The last, but not the least of the features that we implemented were the two new Active Learners. These new learners, which still follow the skeleton of Dasgupta's Active Labeler, would change the way in which the clusters of the tree are formed.

Until this point we used the Single Linkage clustering, based on the Euclidean distances, for initially creating the binary tree that would be pruned but never modified during the labelling process. But considering that the data might be better divided using other heuristics, we implemented the LDA and the LMNN algorithms. Both of them cause a change on the way the tree is divided but not on the way the purity of the clusters is calculated.

To follow with the original scheme, we added an extra script that would do the same as the original one (divide_partition_das2.m) but that additionally would apply the LDA or the LMNN depending on the one that the user chose.

Each of the corresponding functions applies the same techniques (see section 2. Methodology) and the same algorithm:

Input: Features of the seen (we have the real labels) and the unseen elements from a cluster.

1: Apply local balanced KNN to predict the labels from the unseen samples (KNN_for_tree.m).
2: Assign the temporal labels to the unseen elements.
3: Balance all the elements.
4: Apply PCA( ).
5: **If** selected_LDA.
6: Apply LDA( ).
7: **ElseIf** selected_LMNN.
8: Apply LMNN( ).
9: Apply Kmeans with nClusters = k = 2 to continue following the binary tree structure.
10: Forget labels from useen samples.

Output: Cluster divided in two new clusters by LDA or LMNN.

3.9 Simulation Module

Apart from the previously analysed modules implemented, we added a simple simulation agent so as to be able to do automatic tests and, over all, for the massive Forest Tests that we will see in section 5.9 Forest Labelling Simulation and 5.10 Clicks Vs Time Simulation Results.

This module can only be activated if the selected video set has been previously labelled, this way it will only have to change the wrong labels previously selected by the user on another session, in one of the two available labelling windows (Queries or Cluster, see section 4.1 Main Window). The selection method of the window lies on a very simplistic rule: it chooses the window that has less wrongly labelled samples.

4. Final Application for Active Labelling of Food-Related Objects

4.1 Main Window

If we want any application to really be useful for the final user we must always create a good interface for them to interact with. One of the main features that we considered ours had to possess, was the easy accessibility and visibility of the most used features and the easiness to know at which point of the labelling process we are. Fig. 19 shows how we managed a great part of them.

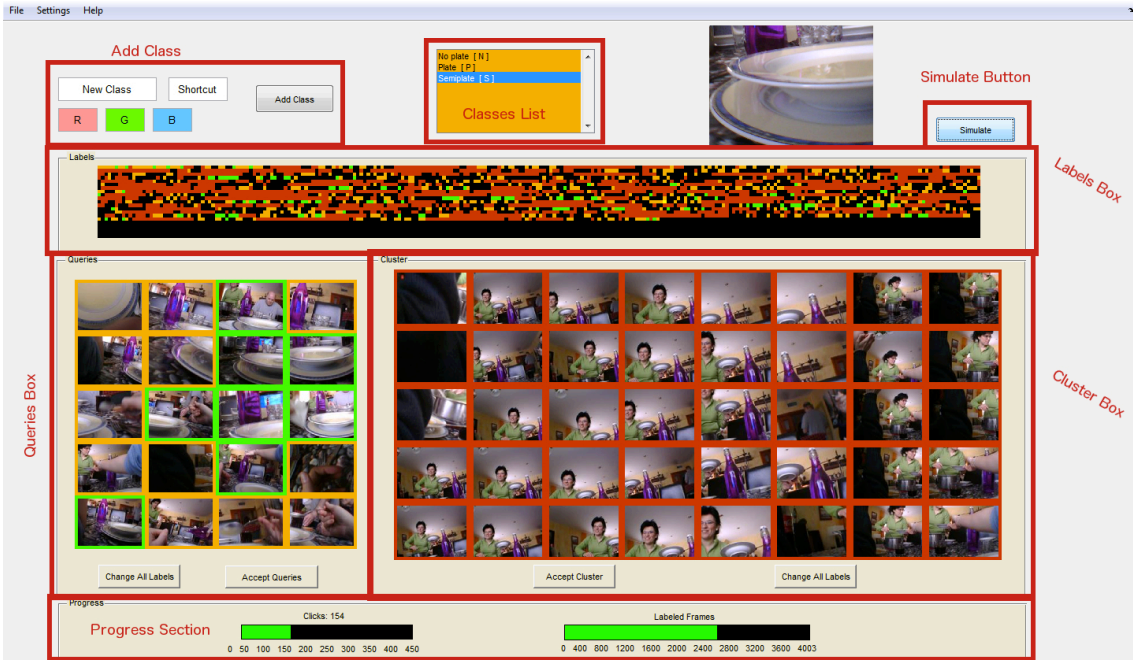

Fig. 19. Main application window naming its different zones.

Starting from the top-left corner and going to the right-bottom one, we will overview its different features and ways of visualizing the data.

First of all, the **top menus** will simply let us load a set of images (File > Load), change the application settings (Settings > Edit, see section 4.2 Settings) and review how to use some of the features (Help, see section 4.3 Human-App easy interaction features). For loading a new set we must be sure that its name is formatted correctly and their images have already been pre-processed (see section 4.4 Pre-processing module).

Later, once we have loaded a set of photos, the program will automatically load the classesLabeling.mat file (see classes_path in Table 1), which will store the information of our labels. In case that the file does not exist, it will automatically create one with standard classes, which will have to be manually modified in Matlab if we want to erase some of its classes. But if we want to add new classes we can use the **"Add Class"** section, with which we can set a name, a colour and a shortcut to the new class.

Right under the "Add Class" tool we can find the **"Labels Box",** which allows us to review the images that we have already labelled (see section 4.3 Human-App easy interaction features), checking their labels and which photos correspond to.

At the right of the image frame from the "Labels Box" we have the **"Simulate Button"**, which lets us simulate the labelling of a single iteration (only in case that our set of images were previously labelled by the user) and can only be used it we deactivated the automatic labelling before loading the set (see section 3.9 Simulation Module).

At the centre of the window we can find the **"Queries Box" and the "Cluster Box"**, which clearly are the main sections and the ones that are used more frequently during the labelling process (see section 3.5 Dasgupta's Active Learner). Each of the images that appear in them is surrounded by a coloured border, which identifies its label and can be checked at any moment on the top-center widget **"Classes List"**. Once we have chosen the box we will accept, we have to change the wrongly guessed labels, and to do so we only have to click the mouse's left button, which will alternate all the available labels one after the other always on the same order (see section 4.3 Human-App easy interaction features for using Shortcuts). Another way of changing the labels is the "Change All Labels" buttons, which will do the same trick but for the whole box. Once we have switched all the image's labels that we wish, we only have to press the corresponding "Accept" button and the active labeller will do the rest.

During the acceptance process of the Online Learner's predicted labels (see section 3.6 Online Learner), we will also be able to see the likelihood given by the classifier to each image on the top-right corner of each of the Cluster Box's samples.

And finally, in the bottom **"Progress Section"**, we can find the clicks and the labelled images bars, which can be used to check at any moment how many clicks we have done and how many images have already been labelled.

4.2 Settings

Due to all the different algorithms and ways of configuring the active learner, the application has a considerable amount of different parameters and settings to tune. Although it is essential to be able to change their values for getting the best performance possible, we considered that presenting only the ones that are more usually modified (see Fig. 20) to the average user, would be enough.

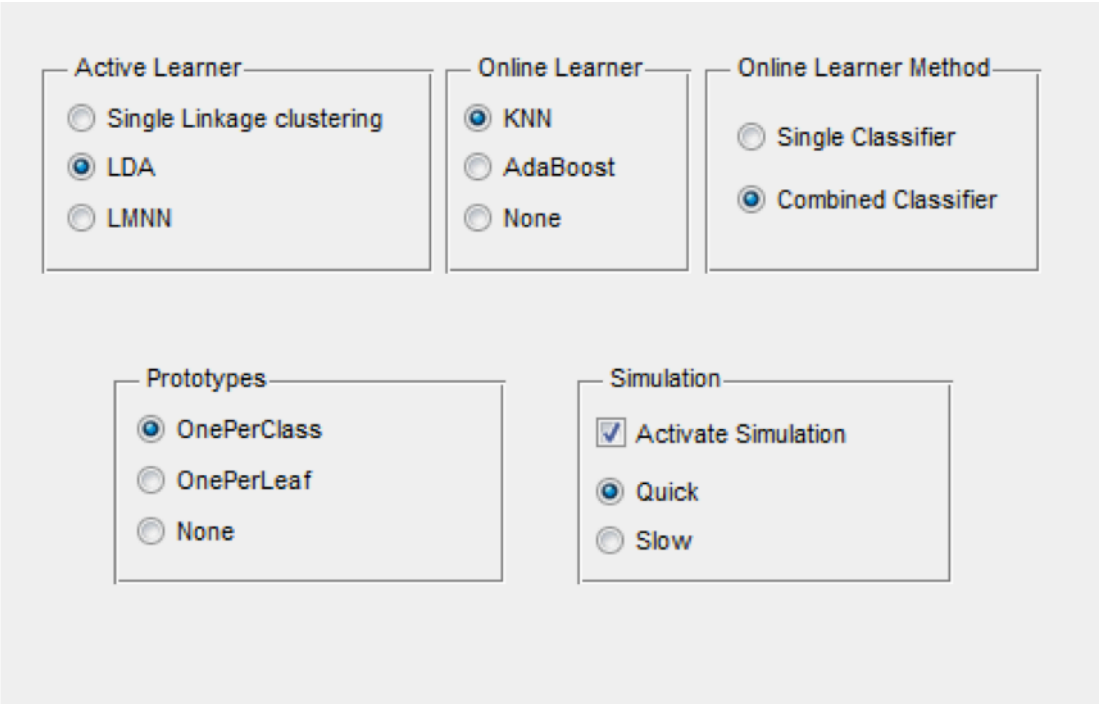

Fig. 20. Snapshot of the settings window.

With this settings window, we can change between the different Active and Online Learning methods, the different Prototypes available, and we can select if we want to automatically simulate the labelling of the images set that we will choose afterwards (only applicable in case that the chosen set has been previously labelled by the user).

If the user is more expert and wants to tune more specific parameters, then we can go to the main application file (gui_intestinal_labeling.m) to switch more specific parameters. In the "GLOBAL VARIABLES" section of the code are all the values that can be modified to change the application performance.

Below there is a summary table (Table 1) about all the modifiable variables:

Name	Usage	Allowed Values	Default
folder_path	Absolute or relative path to the set that we will label.	Any path to a valid set folder.	" "
outerBorder	Border width in pixels of the photo frames indicating its label.	Any value in px.	10
heightLabelAxes	Height of the coloured rectangles in the Labels box.	Any value in px.	64
widthLabelAxes	Width of the coloured rectangles in the Labels box.	Any value in px.	768
heightClicksAxes	Height of the progress bar that indicates the clicks done.	Any value in px.	20
widthClicksAxes	Width of the progress bar that indicates the clicks done.	Any value in px.	230
heightNumLabeledAxes	Height of the progress bar that indicates the already labeled photos.	Any value in px.	20
widthNumLabeledAxes	Width of the progress bar that indicates the already labeled photos.	Any value in px.	320
imgsFormat	Extension of the images that we will label.	Any image format readed by Matlab.	".jpg"
rowsQueries	Number of rows shown in the Queries box.	Any natural value.	5
colsQueries	Number of columns shown in the Queries box.	Any natural value.	4
rowsClusters	Number of rows shown in the Cluster box.	Any natural value.	5
colsClusters	Number of columns shown in the Cluster box.	Any natural value.	8
classes_path	Relative path to the folder where the file classesLabeling.mat is stored.	Any real path to a well formated file.	".."
activeLearnerType	Active Learner algorithm that we will use.	{ 1 = HCT, 2 = LDA, 3 = LMNN }	2
classifierType	Online Learner algorithm that we will use.	{ 0 = AdaBoost, 1 = KNN }	1
onlineLearnerType	Defines if we will use a single or a combined online learner.	{ 1 = Single Classifier, 2 = Combined Classifier }	2
prototypesType	Type of prototypes that we will be using.	{ 1 = OnePerClass, 2 = OnePerLeaf.	1
PCAthreshold	Determines if a PCA's eigenvector is relevant (see section 2.2).	Any real value.	0.001
LDAthreshold	Determines if a LDA's eigenvector is relevant (see section 2.3).	Any real value.	0.001
numNearestNeighboursKNN	Value K of the KNN algorithm.	Any natural and odd value.	15
numNearestNeighboursLMNN	Value K of the LMNN algorithm.	Any natural and odd value.	3
nClusters	Value K of the K-Means algorithm.	Any natural value.	2
usingPrototypes	Says whether we are using prototypes or not.	Boolean	true
usingOnlineLearner	Says whether we are using the online learner or not.	Boolean	true
th_No_Plate	Minimum likelihood for a sample NP to be accepted by the initial screening.	Any real value between 0 and 1.	0.7
th_Plate_Semiplate	Minimum likelihood for a sample P or SP to be accepted by the combined initial screening.	Any real value between 0 and 1.	0.7
th_Plate	Minimum likelihood for a sample P to be accepted by the initial screening.	Any real value between 0 and 1.	0.9
th_Semiplate	Minimum likelihood for a sample SP to be accepted by the initial screening.	Any real value between 0 and 1.	0.8
simulate	Activates or deactivates the simulation agent.	Boolean	false
simulationMode	Determines if the simulation will be quick or slow for us to see the label changes.	{ 1 = quick, 2 = slow }	1
waitTime	Only used with slow simulation. Time span (seconds) between each label change.	Any value grater than 0.	1
silent	Determines if the agen will show on the console which button pushes at each iteration.	Boolean	false

Table 1. Information about all the modifiable variables in the main application (gui_intestinal_labeling.m).

4.3 Human-App Easy Interaction Features

Leaving aside the basic features of the application used for its main functionality, labelling images, we tried to introduce some other aspects that could improve and make easier the user experience and the human-app interaction in general:

Shortcuts: Each of the created classes must have a unique key assigned on the keyboard (No Dish -> N, Dish, -> P and Semidish -> S). This key will be useful at any point of the labelling process, since the user will be able to use the unique key combination, Shift + Shortcut, to change the label of the image that we are clicking with the mouse's left button from its present one to the one referenced by the shortcut. This combination can be used too, when clicking the "Change All Labels" button.

Bigger Axe Window: Due to the necessity of having a great number of images on the window at the same time to label as quickly as possible, sometimes, the shapes and the objects that appear in them are unclear. When this happens the user can right-click the desired image to get a bigger representation of it (see Fig. 21). When we want to close the window we only have to click on it with any of the mouse's button.

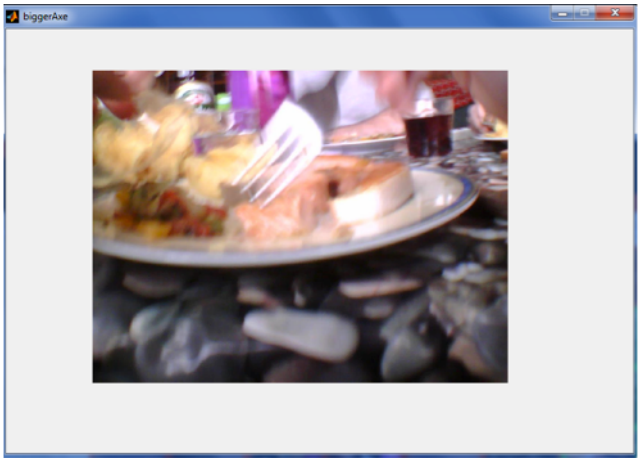

Fig. 21. Bigger Axe Window.

Frames Interaction: In order to build a useful affordance to the Labels Box (see section 4.1 Main Window), we tried to make it look like an old movie film. Ordered from left to right and top to bottom, we can hover over each of the little rectangles (that represent each of the frames of our present set). When doing so, we will get a feedback making appear a little image of its corresponding picture near the top right corner. In addition, during the labelling process, the classes chosen for each of the images will be painted on each of these frames.

Class indicator (or Classes List): Whether we are hovering over the Labels, Queries or Cluster boxes, we will be able to know the class corresponding to its colour on the middle top box. This is a very useful feature if we have a lot of different labels and it gets hard to remember the colour of each of them.

Help: Any time we need help with some of the application features we can go to the Help menu and review some of them.

4.4 Pre-processing Module

Besides the main application we will also have to use the pre-processing module in order to have the folders on the right format for them to be accepted by the Active Learning application. And the only script that we will have to run for this procedure is the mainPreprocessing.m file, whose modifiable variables are described in the Table 2.

The overview of its operation can be reviewed in section 3.3 Sliding Window and Pre-processing. But the most important thing for the user to know, is that once we give it the source folder (with all our images) and we tune its parameters, at the end we will get 2 sets of folders with their names* formatted like:

Original resolution sets: [original-folder-name]_[init-image]to[fin-image]_slides-part[X]

Low resolution sets: [original-folder-name]_[init-image]to[fin-image]_slides-part[X]_[resolution]

The original resolution sets will be used for the bigger axe window, and the low resolution sets are the ones that we will have to chose in the labelling application, and none of them should be removed. These resulting folders are referenced as “Sets X” by Fig. 12 scheme in section 3.1 Application General Flowchart.

Name	Usage	Allowed Values	Default
main_path	Absolute path to the folder where we find the sets to label (and to pre-process).	Any valid path.	""
main_folder	Name of the folder that we want to pre-process.	Any valid folder name with images in main_path.	""
format	Extension of the images.	Any image format recognized by Matlab.	".jpg"
initPer	Initial percentatge rate of the slidding window that will crop de photos.	Any natural value between finPer and 100.	75
finPer	Final percentatge rate of the slidding window that will crop de photos.	Any natural value between 1 and initPer.	25
jumpPer	Difference between the proportions of a group of cropped images and the next one.	Any natural value smaller or equal to initPer - finPer.	10
diffSlide	Percentual difference (100 - overlap) of one cropped image and the following one.	Any natural value bigger than 1.	10
ini	Position of the first photo that we will pre-process in the main_folder.	Any natural value smaller or equal to fin.	1
fin	Position of the last photo that we will pre-process in the main_folder.	Any natural value bigger or equal to ini.	len(images)
maxSize	Maximum number of cropped images in the resulting folders.	Any natural value.	4000
proportions	Final proportions (resolution in px) of the resulting images.	Any array of two natural values.	[89 120]
haveDate	Indicates if we include the date the photo was taken (stored in the file) in the features.	Boolean	true
hsize	Gaussian's filter side length (px) applied before calculating HOG.	Any natural value bigger than 1.	6
sigma	Gaussian's filter sigma applied before calculating HOG.	Any value bigger than 0.	3

Table 2. Information about all the modifiable variables in the preprocessing module (mainPreprocessing.m).

*These names are very important for the labeller functioning, so they should not be modified.

5. Results

In this section we will talk deeply about all the tests done and all the results that we obtained during the attainment of the data.

First of all we will present how we obtained the data and the sets we used for the following tests, and after that we will start with a test involving the HOG features.

Next, we will expose some tests related to the classifiers (KNN and AdaBoost), how they work individually and for getting the best parameters for their optimal performance: sensitivity and specificity; precision and likelihood and their performance during their usage in the forest tests (see section 5.9 Forest Labelling Simulation).

Later we will apply some tests on the active learners and dimensionality reduction algorithms (LDA, LMNN) and on the clustering K-Means algorithm that we use after them. Checking their performance on synthetic and on our real data.

And last, but not least, we will perform the forest labelling simulation tests on which will be useful to compare all the algorithms and techniques used (separately and combined with others).

5.1 Data Sets

During the compilation of images, the author was the subject who kept the record of his everyday life tasks with the wearable camera, which we could summarize with the following points:

- Images from about 15 different days, including business days and holidays.
- Each day has from about 300 to 4.500 pictures.
- We tried to focus at least on the mealtimes of those days.
- Summing up about 43.750 images.
- 2.900 images per day on average.
- From about 1 to 14 hours per day.

With these images and the ones downloaded from ImageNet (see section 3.2 Dishes Images Compilation), we gathered all the sets used for these tests.

In the following table, we can see a review of all the sets labelled and used for testing all the application's algorithms. In total we labelled 100 images of dishes from ImageNet and 408 images taken with SenseCam, which after the pre-processing and before the labelling became 89.709.

LABELED SETS							
ID	Source	Source Folder	First Image	Last Image	# Images	# Slides	Results Folder
00000	ImageNet	Dishes images...	000001.jpg	000025.jpg	25	4.013	Dishes images..._1to100_slides-part1-part1_76x102
00001	ImageNet	Dishes images...	000026.jpg	000050.jpg	25	4.012	Dishes images..._1to100_slides-part1-part2_76x102
00002	ImageNet	Dishes images...	000051.jpg	000075.jpg	25	4.013	Dishes images..._1to100_slides-part2-part1_76x102
00003	ImageNet	Dishes images...	000076.jpg	000100.jpg	25	4.011	Dishes images..._1to100_slides-part2-part2_76x102
00004	SenseCam	EE6B5EA5...	000149.jpg	000168.jpg	20	4.000	EE6B5EA5..._150to350_slides-part1_76x102
00005	SenseCam	EE6B5EA5...	000169.jpg	000188.jpg	20	4.000	EE6B5EA5..._150to350_slides-part2_76x102
00006	SenseCam	EE6B5EA5...	000189.jpg	000208.jpg	20	4.000	EE6B5EA5..._150to350_slides-part3_76x102
00007	SenseCam	EE6B5EA5...	000209.jpg	000228.jpg	20	4.000	EE6B5EA5..._150to350_slides-part4_76x102
00008	SenseCam	650EA48B...	006851.jpg	006870.jpg	20	4.000	650EA48B..._11to91_slides-part1_76x102
00009	SenseCam	650EA48B...	006871.jpg	006890.jpg	20	4.000	650EA48B..._11to91_slides-part2_89x120
00010	SenseCam	650EA48B...	006891.jpg	006910.jpg	20	4.000	650EA48B..._11to91_slides-part3_89x120
00011	SenseCam	650EA48B...	006911.jpg	006930.jpg	20	4.000	650EA48B..._11to91_slides-part4_89x120
00012	SenseCam	650EA48B...	006931.jpg	006931.jpg	1	200	650EA48B..._11to91_slides-part5_89x120
00013	SenseCam	D165DFFA...	016464.jpg	016501.jpg	36,5	4.000	D165DFFA..._3783to3868_slides-part1_89x120
00014	SenseCam	D165DFFA...	016501.jpg	016538.jpg	37	4.000	D165DFFA..._3783to3868_slides-part2_89x120
00015	SenseCam	D165DFFA...	016538.jpg	016551.jpg	13,5	1.460	D165DFFA..._3783to3868_slides-part3_89x120
00016	SenseCam	FD214597...	005975.jpg	005994.jpg	20	4.000	FD214597..._3844to3903_slides-part1_89x120
00017	SenseCam	FD214597...	005995.jpg	006014.jpg	20	4.000	FD214597..._3844to3903_slides-part2_89x120
00018	SenseCam	FD214597...	006015.jpg	006034.jpg	20	4.000	FD214597..._3844to3903_slides-part3_89x120
00019	SenseCam	0BC25B01...	000002.jpg	000021.jpg	20	4.000	0BC25B01..._3to62_slides-part1_89x120
00020	SenseCam	0BC25B01...	000022.jpg	000041.jpg	20	4.000	0BC25B01..._3to62_slides-part2_89x120
00021	SenseCam	0BC25B01...	000042.jpg	000061.jpg	20	4.000	0BC25B01..._3to62_slides-part3_89x120
00022	SenseCam	6FD1B048...	000080.jpg	000099.jpg	20	4.000	6FD1B048..._81to120_slides-part1_89x120
00023	SenseCam	6FD1B048...	000100.jpg	000119.jpg	20	4.000	6FD1B048..._81to120_slides-part2_89x120
				TOTAL:	508	89.709	

Table 3. Sets of images used for the realization of these tests.

5.2 Gaussian and HOG Test

In this first test, our intention was to polish the result of the HOG gradients as much as we could, in order to get the features that could describe our samples the best way possible. As a result all the machine learning algorithms that we applied afterwards would benefit from their performance.

After doing also some checks of the best possible image resolutions (seeing that the difference was not very relevant but that 89x120 was slightly better than the rest), we applied a Gaussian filter to the images right before calculating its HOG (Fig. 22).

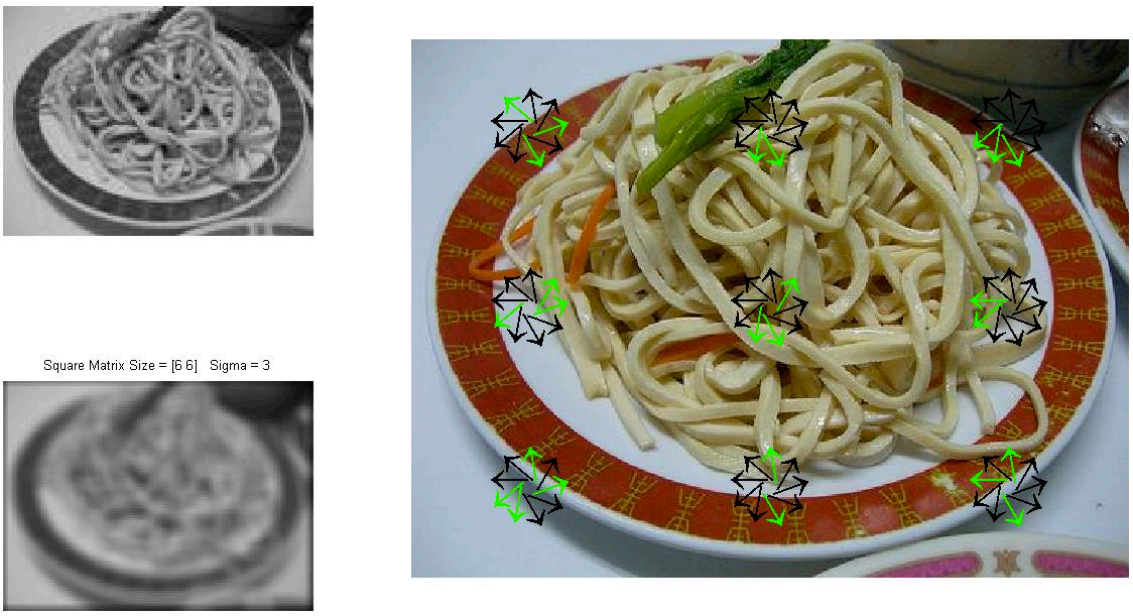

Figure 22. Most relevant (green) HOG gradients on a dish image (right) when applying a Gaussian filter (bottom-left).

First of all, we must know that each of the arrows drawn over the bigger images correspond to each of the HOG features (see section 3.4 Sample Features and Classes Selection), which represent the intensity of the gradients on that direction. In spite of their continuous values, in this test we focused only on showing the 25 gradients with higher values (coloured in green) and left the rest of them uncoloured.

Taking this into account, we can see that 22 out of the 25 bigger greater gradients give us valuable information for distinguishing a dish in the image. Only the 3 green gradients in the centre of the dish will act as noise. For obtaining this image, we used a Gaussian filter with matrix size [6 6] and sigma 3, which were obtained after comparing the output between different parameters.

5.3 K-Nearest Neighbours Examples

In this test, which was simply done to check how did the KNN algorithm work, we can see the classification of two different test samples using k = 15. One of them obtains a high likelihood value (Fig. 23) by the KNN classification, and the other one obtains a low likelihood value (Fig. 24). And right under them we can see the 15 nearest neighbours that the KNN algorithm choses to decide the label of each of these samples.

Fig. 23. Example of a high likelihood image given by KNN with its 15 nearest neighbours (all semidish).

Fig. 24. Example of low likelihood image given by KNN with its 15 nearest neighbours (top images: semidish, first 3 center: semidish, last 2 center: no dish, bottom no dish).

5.4 Sensitivity and Specificity Online Learners

The sensitivity and specificity of a classifier is an essential performance and optimality measure. Given that:

$$Sensitivity = \frac{TruePositives}{TruePositives + FalseNegatives}$$

$$Specificity = \frac{TrueNegatives}{TrueNegatives + FalsePositives}$$

we can describe the sensitivity as the probability of a positive result given that it should be positive, and the specificity as the probability of a negative result given that it should be negative.

To maximize these parameters, we performed a 10-fold test simulating a Single Classifier (see section 3.6 Online Learner), taking a 10% of the total samples as validation and the 90% left for training the corresponding classifier. We repeated the test for our two classifiers (AdaBoost and KNN) and with different parameters. Saving for each calculation their TN, TP, FN and FP values and with them the sensitivity (Fig. 25) and specificity (Fig. 26).

Note: for all the following graphics, NR = number or AdaBoost rounds, NT = number of tests performed to calculate the average, NN = k = number of nearest neighbours considered.

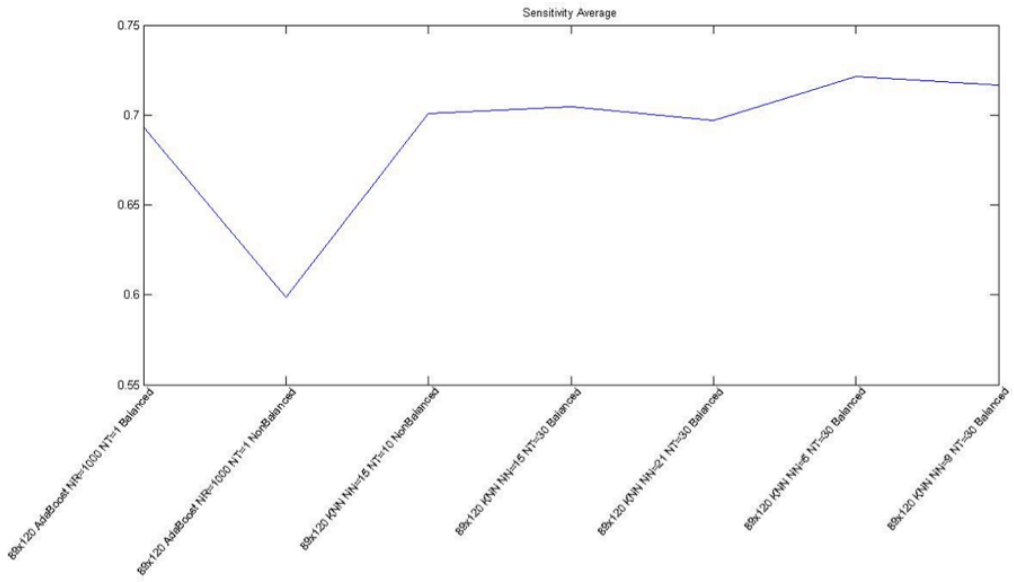

Fig. 25. Average sensitivity for the classifier NPvsPvsSP.

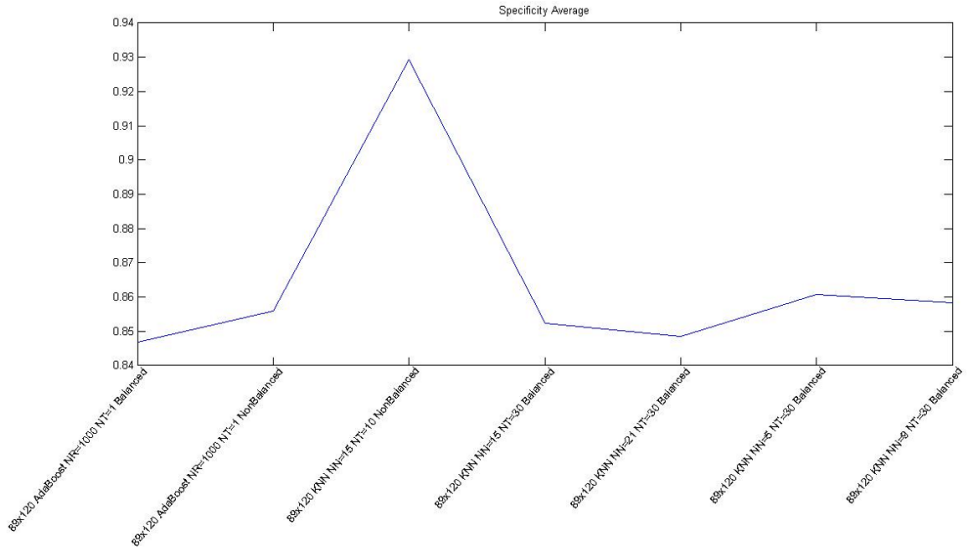

Fig. 26. Average specificity for the classifier NPvsPvsSP.

First of all, we were able to detect that the differences between AdaBoost and KNN were not very remarkable. Then, about the balancing of the different classes, we knew that the nature of our samples and of the objects that we are trying to detect (dishes), always causes that we have an unbalanced distribution of our data. This means that we always inevitably tend to have NP >>> SP >>> P. Due to this fact and the differences and similarities between the features of our classes, which we could summarize saying that the images with label P are more similar to the label SP than NP, we thought that creating a Combined Classifier, similar to an ECOC (Error-Correcting Output Code) classifier [24], could be more beneficial to us.

So as to try to improve the results, we created a cascade-like double classifier, first differentiating NP vs (P + SP) and P vs SP after that, getting the specificity and sensitivity results from Fig. 27 and Fig. 28 respectively.

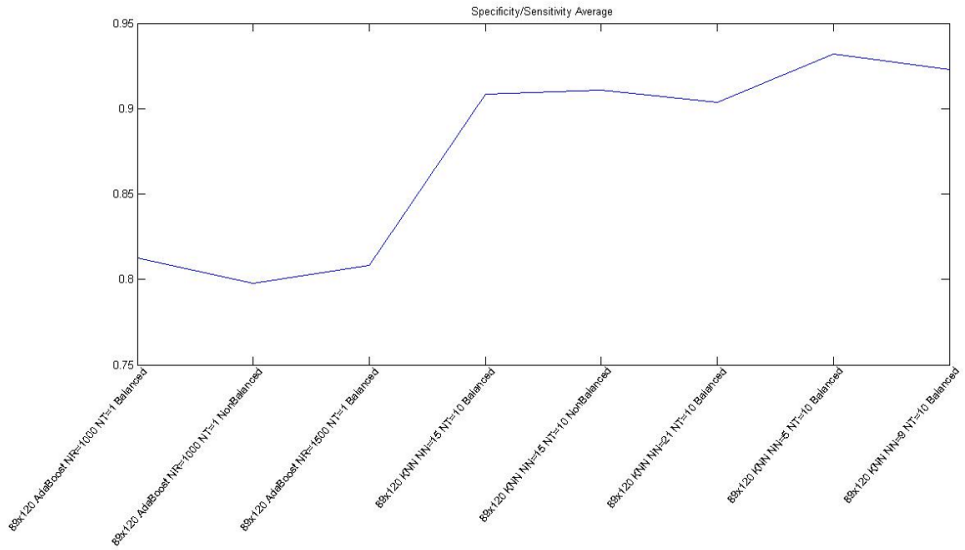

Fig. 27. Average specificity and sensitivity for the classifier NP vs (P + SP).

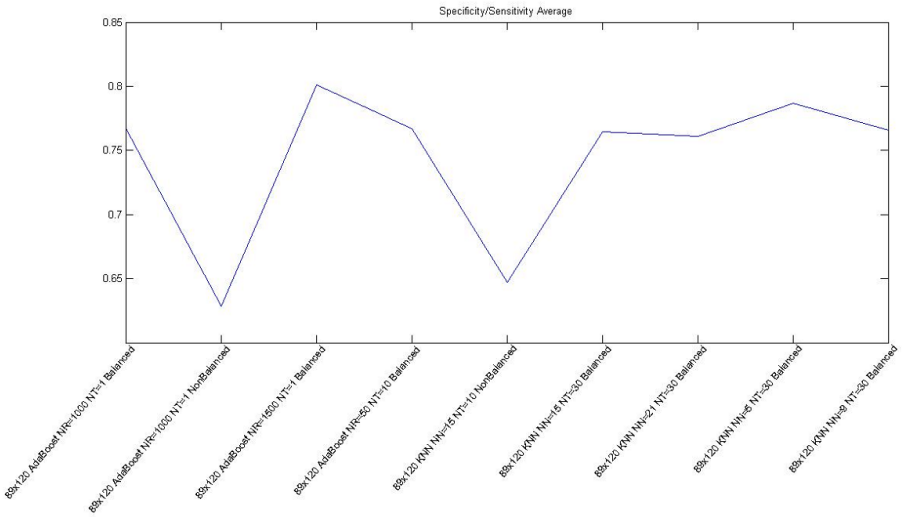

Fig. 28. Average specificity and sensitivity for the classifier P vs SP.

At last, we could see that classifying the different samples in the cascade would give us a better performance, and that in this case KNN seemed to be better than AdaBoost. But even though k=5 seemed to get better results, we decided taking k=15 due to its better generalization.

5.5 Precision vs Likelihood Online Learners

After the sensitivity and specificity test we proceeded with the calculation of the precision and the likelihood (probability of a given sample of belonging to a specific class) for the chosen methods (see Fig. 29, for AdaBoost results and Fig. 30, for KNN results). These measures are very important in order to be as sure as possible that we only choose as predicted labels the ones with a high enough precision.

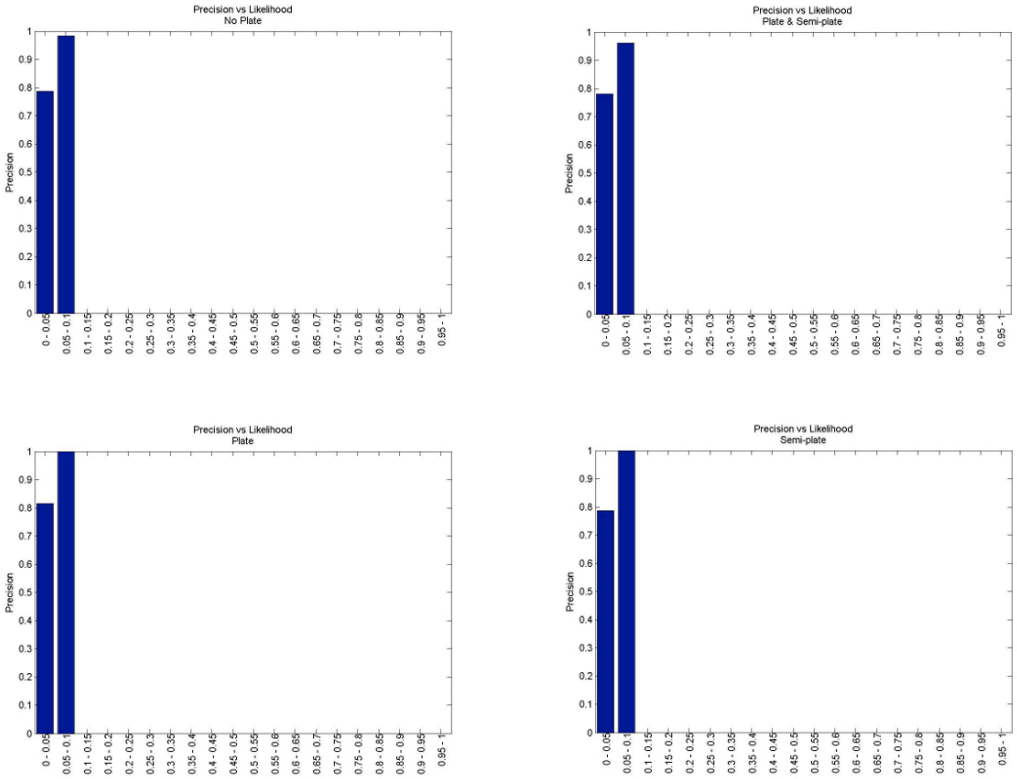

Fig. 29. Precision vs Likelihood graphics from AdaBoost classifier.

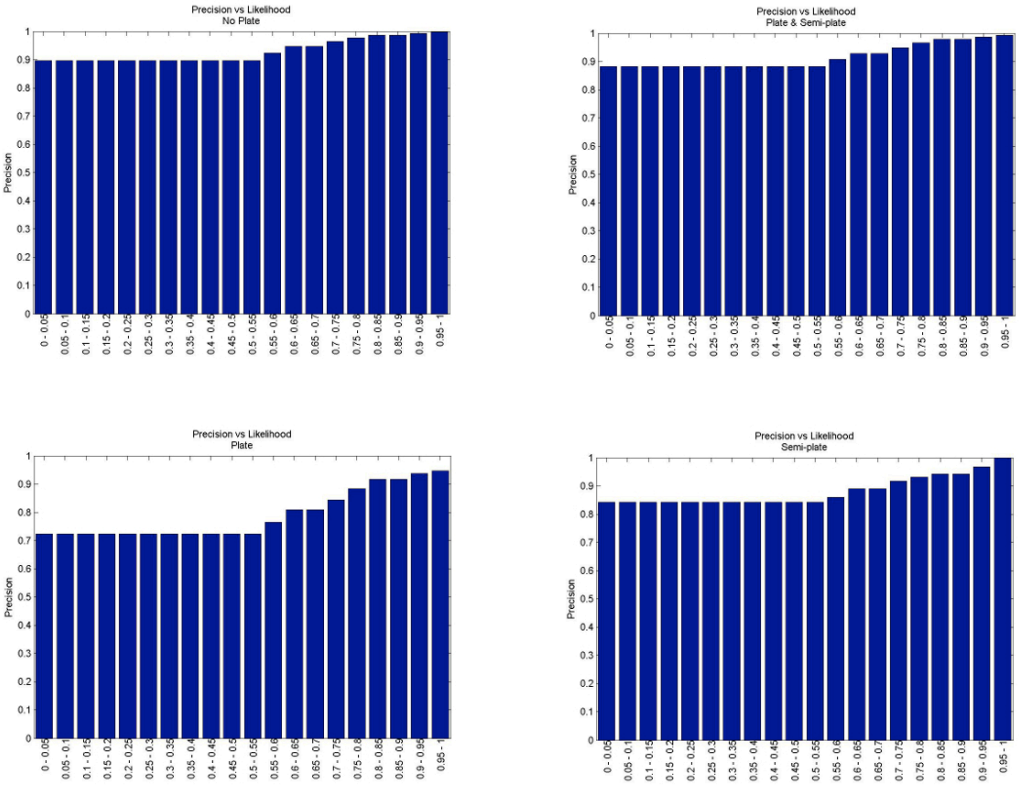

Fig. 30. Precision vs Likelihood graphics from KNN classifier.

Considering that a 95-97% of minimum precision would be enough for the predicted labels, we took their corresponding likelihoods (the described as follows) as measures to use in the Online Learner:

KNN, k = 15:	No Dish: 0.7	Dish: 0.9
	Dish & SemiDish: 0.7	SemiDish: 0.8
AdaBoost, 1500 rounds:	No Dish: 0.075	Dish: 0.075
	Dish & SemiDish: 0.075	SemiDish: 0.075

Whenever a new sample, during the initial screening, has a lower likelihood than its corresponding value previously described, it will not be assigned as a sample with a predicted label. On the other side, if its likelihood is higher, it will be one of the starting samples with predicted labels.

5.6 Classification Time and Results Online Learners

Using the tests performed in section 5.9 Forest Labelling Simulation, we calculated the mean time needed by our classifiers (KNN and AdaBoost) to classify a set of images and the mean number of elements of each label that they managed to predict for each set iteration.

Looking at the first two images, we can see the mean time needed to classify the images in each new set (Fig. 31, left), and the mean number of labels predicted for each of the classes (Fig. 31, right).

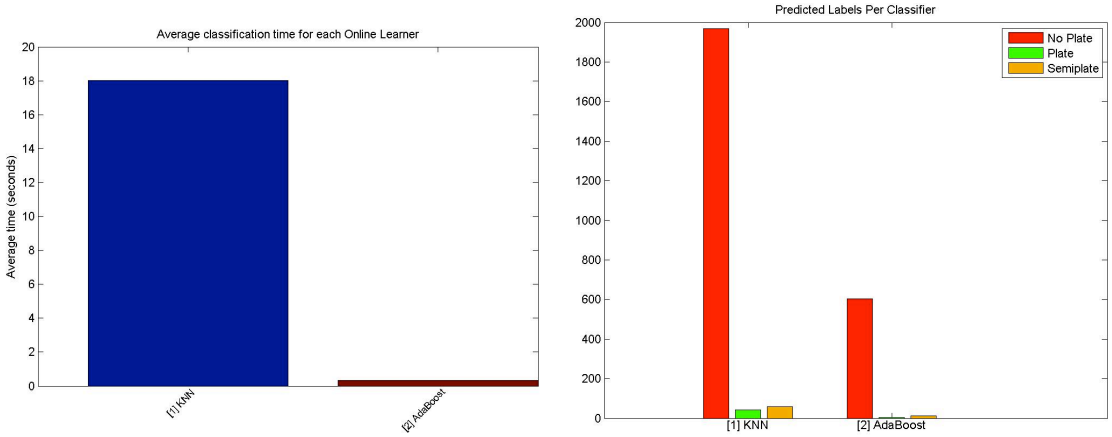

Fig. 31. Left: mean time needed for the classification of a single set. Right: mean number of predicted labels in each set and each class.

Although the time needed for the KNN classification is much longer than the one needed for the AdaBoost, it is nothing compared with the average classification time needed for a manual set labelling, which is about 40 minutes. And checking the predicted results, we can see that the time is worth. The number of KNN's predicted labels is about 3 times the ones predicted by AdaBoost.

Checking the evolution of those predicted labels (Fig. 32) we can see a good improvement at the first sets and a maximum performance around the 11th and 12th set, but due to the differences between the images taken in different situations and days, the results of the Online Learners seem to oscillate.

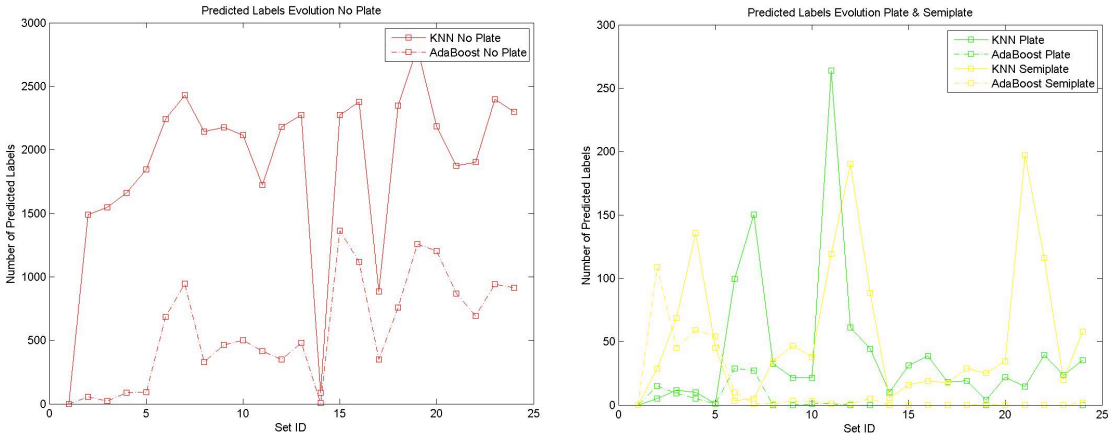

Fig. 32. Left: evolution in the number of the No Dish labels predicted for each of the classifiers. Right: evolution in the number of Dish and Semidish labels predicted for each of the classifiers.

5.7 Synthetic Data Tests (LDA, LMNN and K-Means)

Before trying how our algorithms for dividing our data in classes worked with the real data, we created a synthetic generator of samples. In order to simulate the distribution of our real data in the best way possible, we created 10 random points of 94 dimensions and after that, generated groups of 400 more samples normally distributed around each of them (around 4000 samples like in our real sets). Then, modifying its sigma parameter we were able to extend or shorten their distribution and overlap or separate them. Later, the only thing we had to do was randomly assigning to each of the 10 groups one of our 3 labels. And as we can see in Fig. 33, we are able to get a reasonable simulation with data easier or harder to separate.

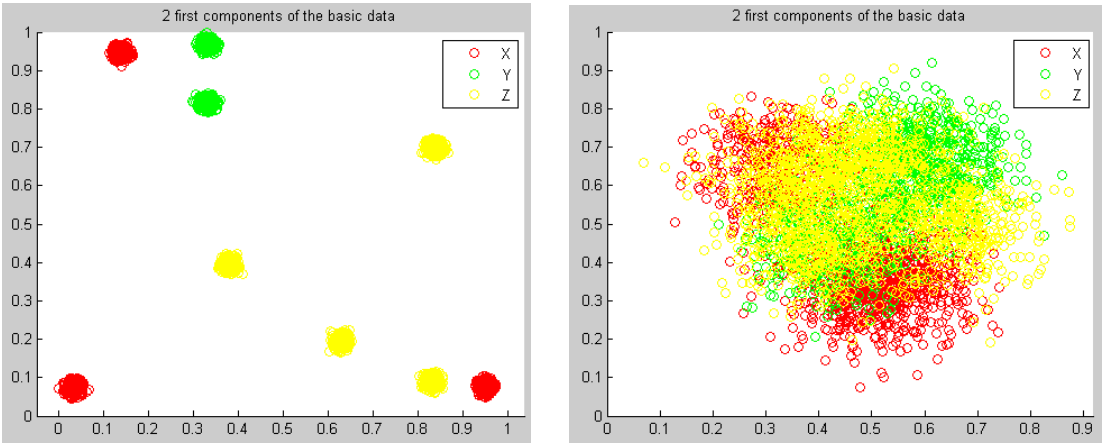

Fig. 33. First 2 components of synthetic samples. Left: sigma = 0.01. Right: sigma = 0.2.

After generating the data, and taking for example a value of sigma = 0.2, we can apply the PCA + LDA algorithms on one side and PCA + LMNN on the other, and compare their performance (Fig. 34). The first thing we can see is that as we knew, the LDA tries to linearly separate the different labels as best as possible, and LMNN tries to locally shorten the distances between its nearest neighbours with the same labels and further the distances to the rest of the labels. Both of them separate the data quite well.

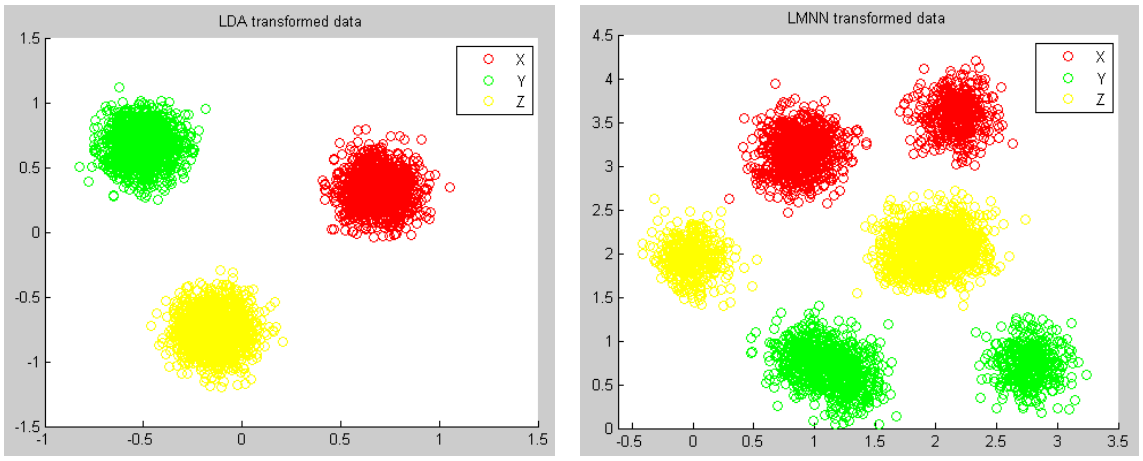

Fig. 34. Left: PCA+LDA application to samples with sigma = 0.2. Right: PCA+LMNN (k=7) application to samples with sigma = 0.2.

And finally, as the last step, we can use K-Means (K=2) to separate the samples in two clusters like we do in our active learner with the real data (Fig. 35). Seeing that even though both of them work fairly well, it seems that LDA will need less iterations than LMNN to separate the samples.

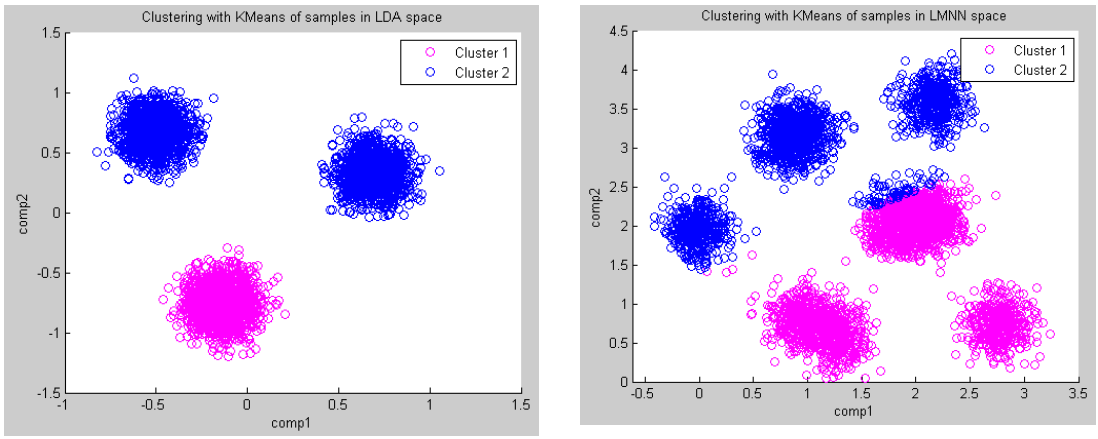

Fig. 35. K-Means separation (k=2) of samples in LDA space (left) and LMNN space (right).

Although, there are two important points to consider that we are not able to compare with these tests:

i) The real samples might not be as normally distributed as our synthetic ones, and in that case it would not benefit LDA's performance as much as the Gaussian distributed do, and the LMNN's local separation could be better.

ii) We are executing the LMNN test with an output dimensionality of 2 (being able to see the difference with LDA), but in the real case we can set a higher dimensionality and get better results.

5.8 Real Data Tests (LDA, LMNN and K-Means)

Now, following the same procedure but with the real data (data set with ID = 00011, see section 5.1 Data Sets), we can visualize the data distribution right before applying the division in the first cluster among the different classes (Fig. 36). This fact indicates us that at a first sight there is little distinctive features values between the classes.

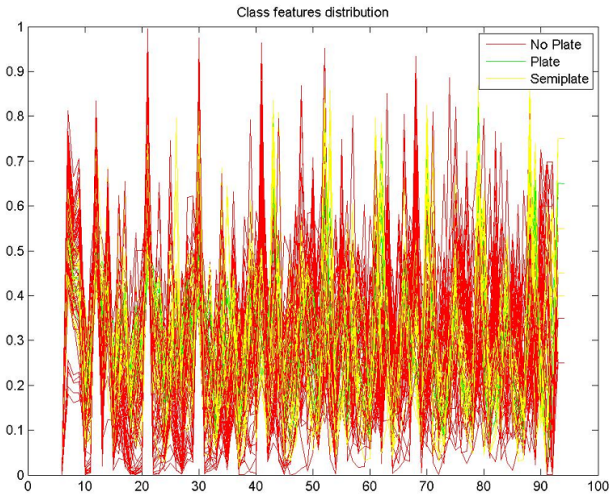

Fig. 36. Data distribution of samples in set 00011. Each line represents the value for each of its features of a sample.

Now, as we did on the previous test, we can visualize the division of the data done by LDA and by LMNN (k=7) (this time we use output dimensions = 6. This means that we will not be able to visualize all of them) (Fig. 37). And later we applied the K-Means (k=2) (Fig. 38).

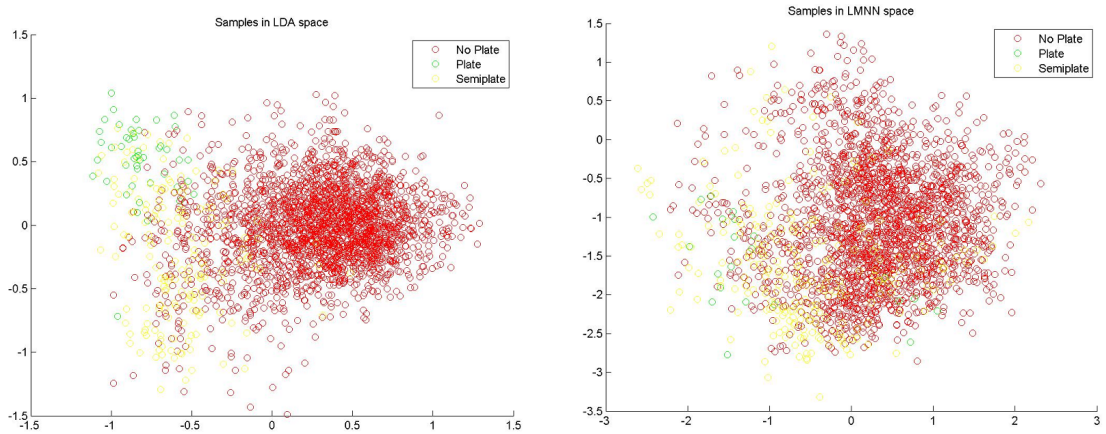

Fig. 37. Set 00011 samples in LDA (left) and LMNN (right) (2 of 6 dimensions) spaces.

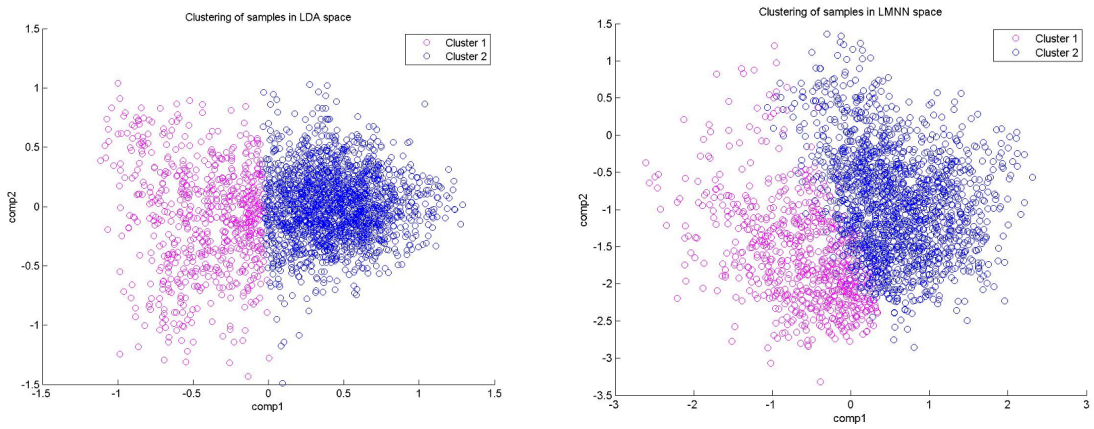

Fig. 38. Set 00011 samples in LDA (left) and LMNN (right) (2 of 6 dimensions) spaces. K-Means (k=2) clusterization.

If we compare these results with the ones obtained on the previous test (5.7 Synthetic Data Tests (LDA, LMNN and K-Means)), we see that now there does not seem to be any handicap for LMNN against LDA, they seem to obtain similar results. Both of them put P and SP in a cluster (which is logic due to their closest similarity) and the NP samples into another. Only some NP samples were left in the first cluster.

But the aspect that might make the difference but we can not appreciate with this test, is that the combination of the higher dimensionality of the LMNN space may get a better division after using K-Means than LDA does only with its 2D space.

5.9 Forest Labelling Simulation

With the following tests we intended to simulate the labelling of a huge amount of images, divided in sets of different sizes and with all the different methods and algorithms that we implemented during the project.

First of all we present each of the tests carried out. Each of the names is formatted in the following way:

[id] Online Learner Type + Active Learner Type - Division of Sets - Prototypes Used

And the values can be:

i) **Online Learner Type**: KNN, AdaBoost or No Classifier.
ii) **Active Learner Type**: LDA, LMNN or Single Linkage.
iii) **Division of Sets**: Single Sets (24 sets exposed in section 5.1 Data Sets), Join Days Sets (7 sets divided by the common Source Folder in section 5.1 Data Sets), or Join All Sets.
iv) **Prototypes Used**: OnePerClass, OnePerLeaft or No Prototypes.

Test Names

[1] KNN + LDA - Single Sets - No Prototypes

[2] KNN + LDA - Single Sets - OnePerClass

[3] KNN + LDA - Single Sets - OnePerLeaf

[4] AdaBoost + LDA - Single Sets - OnePerClass

[5] KNN + LMNN (k=3) - Single Sets - OnePerClass

[6] AdaBoost + LMNN (k=3) - Single Sets - OnePerClass

[7] KNN + LDA - Join Days Sets - OnePerClass

[8] No Classifier + LDA - Join All Sets - No Prototypes

[9] KNN + Single Linkage - Single Sets - OnePerClass

[10] No Classifier + LDA - Single Sets - OnePerClass

[11] KNN + LMNN (k=7) - Single Sets - OnePerClass

[12] AdaBoost + LMNN (k=7) - Single Sets - OnePerClass

[13] (Dasgupta) No Classifier + Single Linkage - Single Sets - No Prototypes

In the following tables (Table 4), we can see the detailed numerical information about means and standard deviations of each of the tests, even though, in the following pages we will detail and analyse the differences between the CLICKS and TIME with more visual graphics.

CLICKS

Test Name	Mean	Std Deviation	Num. Tests
[1]	18744.6667	283.9742	3
[2]	18999.5	130.921	6
[3]	19709.3333	229.2386	3
[4]	19647	175.8958	4
[5]	18161.5	105.9512	4
[6]	16978.5	14.8492	2
[7]	20372.6	111.6392	5
[8]	24515.6667	77.938	3
[9]	15366.3333	105.6709	3
[10]	20445.3333	283.179	3
[11]	18176.6667	27.7369	3
[12]	17060	62.2254	2
[13]	11832	176.0767	3

TIME

Test Name	Mean	Std Deviation	Num. Tests
[1]	2766.498	220.6277	3
[2]	2970.1137	404.0357	6
[3]	2664.1583	61.3742	3
[4]	4734.5765	136.7274	4
[5]	5969.2745	352.217	4
[6]	5578.259	611.6049	2
[7]	4327.1736	244.7131	5
[8]	31480.493	1270.9214	3
[9]	3873.065	138.4481	3
[10]	4522.463	392.6235	3
[11]	8142.2253	414.5577	3
[12]	6709.4295	53.0974	2
[13]	7176.3753	99.0568	3

NORMAL CLUSTERS

Test Name	Mean	Std Deviation	Num. Tests
[1]	5445.3333	501.5519	3
[2]	5749.6667	224.1015	6
[3]	5746.6667	271.9491	3
[4]	9649.75	264.4181	4
[5]	9434.5	441.2788	4
[6]	13060	67.8823	2
[7]	7731	218.9874	5
[8]	18380.3333	655.6229	3
[9]	9269	209.0072	3
[10]	9730.6667	327.813	3
[11]	9574.3333	227.7199	3
[12]	13049.5	603.1621	2
[13]	25800.3333	147.8862	3

PURE CLUSTERS

Test Name	Mean	Std Deviation	Num.Tests
[1]	8384.6667	614.8376	3
[2]	8729.6667	257.3563	6
[3]	9172.3333	328.7801	3
[4]	13992.75	295.774	4
[5]	12911	416.3436	4
[6]	16106.5	20.5061	2
[7]	11468.6	271.4633	5
[8]	24864.6667	600.5017	3
[9]	12645.3333	189.3207	3
[10]	14417.3333	436.6238	3
[11]	12943.3333	261.3988	3
[12]	16071.5	577.7062	2
[13]	29600.3333	186.4037	3

Table 4. Mean and standard deviation of each measure for all the tests. Top left: clicks needed to label all the images. Top right: time needed to label all the images. Bottom left: sum of clusters obtained at the leaves position for all the trees used. Bottom right: sum of "purified" (separating the ending clusters by their different labels) leaves for all the trees.

Note: we must take into account that in the previous tables and in the following comparisons, the measure TIME represents the time needed during the labelling process. This means that it does not consider neither the online learner training time nor the set loading and initial screening time.

In order to analyse more easily all the tests, we divided them in 5 different groups in order to compare: Prototypes, Sets division, Active Learners, Online Learners and the combinations of Active Learner + Online Learner (including the most similar possible Dasgupta's approach, since we should have used Join All Sets, but Matlab can not generate a bigger enough Single Linkage hierarchical tree without running out of memory).

1- Prototypes

Comparison Subjects:

No Prototypes: [1]

OnePerClass Prototypes: [2]

OnePerLeaf Prototypes: [3]

Clicks:

There is not any relevant difference between the tests. Maybe this definition of the prototypes (median of all the samples in the corresponding group) does not represent the previously labelled examples as well as we thought.

Time:

There is no remarkable difference in this aspect either.

Overall:

None of them seem to be better than the rest. Further tests should be done in the future to improve and take advantage of these strategies.

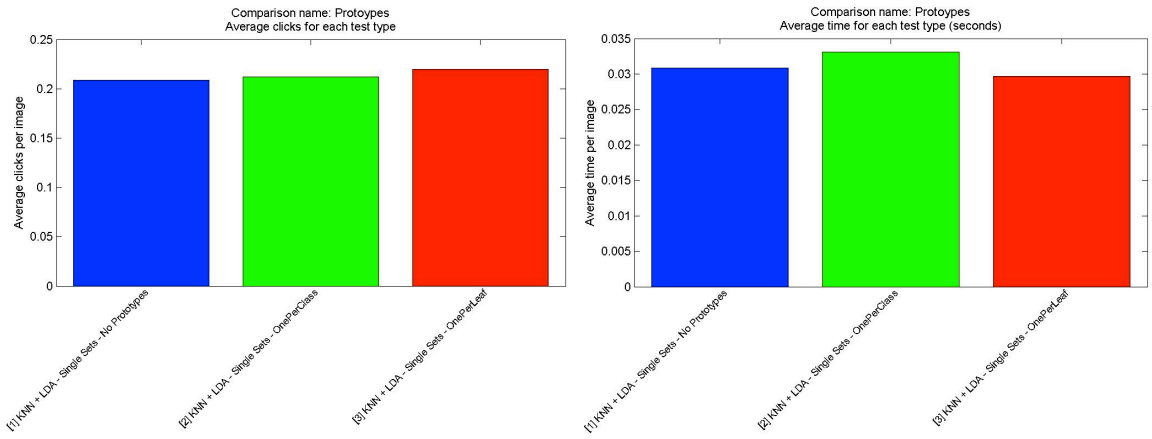

Fig. 39. Average clicks (left) and average time (right) per image for each of the tests. No prototypes [1] blue, OnePerClass [2] green and OnePerLeaf [3] red.

2- Sets division

Comparison Subjects:

Single Sets [2]

Join Days Sets [7]

Join All Sets [8]

Observation: the "join all sets" test can not use any online classifier because we label all the images on a single tree. This means that it can influence the results of this comparison (analysed in test 4- Online Learners).

Clicks:

These tests prove that clearly the smaller the sets are, the less clicks the user has to do.

Time:

The leap between "Join Days" and "Join All" is impressive and relevant, even though, this difference on the performance is probably leaded by the lack of an online learner.

Overall:

Single Sets is clearly better than the other methods.

It can seem intuitively probable that reducing even more the number of samples per set can improve the performance, until a critic line on which the results may become worse. In more exhaustive tests in some future work it would be interesting to check this limit to optimize the algorithm performance.

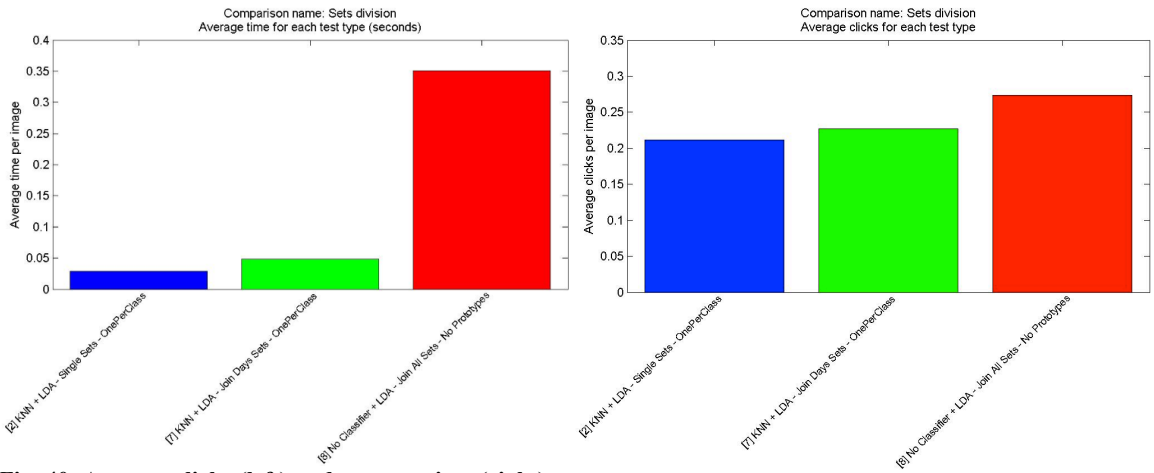

Fig. 40. Average clicks (left) and average time (right) per image for each of the tests. Single Sets [2] blue, Join Day Sets [7] green and Join All Sets [8] red.

3- Active Learners

Comparison Subjects:

LDA [2]

LMNN (k=3) [5]

LMNN (k=7) [11]

Single Linkage Hierarchical Cluster Tree [9]

Clicks:

On this feature the Single Linkage seems slightly better than the rest. And increasing the value of K of LMNN does not improve its performance, so in this aspect the algorithm seems quite robust with respect to different values of K.

Time:

LDA is fairly quicker than Single Linkage and at the same time the last one is quicker than LMNN. Considering increasing the value of K of the method LMNN, we can see that the difference in time is even more remarkable due to the more time needed for the LMNN training.

Overall:

Single Linkage still seems to do better in terms of clicks, but LDA seems to outperform it talking about the time.

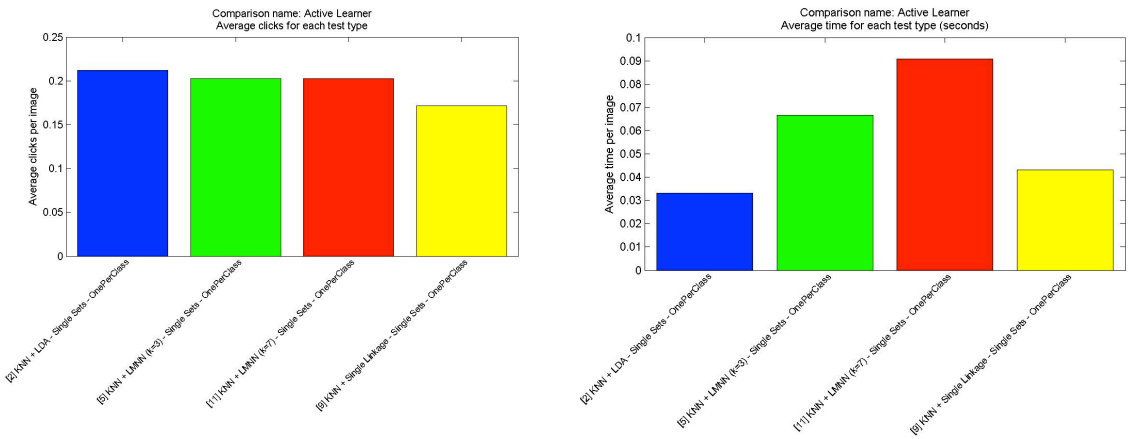

Fig. 41. Average clicks (left) and average time (right) per image for each of the tests. LDA [2] blue, LMNN (k=3) [5] green, LMNN (k=7) [11] red, Single Linkage [9] yellow.

4- Online Learners

Comparison Subjects:

KNN [2]

AdaBoost [4]

No Classifier [10]

Clicks:

The differences between the different approaches are not considerably remarkable, but it is true that we can intuit that using some kind of Online Classifier can decrease a little bit the number of clicks.

Time:

At this aspect KNN outperforms by far the rest of the methods. It reduces the labelling time span considerably compared with the others.

Overall:

For our type of data, KNN takes a greater advantage on both features, clicks and time. Considering the time needed for the classifier training (which is not considered on this comparison), it is overwhelmingly better than AdaBoost, because KNN needs about 30 seconds with about 85.000 photos and AdaBoost needs more than 20 minutes.

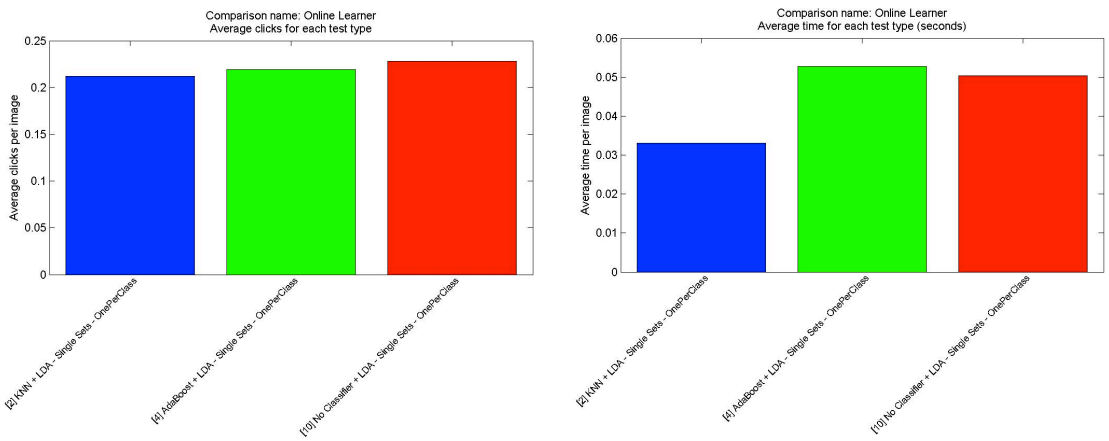

Fig. 42. Average clicks (left) and average time (right) per image for each of the tests. LDA [2] blue, AdaBoost [4] green, No Classifier [10] red.

5- Online & Active Learners

Comparison Subjects:

KNN + LDA [2]

AdaBoost + LDA [4]

KNN + LMNN (k=3) [5]

AdaBoost + LMNN (k=3) [6]

KNN + LMNN (k=7) [11]

AdaBoost + LMNN (k=7) [12]

(Dasgupta) No Classifier + Single Linkage [13]

Clicks:

On one side, on this point, and looking at the Active Learners performance, we can still see a better behaviour of the subjects using LMNN (3rd to 6th), even though, using a higher value of K does not seem to improve these results.

On the other side, comparing the Online Learners, contrary to what one might think, seems that the combination of AdaBoost + LMNN needs less clicks than KNN + LMNN, but when combining them with LDA the results are the other way round. This seems to indicate that these methods combined can obtain better results.

Comparing the Dasgupta's method, we see that it still needs less clicks that any other kind of algorithms.

Time:

As we saw on the 3- Active Learners comparison, LDA is far quicker than LMNN. And KNN also is quicker than AdaBoost, but if we look at the combination of AdaBoost + LMNN, as we saw on the clicks, surprisingly it does not exactly follow the same pattern, and seems to do better than KNN + LMNN.

Analysing the results of Dasgupta's method seems to indicate that on this feature KNN + LDA outperforms it, and that using an Online Learner increases the performance considerably.

Overall:

KNN + LDA seem to decrease the time needed for the labelling, but Dasgupta's method is still the best on the number of clicks.

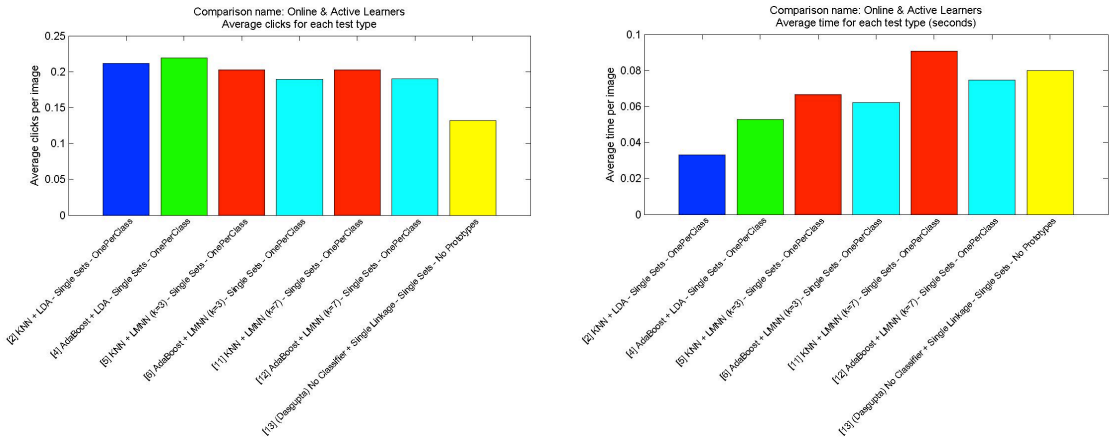

Fig. 43. Average clicks (left) and average time (right) per image for each of the tests. KNN + LDA [2] blue, AdaBoost + LDA [4] green, KNN + LMNN (k=3 and k=7) [5 and 11] red, AdaBoost + LMNN (k=3 and k=7) [6 and 12] light blue and (Dasgupta) No Classifier + Single Linkage [13] yellow.

5.10 Clicks Vs Time Simulation Results

In this final test, we can see a review of all the methods used in the previous section 5.9 Forest Labelling Simulation. Using a scatter plot along a time axis and a clicks axis and representing each of the tests with a dot, it is easy to check which of them shows some of the best results.

First of all, knowing that the closer to both clicks and time 0 values the dot is, the best click/time rate will have, we can see (Fig. 44) that the test [8] is the worst with difference. Which taking into account the previous tests, clearly seems to be leaded by the Join All Sets procedure.

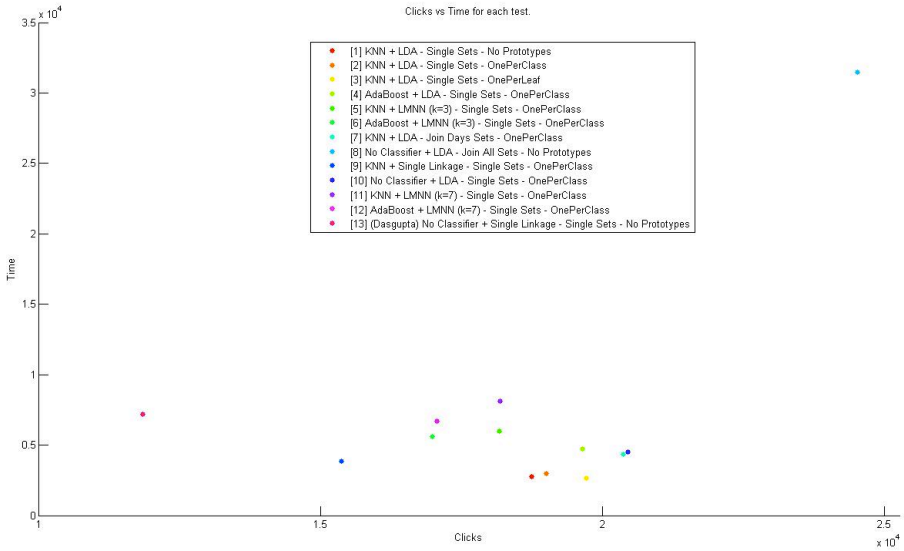

Fig. 44. Scatter plot with each of the tests represented by a coloured dot. The closer to the 0 time and clicks values a test is, the better the time/clicks rate will be.

Now leaving the [8] aside (Fig. 45), taking a closer look to the rest of the tests, and considering, too, that the scales in the axis of these figures are smaller for the clicks and higher for the time, we can see better the differences.

We can see that the best combinations seem to be [9] KNN + Single Linkage - Single Sets - OnePerClass (blue) and [1 and 2] KNN + LDA - Single Sets (No Prototypes and OnePerClass). And although it could seem that [9] is closer to the 0, if we compare the difference in clicks (around +10.5 %) and in time (around +51%), then the KNN + LDA approach seems to do way better.

The other one of the best tests is Dasgupta's method [13], which is still the best talking just about clicks (Clicks [1] around + 58% and clicks [9] around + 29%), but is a way worse talking about the time (Time [1] around – 154% and time [9] around – 82%).

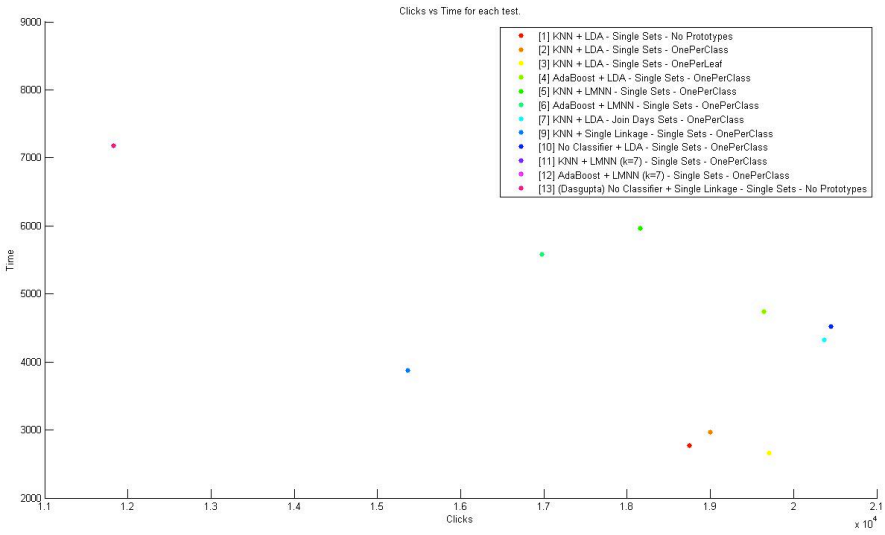

Fig. 45. Scatter plot with each of the tests (except [8]) represented by a coloured dot. The closer to the 0 time and clicks values a test is, the better the time/clicks rate will be. Note that the scale in the Clicks axis is considerably smaller than the Time axis.

6. Conclusions and Future Lines

In conclusion, we presented a novel Active learner method based on a combination of Forest Hierarchical Active Learning, supervised classification and dimensionality reduction techniques that make possible to label huge amounts of data (of order of 90.000 images). And last, but not least, we have found alternative algorithms for the Active Labelling process that can improve some of the results given by Dasgupta's approach, or give alternatives that could do it better with different kinds of features. Being as one reasonable improvement the time needed for the labelling of a set (4.000 images labelled manually) with a wide variety of labels: about 37 min with KNN + LDA + OnePerClass Prototypes, and about 43 min with Dasgupta's approach, even though the classical Dasgupta is oriented to the "Join All Sets" method, it leads to a greater time needed for the labelling.

Moreover, we developed an Active Labelling application that can be easily feeded with SenseCam images, or from any other source, and that can be very valuable for being used for the recognition of any kind of object, adapting their features and the Online Learners. We successfully applied our application to label food-related objects. To this purpose, a wide set of features was extracted to be used in the clustering and recognition process.

Furthermore, we have introduced a very useful and user friendly graphic user interface, which makes the labelling process quicker and a little less stressing.

To summarize the new contributions made to the research world (dimensionality reduction applied to active learning; trimming or initial screening used between every labelling session and forest learner for labelling a large number of samples in sets), we present them in three points:

i) New approach to Active Learning.
ii) New technique applied to food-recognition.
iii) New user-oriented application for labelling huge amounts of samples.

Overviewing the aspects that we have not been able to study in detail and improve, as future work we could:

i) Make the sets divisions for each labelling session smaller to find the point of optimality, referring to the time and the clicks needed.
ii) Continue researching about the prototypes and try to look for alternatives for a better benefit to the active labeller (changing their definition or using them during the branching process with the Active Labellers, LDA and LMNN).
iii) Create different types of extended ECOC Online Learners depending on the objects that we are trying to recognise.
iv) Apply to different objects and events to label and recognize in order to be able to extract evidences about the health habits of people based on lifelogging.

7. References

[1] A. Sellen and S. Whittaker (2010). Beyond total capture: A constructive critique of Lifelogging. Communications of the ACM. Vol. 53. No. 5.

[2] S. Hodges et al. (2006) SenseCam: A Retrospective Memory Aid. UbiComp 2006: Ubiquitous Computing. Lecture Notes in Computer Science. Volume 4206, pp 177-193

[3] Taiwo Oladipupo Ayodele (2010). Types of Machine Learning Algorithms, New Advances in Machine Learning, Yagang Zhang (Ed.), ISBN: 978-953-307-034-6, InTech.

[4] Mingkun Li and Ishwar K. Sethi (2006). Confidence-Based Active Learning. IEEE Transactions on Pattern Analysis and Machine Intelligence. Vol. 28. No. 8.

[5] Dasgupta, S., and Hsu, D. (2008, July). Hierarchical sampling for active learning. In Proceedings of the 25th international conference on Machine learning (pp. 208-215). ACM.

[6] S. Dasgupta (2011). Two faces of active learning. Theoretical Computer Science 412 (2011) (pp. 1767-1781).

[7] Abdi Hervé, Williams Lynne J. (2010). Principal component analysis. *WIREs Comp Stat* 2010, 2: 433-459. doi: 10.1002/wics.101.

[8] Yong Jae Lee; Ghosh, J.; Grauman, K. (June, 2012). Discovering important people and objects for egocentric video summarization. Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on , vol., no., pp.1346,1353, 16-21.

[9] Chang Xu et al. (2013). Model-Based Food Volume Estimation Using 3D Pose. IEEE ICIP 2013. Paper #2640.

[10] Ye He et al. (2013). Context Based Food Image Analysis. IEEE ICIP 2013. Paper #3111.

[11] Doherty, Aiden R. and Smeaton, Alan F. (2008) Automatically segmenting lifelog data into events. In: WIAMIS 2008 - 9th International Workshop on Image Analysis for Multimedia Interactive Services, 7-9 May 2008, Klagenfurt, Austria. ISBN 978-0-7695-3130-4

[12] Welling, M. (2005). Fisher linear discriminant analysis. Department of Computer Science, University of Toronto.

[13] Balakrishnama, S., & Ganapathiraju, A. (1998). Linear discriminant analysis-a brief tutorial. Institute for Signal and information Processing.

[14] Belhumeur, P. N., Hespanha, J. P., & Kriegman, D. J. (1997). Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 19(7), 711-720.

[15] Weinberger, K. Q., & Saul, L. K. (2009). Distance metric learning for large margin nearest neighbor classification. The Journal of Machine Learning Research, 10, 207-244.

[16] Seidl, T., & Kriegel, H. P. (1998, June). Optimal multi-step k-nearest neighbor search. In ACM SIGMOD Record (Vol. 27, No. 2, pp. 154-165). ACM.

[17] Simard, P., LeCun, Y., & Denker, J. S. (1992, November). Efficient pattern recognition using a new transformation distance. In Advances in Neural Information Processing Systems 5,[NIPS Conference] (pp. 50-58). Morgan Kaufmann Publishers Inc..

[18] Belongie, S., Malik, J., & Puzicha, J. (2002). Shape matching and object recognition using shape contexts. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 24(4), 509-522.

[19] Weinberger, K. Q., Blitzer, J., & Saul, L. K. (2006). Distance metric learning for large margin nearest neighbor classification. In In NIPS.

[20] Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. Journal of computer and system sciences, 55(1), 119-139.

[21] MacQueen, J. (1967, June). Some methods for classification and analysis of multivariate observations. In Proceedings of the fifth Berkeley symposium on mathematical statistics and probability (Vol. 1, No. 281-297, p. 14).

[22] Drozdzal, M., Seguí, S., Malagelada, C., Azpiroz, F., Vitrià, J., & Radeva, P. (2011). Interactive labeling of WCE images. In Pattern Recognition and Image Analysis (pp. 143-150). Springer Berlin Heidelberg.

[23] Radeva, P., Drozdzal, M., Segui, S., Igual, L., Malagelada, C., Azpiroz, F., & Vitria, J. (2012, July). Active labeling: Application to wireless endoscopy analysis. In High Performance Computing and Simulation (HPCS), 2012 International Conference on (pp. 174-181). IEEE.

[24] Baró, X., Escalera, S., Vitrià, J., Pujol, O., & Radeva, P. (2009). Traffic sign recognition using evolutionary adaboost detection and forest-ECOC classification. Intelligent Transportation Systems, IEEE Transactions on, 10(1), 113-126.

[25] Duin, R. P. W., & Loog, M. (2004). Linear dimensionality reduction via a heteroscedastic extension of LDA: the Chernoff criterion. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 26(6), 732-739.

[26] Drozdzal M., Seguí S., Radeva P., Vitrià J. (2011). An Adaptive Hierarchical Sampling Strategy for Active Learning.