CS 131 Computer Vision: Foundations and Applications
(Fall 2016)
Homework 5

1 Curse of Dimensionality

In class, we used the nearest neighbor method to perform face recognition. The nearest neighbor method is intuitive and very powerful, especially in low dimensional applications. When the number of dimensions becomes higher, however, we may not want to use this method directly because of its computational overhead and counterintuitive geometry. To work around these limitations, we introduced PCA to reduce the dimensions of the data.

(a) Suppose we have n data samples and w test points in d dimensional space, we want to find each test point's nearest neighbor based on Euclidean distance $\|\cdot\|_2$ where

$$\|x-y\|_2 = \sqrt{\sum_{i=1}^{d}(x_i-y_i)^2}$$

Please give the computational complexity of this procedure using *big-O* notation, and discuss what happens when d becomes very large.

(b) When performing nearest neighbor search using Euclidean distance, we can use a simple heuristic to speed up the search: we quickly reject candidates by using the difference in one dimension as a lower bound for distances. Explain in words why this heuristic will not help much in high dimensional space (i.e. when the dimension d is large).

(c) Given a unit cube in d-dimensional space $C = \{x \in \mathbb{R}^d | 0 \leq x_i \leq 1, i = 1, 2, \ldots, d\}$, show that nearly all the volume of C is concentrated near its surface when d is large. *Hint:* To get the volume of a cube near its surface, you can shrink the cube from its center by some constant c and subtract the volume of shrunk cube from the volume of the original one.

(d) Consider generating points uniformly at random on the surface of a unit-radius sphere in d dimensions. To get an intuition, let's first consider the 2-dimensional version of generating points on the circumference of a unit-radius circle by the following method:

- Independently generate each coordinate uniformly at random from the interval $[-1, 1]$. This produces points distributed over a square that is large enough to completely contain the unit circle.

- Discard all points outside the unit circle and project the remaining points onto circle (by normalizing them into unit 2d vectors).

Can we generalize this technique in the obvious way to higher dimensions? What will happen when d becomes very large? Use your result from part (c) to explain.

(e) Please list at least two methods you could use to overcome problems with high-resolution images (i.e. data points in very high dimensions).

2 1-NN with Different Distance Metrics

Given two vectors x and y in n-dimensional space $\mathbb{R}^n$, we can measure distances between them in multiple ways:

- Eulidean distance (or ℓ_2 distance)

$$Dist^{\text{Euc}}(x, y) = \|x - y\|_2 = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

- Cosine distance

$$Dist^{\text{Cos}}(x, y) = 1 - \frac{x^T y}{\|x\|_2 \|y\|_2}$$

- Mahalanobis distance

$$Dist^{\text{Mah}}(x, y, \Sigma) = (x - y)^T \Sigma (x - y)$$

(a) Let Σ be an identity matrix. Prove that:

$$Dist^{\text{Mah}}(x, y, \Sigma) = \|x - y\|_2^2$$

for any x and y. The right hand side of equation is the squared Euclidean distance.

(b) Let $x_0, \ldots, x_K \in \mathbb{R}^n$. Consider the set of points that are closer (in Euclidean distance) to x_0 than the other x_i, i.e.,

$$V = \left\{ x \in \mathbb{R}^n \middle| \|x - x_0\|_2 \leq \|x - x_i\|_2, \quad i = 1, \cdots, K \right\} \tag{1}$$

Show that the set V can be described as $\left\{ x | Ax \preceq b \right\}$, where $Ax \preceq b$ denotes:

$$\begin{aligned} a_{11}x_1 + \cdots + a_{1n}x_n &\leq b_1 \\ a_{21}x_1 + \cdots + a_{2n}x_n &\leq b_2 \\ \vdots \qquad\qquad &\leq \vdots \\ a_{K1}x_1 + \cdots + a_{Kn}x_n &\leq b_K \end{aligned}$$

In your answer, please make sure to specify what A and b are.

(c) Continuing from part (b), the set V defined in equation (1) is called the *Voronoi region* around x_0 with respect to $x_1, \ldots, x_K$. Let's consider the reverse problem of part (b). Given a non-empty set $V = \left\{x | Ax \preceq b\right\}$, describe in words how to find $x_0, \ldots, x_K$ so that V is the Voronoi region of x_0 with respect to $x_1, \ldots, x_K$.

Hint: The goal is not to find the original, unique points that produced the Voronoi set V. That problem would be under-constrained (fewer equations than unknowns). Instead, all you need to do is describe how to find one possible set of points $x_0, \ldots, x_K$ that will produce that same Voronoi region around x_0 (it won't necessarily produce the original Voronoi regions around the other points).

(d) Using Euclidean distance, we can define a family of sets

$$V_k = \left\{x \in \mathbb{R}^n \middle| \|x - x_k\|_2 \leq \|x - x_i\|_2, \quad i \neq k\right\}$$

The set V_k consists of points in $\mathbb{R}^n$ for which the closest point in the set $\{x_0, \ldots, x_K\}$ is x_k. The family of sets $V_0, \ldots, V_K$ gives a decomposition of the space $\mathbb{R}^n$.

Using Cosine distance, we can also define a family of sets

$$W_k = \left\{x \in \mathbb{R}^n \middle| Dist^{\text{Cos}}(x, x_k) \leq Dist^{\text{Cos}}(x, x_i), \quad i \neq k\right\}$$

Assume $x_0, \ldots, x_K$ are unit vectors. Prove that the decompositions $V_0, \ldots V_K$ and $W_0, \ldots, W_K$ are equivalent. Two decompositions of $\mathbb{R}^n$ are equivalent if and only if for any $a \in \mathbb{R}^n$,

$$a \in V_k \iff a \in W_k$$

(e) Assume, again, that $x_0, \ldots, x_K$ are unit vectors. Would choosing Euclidean 1-NN over Cosine 1-NN make a difference? Please justify your answer.

(f) Consider the following in two dimensional space:

$$x_0 = (0, 3) \quad x_1 = (-3, 0) \quad x_2 = (3, 0) \qquad \Sigma = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

In particular, consider the decomposition U_0, U_1 and U_2 given by

$$U_k = \left\{x \in \mathbb{R}^n \middle| Dist^{\text{Mah}}(x, x_k) \leq Dist^{\text{Mah}}(x, x_i), \quad i \neq k\right\}$$

where $k = 1, 2, 3$. Use Matlab to plot this decomposition. Submit your code and plot for this part of the problem. *Hint:* To plot a decomposition, you only need to plot the "decision boundaries" between set U_i and set U_j.

3 PCA as Fitting Lines

PCA projects the data by an orthogonal matrix. This problem shows how to obtain this orthogonal matrix, as well as the intuition behind it. For simplicity, you are only required to derive PCA in two

dimensional space. After finishing this problem, you can easily extend the results to n-dimensional spaces.

Suppose we have a dataset $\mathcal{D} \in \mathbb{R}^{m\times 2}$ in $\mathbb{R}^2$, where each data sample $x^{(i)}$ is one row of the data matrix. We want to approximate our high dimensional data by an orthogonal line set L in two dimensional space

$$L = \{az + b | z \in \mathbb{R}\}$$

where $a \in \mathbb{R}^2$, $b \in \mathbb{R}^2$, and a is a unit vector.

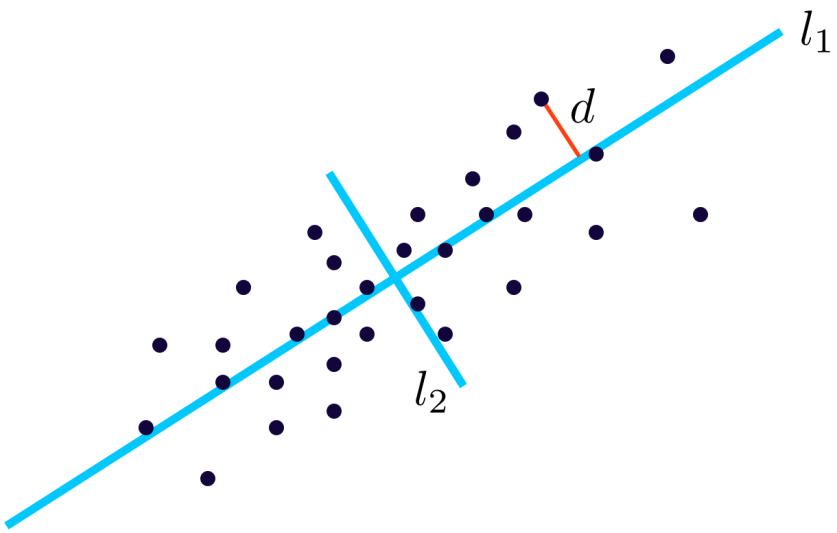


Figure 1: An illustration of fitting a two dimensional dataset to a line set $L = \{l_1, l_2\}$. Each black dot is a data sample.

For each data sample $x^{(i)}$, the distance between x and a line l is defined by

$$d(x^{(i)}, l) = \min_{y\in l} \|x^{(i)} - y\|_2,$$

where $\|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$ (ℓ_2 norm), n is the data dimension and it is 2 in our case. Since we have m data samples, we start by finding a single line set l that minimizes the total squared distance:

$$d_t = \sum_{i=1}^{m} d(x^{(i)}, l)^2$$

In other words, we'd like to fit the data samples to a set of straight lines, as close as possible such that d_t is minimized.

(a) Show that $d(x, l) = \|(I - aa^T)(x - b)\|_2$. *Hint: you might consider using a least squares approach.*

(b) Using your result from (a), the total squared distance is given by

$$d_t = \sum_{i=1}^{m} \|(I - aa^T)(x^{(i)} - b)\|_2^2$$

Show that the optimal b, which minimizes d_t while all other variables are held constant, is given by

$$b_{\text{opt}} = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}$$

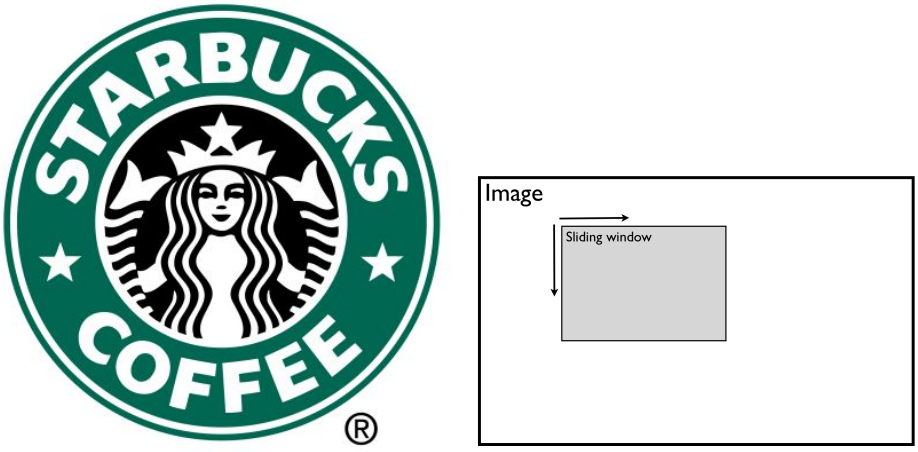
Here, b_{opt} is the mean (center) of our data set. *Hint:* $P = (I - aa^T)$ *is a projection matrix.*

(c) Using this choice of $b = b_{\text{opt}}$, find the optimal value of a which minimizes d_t, in terms of $x^{(i)}$ and b_{opt}. *Hint:* Try to formulate the problem as a maximization problem of the form $\max(a^T \Sigma a)$. The linear algebra review notes from the first week of class may prove useful here.

(d) So far, our results only give us a single line $l = a_{\text{opt}} z + b_{\text{opt}}$ $(z \in \mathbb{R})$. Our final goal is to approximate the 2D data by an *orthogonal* line set L in two dimensional space. Describe how you would choose the other line to add to the set L. *Hint:* The procedure should be very similar to the steps in part (a)-(c).

4 Logo Detection

A lot of computer vision algorithms represent visual data in the form of histograms (such as SIFT, Shape Context), which can be easily handled by computers but allow for "slop" in the exact pixel values and locations. In this problem, we are going to develop a simple detector for Starbucks logos using histograms of image colors, and we'll use a sliding window search to apply our detector to an image.



(a) A template Starbucks logo

(b) Sliding window search

Figure 2: Problem 5 illustration.

Suppose we have a template Starbucks logo, as given in Figure 2(a). For some given images (under the `logo/data` folder), we want to find regions that are visually similar (i.e. likely another Starbucks logo).

We'll use a sliding window search as shown in Figure 2(b), where we slide a fixed size rectangle across the entire image and compare the contents of each rectangle with the given template image. We repeat the process with multiple window sizes. The sliding window code is provided for you; in this problem you will only need to handle the comparison.

We will use color histograms to compare the template to each image rectangle that is grabbed by the sliding window. We'll build a histogram for the image inside the window, compare it against the histogram of the template, and record the sliding window position which yields the best match.

Here is a list of functions you will use:

- `Histogram.m` builds a color histogram from a given image patch.
- `SlideWindowDetector.m` compares the template to all possible image regions, and saves the one with the best match.
- `VisualizeBox.m` visualizes the detection result from `SlideWindowDetector.m`.

(a) In this part, you need to implement two histogram measures and use these two measures to find the best match to the Starbucks logo in several images.

- Histogram intersection distance

$$Dist^{\text{Intersection}}(x, y) = \sum_{i=1}^{n} \min\{x_i, y_i\}$$

  This distance is a similarity measure, i.e. the higher the distance is, the more similar two histograms are.

- Chi-square distance

$$Dist^{\text{Chi-square}}(x, y) = \sum_{i=1}^{n} \frac{(x_i - y_i)^2}{x_i + y_i}$$

  This distance measures the proximity of two histograms, i.e. the lower the value is, the more similar two histograms are. Note: for entries where the denominator above is 0, just add 0.

Complete `HistIntersectDist.m` and `ChiSquareDist.m`, and then test your code by running `LogoRecognition.m` after uncommenting the relevant function call. Please submit printout of your code for this part of the problem.

(b) From part (a), we can see that our algorithm is not very accurate. This is partly because a color histogram of an entire image is a quite weak representation (for example, it discards all the spatial information). In this part, you are going to incorporate spatial information into the color histogram by implementing "Spatial Pyramid"[1].

Read section 3 of the included "Spatial Pyramid" paper [1], and implement its equation (3) in `SpatialPyramidDist.m`. The basic idea is to compute color histogram distances between multiple

sub-regions of the images being tested, so that spatial information is not entirely discarded (e.g. when our sliding window is looking for images of the Starbucks logo, the images will need black pixels near the center, rather than just anywhere). Figure 3 illustrates the histogram-building process.

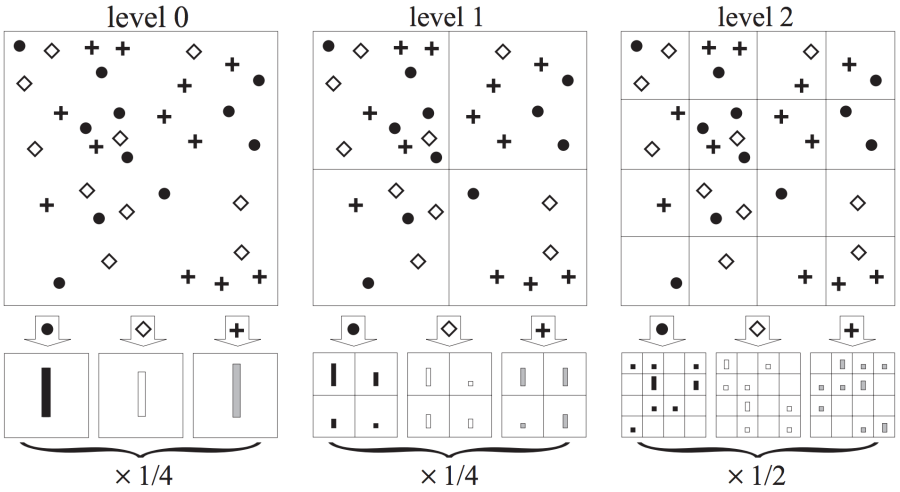


Figure 3: Spatial pyramid

Please submit a printout of your code and an image showing a successful logo detection.

(c) Of the three histogram-based image-comparison methods above, which are invariant to rotations? Explain your answer.

5 Face Recognition

5.1 Introduction

In this assignment, youll train a computer to recognize Sarah Connor, the protagonist of the 1984 movie *The Terminator*.

We have provided you with a training set that includes labeled images of Sarah Connor and other humans faces under multiple lighting conditions and with multiple facial expressions, but with the faces perfectly aligned. In other words, the subjects eyes are always at the same pixel locations, so that faces can be compared to each other easily.

These faces were identified and aligned automatically, via the Viola-Jones algorithm. We wont go over that algorithm in this class, but its basically a way to effectively do a series of cross-correlations to locate face features such as eyes.

(a) Well begin with a simple pixel-by-pixel method of face comparison. In *compareFaces.m*, we have provided code that reads in all the labeled training images as a single matrix. Each column of

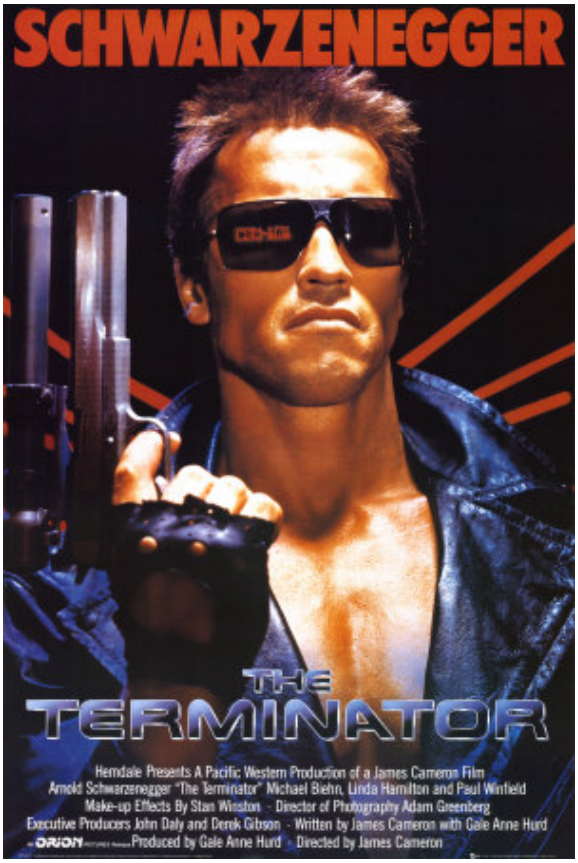


Figure 4: Your End User

the matrix is an image which has been unrolled column-wise (first column, second column, etc) to form one long column vector. You will complete the Part 1 sections of code to compare a provided test image to each of these column vectors. If the most-similar column vector is an image of Sarah Connor, we will decide that the test image is Sarah Connor.

Edit the main *compareFaces()* function to take the given test image and unroll it into a vector. Edit the *indexOfClosestColumn()* function to perform a Euclidean distance comparison to each column from the training data. Finally, edit the main *compareFaces()* function to return the label of the training example which was the closest match to the test face. Read and run *compareFacesTest.m* with the first few test faces to confirm that this simple method works.

(b) What you did above is called a nearest-neighbor algorithm. Note that you didnt use the training data to build a computer model of Sarah Connor. Instead, you just kept all the training data, compared the new image to every training example, and gave it the label of the closest one.

(c) **In your writeup**: what are the advantages and drawbacks of the nearest-neighbor approach to classification, in general? (Be sure to discuss computational difficulties if you have a very large

databases of faces.)

5.2 Eigenfaces

The "Eigenfaces" algorithm is a more efficient version of what you did above. It is still a nearest-neighbor algorithm, but it uses Principal Component Analysis (PCA) to represent each face with just a few numbers. You may want to review our Linear Algebra slides on PCA before continuing. You will complete the Part 2 code in compareFaces.m and doPCA.m to perform the same comparisons as above, but using the PCA form of the face vectors.

(a) We will use the same matrix of training examples as before, and we will reuse the code which finds the mean column of this matrix and subtracts it from every column. This will prevent PCA from representing patterns that are the same in every image.

(b) In *doPCA.m*, use the SVD command to perform PCA, and retain the top *numComponentsToKeep* principal components (also known as the basis vectors.) You should use the *SVD(A,econ)* form of the command, as it is much more efficient when we only need the top few components, as we do here. Also, retain the matrix of component weights:

$$W = \Sigma_{short} V_{short}^T \tag{2}$$

where short indicates a matrix which only contains entries that affect the top *numComponentsToKeep* principal components. (E.g. Σ_{short} is size *numComponentsToKeep* x *numComponentsToKeep*.)

(c) **In your writeup**: What do the weights represent? (Be specific about what each entry in W represents.) How could you use the weights to reproduce the faces that they represent? (You may use MATLAB to verify your answer if desired the provided *viewFace* function will reshape an unrolled face column back into an image and display it.)

(d) In *compareFaces.m*, call your *doPCA()* function to produce basis vectors and weights for the training images. Then, convert your column-vector test image to PCA space. In other words, convert it to a vector of weights on the PCA principal components which could be used to reproduce the original image. *Hint*: the PCA basis vectors are unit vectors. The dot product of a vector x with a unit vector gives the component of x which lies is in the direction of that unit vector.

(e) Use your *indexOfClosestColumn()* function from before to compare the weight vector of the test image (i.e. its representation as a "PCA space" vector) to the PCA space vectors of each training example, to choose the nearest neighbor. Use *compareFacesTest.m* to verify that your algorithm works as before. **In your writeup**: If we wanted to build a mobile robot which can recognize Sarah Connor, what are the advantages of PCA-space representation of face images? Consider the effects on storage and on computation.

5.3 Fisherfaces

Above, PCA found the basis vectors which were best to reconstruct the faces. (That is, they capture the most possible variance across all images.) However, our purpose here is not to reconstruct the faces from their PCA-space representations, its to compare their PCA-space representations. The Fisherfaces algorithm chooses basis vectors which are more optimal for our comparison task. Given

images and a class label for each image, it will choose basis vectors which capture the most between-class variance, and the least within-class variance. Thus, the resulting weight vectors will be better for distinguishing between classes.

(a) In *compareFacesTest.m*, uncomment the Sarah Connor, higher brightness test image. Run it with the Eigenfaces comparison method and again with the simple pixel-by-pixel comparison method, and observe that both fail.

(b) Complete the "Part3" code in *compareFaces.m* to call the provided *fisherfaces()* function in *fisherfaces.m*. Use the basis vectors and weights which it produces to do the comparison instead. It should successfully match the Sarah Connor, higher brightness test image.

(c) **In your writeup**: Explain why Fisherfaces worked better in this specific case. It may help to look at the training images and test images.

5.4 Identification of humans

In the intro, we mentioned that fast algorithms exist to locate faces in an image. But these algorithms occasionally make a mistake, and grab an image region which is not a face. If we are given such a non-face region, we would like to reject it, rather than blindly try to match it to a face.

Design your own method to decide if a given image is a human face or not. The function in *isFace.m* will be given an image, and it must return true if it is a human face, and false if it is not. (If the image is a human face, you can assume that it will be aligned like the faces in the training set. Also, all images will be the same size as the ones in the training set.) You may reuse any code from this homework or previous ones. You can use *isFaceTest.m* to test your method.

Some tips to get you started:

1. You have a lot of useful code in *compareFaces.m*: the basis vectors, the mean face, etc. You can also use other things like MATLABs edge function.
2. If you wish, your code may use the provided faces in the "facedatabase" and the provided non-face images in "nonfacedatabase".
3. If you make use of basis vectors, be aware that the top few PCA vectors tend to capture general brightness patterns, rather than more face-specific patterns. So if you use basis vectors, you may want to experiment with discarding the first 3 to 5 basis vectors.
4. Note that the images are grayscale, so the color-segmentation technique of problem 5, below, will not be useful here.

Grading: For full credit, your method must achieve at least 65% accuracy on the *isFaceTest.m* test set. (Your code may not use the images in the *facetestset* directory, and it must not do any form of memorization or hard-coding of the right answers.)

In your writeup: Make sure to explain in detail exactly how you implemented your function. What design decisions did you make while implementing this function? Also report your accuracy score on the *facetestset*.

5.5 Writeup

In your writeup, answer all writeup questions from above: 5.1(c), 5.2(c), 5.2(e), 5.3(c), and 5.4. Also answer these questions:

1. Why did we have to align the face images? Discuss the effect on the PCA basis vectors if the labeled training faces were not aligned.
2. How would the nearest-neighbor algorithm behave if you had a very small database of labeled faces? (For example, suppose your database only had images of Sarah Connor and one other human.) Explain your answer.
3. Describe how your *isFace()* algorithm works and why it responds to faces.

Make sure to include all the code you write in the writeup as well. Also, include in the writeup any images that you generate in this assignment.

References

[1] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 2169–2178. IEEE, 2006.