PRACTICAL 7

Dimensionality Reduction and Evaluation

In this practical, we are going to explore different techniques for doing dimensionality reduction. Finally, we will calculate some metrics to evaluate more in depth our algorithms. As application examples, we will use code from previous practicals (Digits, prac 5) as well as face recognition.

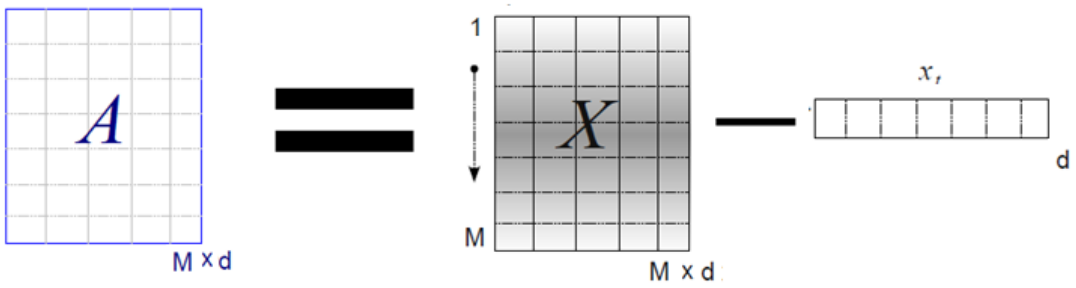
Download from Canvas the provided functions to assist you during the practical as well as the clips and image sequences to be used.

TASK 1: PCA AND EIGENFACES

In this task we are going to make use of PCA to calculate a model of Eigenfaces. First, we will complete the function to perform PCA. Then we will use it on face data.

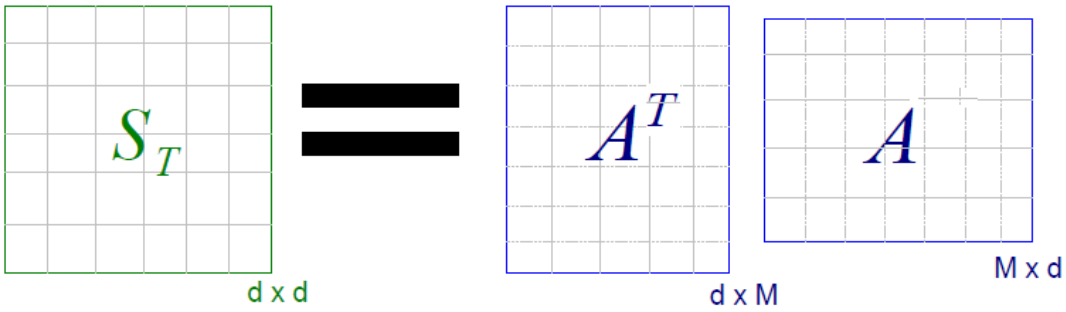
In order to assist you in the process, you have some Matlab functions available as well a dataset containing examples of human daces under different conditions.

STEP 1: Open the function `PrincipalComponentAnalysis` and complete the first 2 instructions. In the first instruction you will be calculating the mean sample. In the second instruction you will subtract this mean sample from each sample in matrix X and store the result in matrix A.

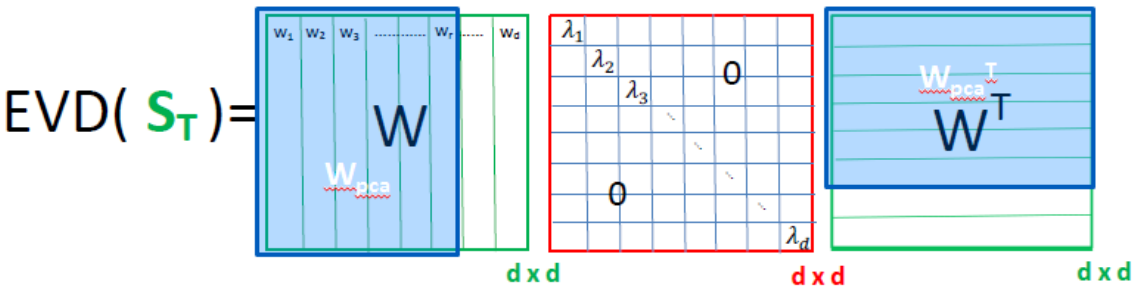


Please remember that the two hour duration of a practical class will probably not be enough time to complete all sections. There is an expectation that you will work outside of class as an independent learner.

STEP 2: Complete the 3rd instruction where you will obtain the scatter matrix S by using previous matrix A.



STEP 3: Read the rest of the code and try to understand how the dimensions containing most of the information are extracted and selected



$$\text{where } \Lambda = diag(\lambda_1 > \lambda_2 > \cdots > \lambda_r > \cdots > \lambda_d).$$

$$W_{pca} = [w_1 \; w_2 \; \cdots \; w_r]_{(dxr)}$$

where w_i is each of the extracted eigenvectors, W_{pca} is the matrix containing all eigenvectors, and λ_i is the eignevalue associated to each eigenvector.

Now, that we are confident about pca, we can proceed to process the face dataset database and observe the meaning of the eigenvectors graphically. For this task, we will be using the Yale face dataset.

STEP 4: Open the script Eigenfaces. Observe how the dataset is loaded. Add a piece of code to visualise some of the images in the training set.

STEP 5: Apply PCA on the full Yale dataset by using the function `PrincipalComponentAnalysis`. Fix the number of reduced PCA dimensions to 15.

STEP 6: Run the rest of the script to observe the average image and the 15 most significant eigenvectors. If everything works fine, the result should be similar to the figure at the beginning of the task.

These eigenvectors (or eigenfaces) represent the main variation modes in which the average face can change/be transformed

To understand even better the function of the eigenvectors, we will select one of them (let's start with the first one) and we will move in that dimension only.

```
%animation for observing the variation of the first eigenvector
eigVector_index=1;
```

The range of values that we will move from the average is given by the next equation:

$$-3 \cdot \sqrt{\lambda_i} < w^i{}_{pca} < +3 \cdot \sqrt{\lambda_i}$$

```
    weights = [-3*sqrt(eigenvalues(eigVector_index)):
6*sqrt(eigenvalues(eigVector_index))/200: 3*sqrt(eigenvalues(eigVector_index))];
```

STEP 7: Write, at the end of your script, the 2 previous line of code plus the next fragment

```
figure
for b=weights
    faceReconstruct = meanX + b*eigenVectors(:,eigVector_index)';
    faceReconstructImage = reshape(faceReconstruct,[h w]);
    imagesc(faceReconstructImage), colormap(gray), axis equal, axis off,
    drawnow
end
```

which will implement the equation

$$X = x_t + W_{pca}{}' \cdot X_{pca}$$

and visualise the results for every weight in the range, creating a little animation. Look at the result, what kind of variation happens in the face when moving along the first eigenvector only?

STEP 8: Now change the value of `eigVector_index` to 2. Observe the animation. Is the same transformation that in the previous step? What kind of variation does the second eigenvector introduce?

TASK 2: DIMENSIONALITY REDUCTION IN CLASSIFICATION

In this task we are going to make use of PCA to reduce the computational time and/or improve the performance in a pattern recognition problem. To illustrate this, we will be using the code you developed during practical 5 for digit recognition.

STEP 1: Open the script OCR_KNN_09 (or similar) where you applied K-NN to recognised all digits from 0 to 9. Run the code again and write the accuracy value down.

STEP 2: Apply PCA to the full image dataset before performing the training (before calling `NNtraining`). Again, make use of the previous function `PrincipalComponentAnalysis` to perform this task. You do not need to give any value to the number of desired dimensions. Now, pass the reduced data Xpca to the training function.

STEP 3: Similarly, we will apply a similar process to the testing data before passing it to the classifier. In the testing loop, right after separating the test image, implement the equation:

$$x_{pca} = (X - x_t) \cdot W_{pca}$$

Now pass the reduced test image x_{pca} to the testing classification function.

STEP 4: Run the full script. Observe the new accuracy and the computation time (you can use tic/toc). Which conclusion can we extract?

Finally, we are going to see if another dimensionality reduction method can provide better or similar results. In this case, we will explore the use of LDA, a similar technique to PCA but aiming to better separate the classes in the reduced space.

STEP 5: Replace the line were you are performing PCA in the training stage by the following one

```
[eigenVectors, eigenValues, meanX, Xlda] = LDA(labels,[],images);
```

which will perform LDA instead. You do not need to make any major change in the testing stage since the projection in the LDA space is given by the same equation than in the PCA space.

$$x_{lda} = (X - x_t) \cdot W_{lda}$$

STEP 6: Change the visualization of the data samples in the reduced space (now we know it was PCA) to display the LDA space instead. You can reuse almost all the code but replacing the matrix `X_reduce` containing the points by the matrix `Xlda`.

STEP 7: Run the code. Observe the 10 classes are now distributed in the LDA space. Is that better or more convenient than before? Look at the accuracy number now. Has it improved or deteriorated? Is this coherent with the visualisation? Elaborate your theory about why this is happening.

TASK 3: EVALUATION METRICS

To understand better which digits are better or worse recognised, overall accuracy is not enough.

STEP 1: Calculate the Confusion matrix for the previous classifier with digits from 0 to 9. Since 10 classes are available, the resulting matrix should be 10 x 10 in size.

Remember than a confusion matrix should have the following characteristics:

- The diagonal contains the accuracy for each class
- The other cells contain the normalised count of the errors
- Every row should add 1

Let's move now to a binary classifier.

STEP 2: Open the pedestrian classifier that you implemented in task 1 during the Practical 6. Go to the evaluation section and calculate the value of TP, FP, TN and FN

	Predicted class	
True Class	Yes	No
Yes	TP: True Positive	FN: False Negative
No	FP: False Positive	TN: True Negative

STEP 3: Calculate the values of the Recall, Precision, Specificity, Sensitivity, F1 and False alarm rate.

- **Recall** = # of found positives / # of positives = TP / (TP+FN) = **sensitivity** = **hit rate**
- **Precision** = # of found positives / # of found = TP / (TP+FP)
- **Specificity** = TN / (TN+FP)
- **F-measure (F1)** = 2x(**precision** x **recall**)/(**precision** + **recall**) = 2xTP / (2xTP + FN + FP)
- **False alarm rate** = FP / (FP+TN) = 1 - Specificity