HW3

1 Clustering (10%)

In the lecture we saw the K-means algorithm for clustering. This algorithm tries to minimize the Euclidian metric between the examples and some point in space which is named "centroid". Other methods try to minimize dissimilarities between the pairwise data examples. A classical algorithm which was designed to handle pairwise data is the K-medoid. This algorithm seeks to find a set of cluster representatives (named medoid) in the dataset and assign other examples to them. The algorithm randomly picks a k-set of medoids from the data and assigns points to each medoid based on their L_1 distances to that medoid. Then, it iteratively tries to improve the assignment by swapping assigned medoid points with non-medoid points until the energy of the entire system (which is measured by the sum of L_1 distances between medoid points and their assigned data points) is minimized.

a. Is K-medoid more robust to noise (or outliers) than the K-means algorithm? Explain your answer.

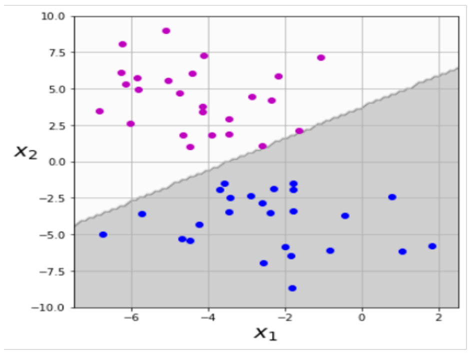
b. Prove that for the 1D case ($x \in \mathrm{R}^1$) of K-means, the centroid (μ) which minimizes the term $\sum_{i=1}^{m}(x_i - \mu)^2$ is the mean of m examples.

c. Prove that for the 1D case ($x \in \mathrm{R}^1$), the centroid (practically, the medioid) which minimizes the term $\sum_{i=1}^{m}|(x_i - \mu)|$ is the median of m examples given that μ belongs to the dataset. You can add a verbal explanation to the cases where m is an even number if needed.

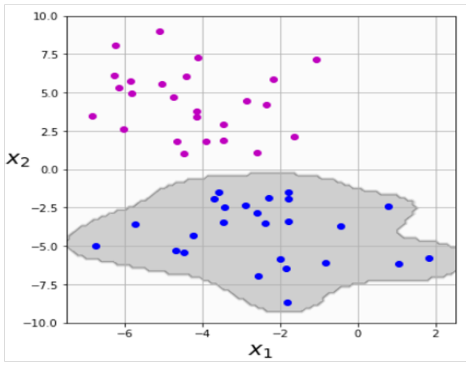
2 SVM (30%)

In the following figures you can see a visualization of SVM running with different settings (kernels and parameters) as follows:

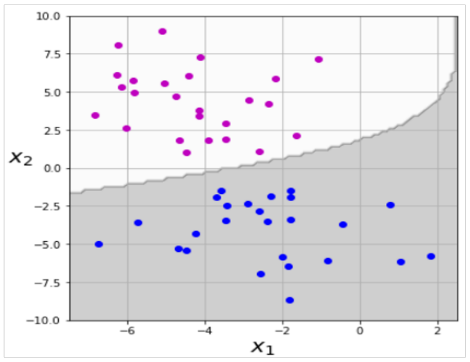
A



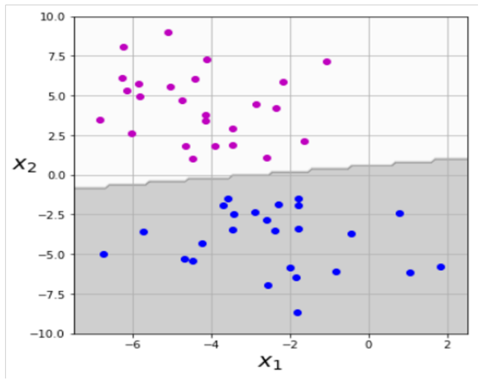
B



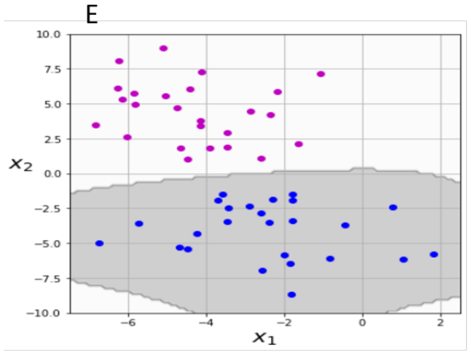
C



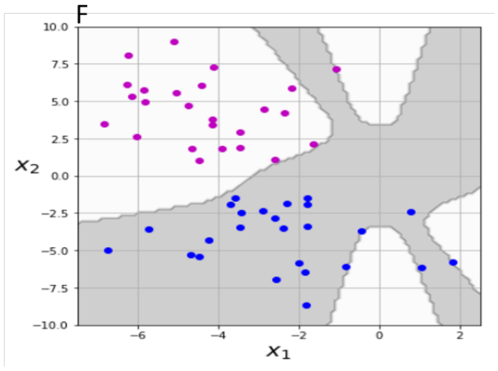
D



E



F



The settings that were used are as following:

1. Linear kernel with $C = 0.01$.
2. Linear kernel with $C = 1$.
3. 2^{nd} order polynomial kernel.

4. 10th order polynomial kernel.
5. RBF kernel with $\gamma = 0.2$.
6. RBF kernel with $\gamma = 1$.

Just to make things clear: the blue and purple colors represent the true labels and the gray and white areas represent the classification, i.e. if all of the blue dots are within the gray area then all of predictions were correct.
Match every image (labeled by a capital letter) to its' setting (number). Explain each of your answers.

Notice: An unexplained answer will not be marked even if it were correct!

3 Capability of generalization (20%)

Ockham's razor states that "Non sunt multiplicanda entia sine necessitate". This is known as the law of parsimony and its' translation to English is "Entities are not to be multiplied without necessity". This concept, which is attributed mostly to the English Franciscan friar William of Ockham, basically means that the simplest explanation is usually preferred.
A more modern variation of this concept (and a much more readable one) is attributed mostly to Albert Einstein and it states that "Everything should be made as simple as possible but not simpler". This balance is a major guideline in science in general and in data science in particular.
We saw in the tutorial two methods of choosing a parsimonious model for K-means. In GMM, there is another criterion to do so and it is known as "Akaike information criterion" (AIC). It is composed of two terms and defined as follows:

$$AIC = 2p - 2ln(\hat{L})$$

where p is the total number of learned parameters and $\hat{L}$ is the estimated likelihood given these parameters.

a. What is the scientific term of the balance that Einstein meant to in machine learning aspect?

b. How does each of the terms $(2p, 2ln(\hat{L}))$ in AIC affect the terms of the balance you defined in (a)?

c. What are the two options that are likely to happen if this balance was violated?

d. What are we aiming for with the AIC? Should it be high or low? Explain.

4 EigenFaces (40%)

The aim of PCA is to find a new space (which happened to be orthornormal) where we can project our data onto it so the variance of the projected data is

maximal. Thus, we get a new "coordinate system" where we can represent our data. The coordinate system is composed of the eigenvectors of the covariance matrix of the data. One of the most interesting projects done in this field is known as "EigenFaces", first published by Sirovich and Kirby in 1987. In this work, many human faces images were collected, centered, flattened into vectors and stacked into a matrix X. The eigenvectors (u_i) of XX^T were extracted and once reshaped back to the image size (vector $\rightarrow$ matrix), the eigenfaces were revealed. These faces that looked like ghosts were the orthonormal basis for the construction of every human face image!
Let's say that we would like to reconstruct your face out of the faces used to build the new coordinate system. If your face image (f) is an hXw matrix, then the flattened image is now a vector with hw elements. Using centering and orthonormality, your flattened and centered image can be calculated as follows:

$$f = \sum_{i=1}^{hw} \langle f, u_i \rangle u_i$$

Due to the fact that our axes are actually *principal axes*, we can approximate your face image as:

$$f \approx \sum_{i=1}^{K} \langle f, u_i \rangle u_i = \sum_{i=1}^{K} c_i u_i$$

where $K << hw$. This approximation of course implies if the eigenvectors are **ordered** by their eignevalues i.e. u_1 has a larger eigenvalue than u_2 which has a larger eigenvalue than u_3 and so on and so forth. Once "corrected back" and reshaped as an hXw matrix, you should see your face. Notice that we treat every pixel as if it were a feature but because all of the pixels have the same range (grayscale values) then scaling is not needed.

Specific task:
In the attached Jupyter notebook you are asked to build the orthonormal basis using PCA and to reconstruct your own face image :). You will also try to apply face recognition as written below. Before you start this task, take three images of your face with less background as possible (crop your image as needed). The first image will be with your face looking directly at the camera and should be called "Orig". The second one will be similar but with your head slightly tilted to one side. This should be called "tilt1". The third one should be with either your head tilted to the opposite side or with a straight hold head that is slightly facing off the camera and will be called "tilt2". Save them in the "Data" folder.

Eigenfaces are also used for face recognition. Each and every individual has his own set of $c_i's$ which are stacked in a vector c that in some aspect represents a new type of an ID. These vectors and the adequate ID's are saved in a database. Once a person needs to be recognized by the algorithm, his/her new face image is centered and then projected onto the same eigenvectors and new $c_i's$ are calculated. This new c vector is then run through the database and the Euclidiean metric is measured between this vector and any other c vector in the databse. The coupled ID of the c vector (of the the database) that results in the minimal Euclidiean distance is the ID of the examined person. At the end of the notebook, you will try to simulate this face recognition used by many companies such as Facebook.

Please make sure that the package `pillow` appears when you activate your venv and then type `conda list`. If it is not there, simply type `pip install pillow`.

Submission: Make sure all of your code runs the whole way through before submitting! You should submit a single pdf file with the answers for the theoretical part and your implemented notebook. Either way, you should not upload your personal images to Github. If, in addition, you don't want us to see the results you've got with your personal images, it is perfectly fine that you submit your **implemented** notebook after your restarted your kernel and cleaned all outputs. We will run your code with different images.

GOOD LUCK!