# 1 PCA & Eigenfaces [40 pts]

In this question, you are expected to analyze facial images using PCA. You will be working on the UTKFace dataset[1] in this part of the assignment. For simplicity, only the subset of the aligned and cropped faces, `crop_part1.tar.gz`[2], is requested to be used in the experiments. The dataset is composed of 9780 images of human faces. For this question, use of any library for PCA calculations is not allowed. You are requested to implement the PCA algorithm yourself. It is allowed to use the `numpy.linalg.eig` or `numpy.linalg.svd` functions to find the eigenvalues and the eigenvectors in your calculations.
Before the analysis, resize the images to $64 \times 64$ pixels by using the nearest neighbor resampling filter[3]. Then, flatten all images of size $64 \times 64 \times 3$ to get $4096 \times 3$ matrix for each image. Remember that most of the libraries read the image files in uint8 format. Since unsigned integer values cannot be negative, the following calculations may fail due to the restriction. In such a case, the problem can be solved by converting the data type to int or float32.

Note that all images are 3-channel RGB. Create a 3-D array, $X$, of size $9780 \times 4096 \times 3$ by stacking flattened matrices of the images provided in the dataset. Slice $X$ as $X_i = X[:, :, i]$ where i corresponds to the first three index. thus obtaining each color channel matrix independently for all images. Reshape all $X_i$ to obtain matrices, instead of 3D arrays.

**Question 1.1 [10 pts]** Apply PCA on $X_i$'s to obtain first 10 principal components for each $X_i$. Report proportion of variance explained (PVE) for each of the principal components, and their sum for each $X_i$. Discuss your results.
**Question 1.2 [5 pts]** Using the first 10 principal components found for each color channel, reshape each principal component to a $64 \times 64$ matrix. Stack corresponding principal components of each color channel, after min-max scaling the values, to obtain 10 RGB images of size $64 \times 64 \times 3$ and display all. Discuss your results.
**Question 1.3 [10 pts]** Describe how you can reconstruct an original facial image using the principal components you obtained in question 1.1. Use first $k$ principal components to analyze and reconstruct the first image [4] in the dataset where $k \in \{1, 50, 250, 500, 1000, 4096\}$. Discuss your results.

**Hint:** Do not forget to use the mean values in the reconstruction process, which you subtracted from the data to calculate the principle components.

[1] https://susanqq.github.io/UTKFace/
[2] https://drive.google.com/drive/folders/0BxYys69jI14kU0I1YUQyY1ZDRUE
[3] For the PIL library: PIL.Image.open(image_path).resize((64,64), Image.NEAREST)
[4] The order of the images may differ in different operating systems. The name of the first image: `100_1_0_20170110183726390.jpg.chip.jpg`

## 2 Logistic Regression [35 pts]

One of the largest national banking institutes in digital banking, Yapı Kredi[5], hired you as a data scientist. As your first task, you are asked to develop a system that can detect suspicious transactions that can be considered as fraud. The system that you are going to develop must be fast, so you decided to use a Logistic Regression Classifier for this task. According to the feedback that the banking system gets from your model, it will cancel the transaction and notify the customer involved in the transaction.

For this task, you are provided a subset of Credit Card Fraud detection dataset[6]. This dataset includes PCA features extracted from past banking transactions, which are given as V1-V28. Additionally, you are provided with the `Amount` of the transaction (how much money is transferred), `Time` (how much time completing the transaction took) and `Class` (1 if transaction is fraud, 0 otherwise). The subset of the dataset is fairly-balanced, which includes 452 fraud transactions and 600 non-fraud transactions. The dataset is already divided into training and test sets (you will use it as a validation set), and provided to you as `csv` files (Ratio is roughly 80% to 20%). The corresponding files are as follows:

- `q2_train_samples.csv`
- `q2_train_labels.csv`
- `q2_test_samples.csv`
- `q2_test_labels.csv`

Maximum Conditional Likelihood Estimators (MCLE) for Logistic Regression classifier with weights $w$ are provided as follows:

$$P(Y = 0|X, w) \approx \frac{1}{1 + e^{w_0 + \sum_i w_i X_i}} \tag{2.1}$$

$$P(Y = 1|X, w) \approx \frac{e^{w_0 + \sum_i w_i X_i}}{1 + e^{w_0 + \sum_i w_i X_i}} \tag{2.2}$$

Following this formulation, the update rule for $w$ are provided below. In the given notation, j denotes a data sample and $\forall j, x_0^j = 1$. For the weights $w$, $w_i^t$ denotes weight i at iteration t. $\lambda$ is the learning rate for the update.

$$w_i^{(t+1)} \leftarrow w_i^{(t)} + \lambda \sum_j x_i^j [y^j - \hat{P}(Y^j = 1|x^j, w^{(t)})] \tag{2.3}$$

To get a clue on what score you should go for, you perform a training with a Logistic Regression classifier provided by a library (eg. `scikit-learn` for Python). Your implementation must be your own work (these packages cannot be used in your submitted work), but to set a baseline for yourself you can follow such an approach. As these implementations make some implicit assumptions in their code, you do not have to exactly match the metrics but your results should be close.

**Note:** For the confusion matrix, take label 1 as the positive class.

**Question 2.1** **[16 pts]** Implement a Logistic Regression Classifier with full-batch gradient ascent algorithm to train your model. Try out different learning rates ($\lambda$) and select the one that works best for you (you may start by trying the values {1, 0.5, 0.05, 0.005}). Initialize your weights from the distribution $\mathcal{N}(\mu = 0, \sigma = 0,01)$. Train your model for 40 epochs (When you traverse through the dataset one time, it is one epoch). After training, report **confusion matrix**, **accuracy** (both class-wise and general), **precision**, **recall**, $F_1, F_2, F_{0.5}$ scores for the best model on the test set. For all of your hyperparameter selections (learning rate in this case), explain how do you decide on the best model and which metrics you use. Then comment on the stability of full-batch gradient descent, if you make several trainings with the same configuration does the model performance change? If you observe such behavior, mention it.

[5] https://www.yapikredi.com.tr/
[6] https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

In the dataset, feature scales are significantly different from each other. To train a model that is not influenced by feature scales, you need to normalize the data. In order to do that, you can apply min-max normalization to scale the features in the range [0,1]. Formulation of this normalization is provided in equation 2.4. While selecting the $X_min$ and $X_max$, use only the samples from the training set since you will apply this normalization to unseen samples, if the classifier would be used in a real-world scenario.

$$\hat{x} = \frac{x - X_{min}}{X_{max} - X_{min}} \tag{2.4}$$

**Question 2.2** **[10 pts]** Modify the classifier you implemented for the previous question such that weights are now updated with mini-batch gradient descent algorithm, with batch size 32. Following the procedure in the previous part, find the optimal learning rate by performing some experiments on it. Then modify your classifier again, such that weight updates are performed by stochastic gradient ascent algorithm. Find the optimal learning rate for this model also. In all of your models, initialize the weights by sampling from $\mathcal{N}(\mu = 0, \sigma = 0,01)$. For the two optimal models you found (one for mini-batch gradient ascent, one for stochastic gradient ascent), report **confusion matrix**, **accuracy** (both class-wise and general), **precision**, **recall**, $F_1, F_2, F_{0.5}$ scores on the test set.

**Question 2.3** **[5 pts]** Using the results on the test set provided to you (now we are treating it as a validation set), select one of the three models that you think it is the best for you. Justify your reasoning, if any of two models have differences that are indistinguishable you can prefer any of them. Mention the metrics that affect your model choice. You will perform your experiments described below with this best model that you selected.

As the final step, you will experiment on different weight initialization techniques. For the optimal model you found in question 2, you will initialize the weights in three different ways. The first method is the one used in the previous question, which samples the initial weights from $\mathcal{N}(\mu = 0, \sigma = 0, 1)$. As the second variant, you will initialize the weights from uniform distribution $\mathcal{U}$. And lastly, you will initialize the weights as zero. Then observe what kind of a change does weight initialization make in training. Report the **confusion matrix**, **accuracy** (both class-wise and general), **precision**, **recall**, $F_1, F_2, F_{0.5}$ scores on the test set (If all three initializations give similar results, reporting only one is enough). Can gradient ascent perform optimization successfully with different weight initializations? Comment on your results. (**Hint**: To observe the behavior change during training, you can look at the plot of change in overall accuracy in succeeding iterations (or epochs), on the training set. Model aims to fit to the training data during training, you can check how good it learns this way.)

**Question 2.4** **[4 pts]** Considering your task, how different the measures $F_1$, $F_2$ and $F_{0.5}$ are? Considering the nature of credit card fraud detection task, to which one would you give the most importance? How are these metrics useful, in addition to precision, recall and accuracy? Explain.

# 3 Support Vector Machines (SVMs) [25 pts]

You will use a wine quality dataset [7] in this question. In the dataset, there are different attributes for wines and target value you want to predict is quality value of each wine which is a value between 3 and 9. The data you are going to use have been already split into train and test subsets. You will use the following files:

- `X_train.csv`
- `X_test.csv`
- `y_train.csv`
- `y_test.csv`

The files whose names start with X are the features where each row corresponds to a data sample. Corresponding target values are stored in y_train.csv and y_test.csv files in the same order. You will use X_train.csv and y_train.csv files during the training and test your model with X_test.csv and y_test.csv files.

In this question, you will train one soft margin and one hard margin SVM classifier on the dataset explained above. You must perform 5-fold cross validation WITHOUT using any libraries but you CAN use scikit-learn package [8] to train your SVM.
In this question, you will train one soft margin and one hard margin SVM classifier on the dataset explained above. You must perform 5-fold cross validation WITHOUT using any libraries but you CAN use libraries or software packages to train your SVM.

**Question 3.1 [9 pts]** In this part, you will train a linear SVM model with soft margin (no kernel). Your model's hyper-parameter is C. Using 5-fold cross validation on your *training set*, find the optimum C value of your model. Look for the best C value within the interval from $[10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}]$ and calculate accuracy on the left-out fold. For each value of C, calculate mean cross validation accuracy by changing the left-out fold each time and plot it in a nice form. Report your optimum C value. Then, run your model on the *test set* with this C value and report test set accuracy along with the confusion matrix. Calculate and report micro and macro averages of precision, recall, negative predictive value (NPV), false positive rate (FPR), false discovery rate (FDR), F1 and F2 scores.

**Question 3.2 [9 pts]** This time, use radial basis function (RBF) kernel to train your hard margin SVM model on the processed data set. Use C as 0.1 for this part. RBF kernel is defined as in Eqn. 3.1

$$K(x, x') = exp\left( - \frac{||x - x'||^2}{2\sigma^2} \right) \tag{3.1}$$

In RBF kernel formula, $\gamma = -\frac{1}{2\sigma^2}$ is a free parameter that can be fine-tuned. This parameter is the inverse of the radius of the influence of samples selected by the model as support vectors. Similar to linear SVM part, train a SVM classifier with RBF kernel using same training and test sets you have used in linear SVM model above. $\gamma$ is your new hyper-parameter that needs to be optimized. Using 5-fold cross validation and calculating mean cross validation accuracy as described in Question 3.1, find and report the best $\gamma$ within the interval from the logarithmic scale $[2^{-4}, 2^{-3}, 2^{-2}, 2^{-1}, 2^{0}, 2^{1}]$. After tuning $\gamma$ on your *training set*, run your model on the *test set* and report your accuracy along with the confusion matrix. Calculate and report micro and macro averages of precision, recall, negative predictive value (NPV), false positive rate (FPR), false discovery rate (FDR), F1 and F2 scores.

**Question 3.3 [7 pts]** Do you see any imbalance in the dataset? If there is an imbalance, which metrics are more reliable in that case? Please comment on this issue.

[7] https://archive.ics.uci.edu/ml/datasets/wine+quality
[8] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html