DATA.ML.100 Introduction to Pattern Recognition and Machine Learning
TAU Computing Sciences
Exercise 3 *Visual classification (CIFAR-10 dataset)*

Be prepared for the exercise sessions (watch the demo lecture). You may ask TAs to help if you cannot make your program to work, but don't expect them to show you how to start from the scratch.

1. **CIFAR-10 – Bayesian classifier (good)** (40 points)

   To reduce the dimension of our data vectors $\boldsymbol{x}_i$ from 3072-dimensions we rescale the images from 32×32 to 1×1. When you do this separately for each color channel (RGB) every CIFAR-10 image is represented with three values that are the mean values of each color channel, i.e. $\boldsymbol{x}_i = \boldsymbol{x}^{(i)}_{3\times 1} = (m^i_R, m^i_G, m^i_B)^T$.

   Write a function *def cifar10_color(X)* that converts the original images in X ($50000 \times 32 \times 32 \times 3$) to Xp ($50000 \times 3$). It might be easier that you first convert the 3072 length vectors to images and then resize them using the *resize()* function in *skimage.transform* package.

   Now, you can compute the mean colors and variances of all training images, e.g., $\mu_{R,aeroplane}$, $\sigma_{R,aeroplane}$, $\mu_{G,aeroplane}$, $\sigma_{B,aeroplane}$, denoted by $(\mu_{R,c}, \mu_{G,c}, \mu_{B,c}, \sigma_{R,c}, \sigma_{G,c}, \sigma_{B,c})$ for each class c.

   The naive Bayes classifier assumes that features m_R, m_G and m_B are independent and therefore a class specific posterior probability can be computed from

$$\begin{aligned} P(class_1|\boldsymbol{x}) &= \frac{P(\boldsymbol{x}|class_1)P(class_1)}{\sum_j P(\boldsymbol{x}|class_j)P(class_j)} \\ &= \frac{\mathcal{N}(m_R;\mu_{R,c_1},\sigma_{R,c_1})\mathcal{N}(m_G;\mu_{G,c_1},\sigma_{G,c_1})\mathcal{N}(m_B;\mu_{B,c_1},\sigma_{B,c_1})P(c_1)}{\sum_j \mathcal{N}(m_R;\mu_{R,c_j},\sigma_{R,c_j})\mathcal{N}(m_G;\mu_{G,c_j},\sigma_{G,c_j})\mathcal{N}(m_B;\mu_{B,c_j},\sigma_{B,c_j})P(c_j)} \end{aligned}$$

   Write a function *def cifar_10_naivebayes_learn(Xp, Y)* that computes the normal distribution parameters *(mu, sigma, p)* for all ten classes (mu and sigma are 10×3 and priors p is 10×1).

   Finally write a function *def cifar10_classifier_naivebayes(x,mu,sigma,p)* that returns the Bayesian optimal class c for the sample x.

   Run your classifier for all CIFAR-10 test samples and report the accuracy. For evaluation you can use the functions implemented for the previous experiments.

2. **CIFAR-10 – Bayesian classifier (better)** (20 points)

   In this experiment we continue the previous Bayesian classifier, but we relax the naive assumption that the red, green and blue channels are independent. Instead of three 1-dimensional Gaussians we assume a single 3-dimensional Gaussian, i.e. *multivariate normal distribution* (Python: numpy.random.multivariate_normal()). Classification becomes

$$P(class_1|\boldsymbol{x}) = \frac{P(\boldsymbol{x}|class_1)P(class_1)}{\sum_j P(\boldsymbol{x}|class_j)P(class_j)} = \frac{\mathcal{N}(\boldsymbol{x};\boldsymbol{\mu}_{c_1},\Sigma_{c_1})P(c_1)}{\sum_j \mathcal{N}(\boldsymbol{\mu}_{c_j},\Sigma_{c_j})P(c_j)}$$

   in which $\boldsymbol{x} = (m_R, m_G, m_B)^T$ is a three-dimensional vector.

   You need to write new functions to replace the naive versions. *def cifar_10_bayes_learn(Xf, Y)* computes the multivariate normal distribution parameters *(mu, sigma, p)* for all ten classes (mu is 10×3, sigma $10 \times 3 \times 3$ and priors p is 10×1 NumPy array). *def cifar10_classifier_bayes(x,mu,sigma,p)* computes probabilities.

   Compute the classification accuracy for the whole test set and compare it to the naive version - which one is better and why?

3. **CIFAR-10 – Bayesian classifier (best)** (20 points)

   Extend *def cifar10_color(X)* to *def cifar10_2x2_color(X)* that resizes the 32×32 image to 2×2 and computes 4 color features for each sub-window. Now the feature vector $\boldsymbol{x}$ length is $2 \times 2 \times 3 = 12$.

   Your previous bayesien learning and classification functions should still work, now only mu is 12×1 and $sigma$ is 12×12.

   Compute the performance for 1×1, 2×2, 4×4, ..., 32×32 images (if only covariance can be computed) and report how the performance improves as the function of the image size (plot a graph). It might happen that after some point the accuracy collapses as we don't anymore have enough data points to robustly esimate the covariance matrix.