ASSIGNMENT

Machine Learning for EDS

2025/2026

Instructions

Carefully read the following instructions before working on this assignment:

1. This assignment is to be made in groups of no more than three students and is mandatory.

2. You can program in any language you like, but support is only offered for Python and R. In the appendix at the end of this assignment you can find some hints and snippets of code that you may use. You can also find the code in `histfuncs.py` and `histfuncs.R` on Canvas.
   Important: please set a random seed at the beginning of your program, such that your results are reproducable. You can do this by calling `numpy.random.seed(seed)` (in Python) or `set.seed(seed)` (in R), where `seed` is set at some integer value, e.g. set `seed=1234`.

3. Hand in your work as a zip file containing your solutions as a pdf and your code. Make sure your code works properly, is well-structured and well-commented. Make sure your names and group number are on the first page of your pdf document.

4. The deadlines are:
   - Part I: Friday, February 27th, 2026 at 23:59 (week 4)
   - Part II: Monday, March 16th, 2026 at 23:59 (week 7)

   Late submissions are not accepted and will receive a 1.

5. The grade will be awarded based on the overall quality of your solutions. Your answers should be correct, clear and complete, but also try to be concise and note that there is no need to restate the questions. Give justifications for your answers and give thorough explanations for your interpretations. Make sure the report looks clean and professional, e.g. make sure the graphs have appropriate axis labels, titles etc. For each part, 100 points lead to a 10.

6. It should go without saying that plagiarism is forbidden. Each group should work on the assigment individually and it is not allowed to copy code or text from other groups. Indications of plagiarism will force me to take action.

7. The use of AI for writing text, coding, data analysis, and reporting is not allowed. The use of generative AI is only allowed to enhance the writing quality, but even then only use it with caution.

ASSIGNMENT PART I: Bayes predictors and Empirical Risk Minimization

Let the feature space be $\mathcal{X} = [0,1]^2$ and let the output space be $\mathcal{Y} = \{0,1\}$. Consider the 0-1 cost function, $c(y', y) = \mathbb{1}_{y' \neq y}$.

Let $(X, Y) \sim P$ be a random example drawn from a joint distribution P. In particlar, we let $X \sim U[0,1]^2$, i.e. $X = (X_1, X_2)$, where X_1 and X_2 are independent random variables drawn uniformly over $[0,1]$. Conditional on $X = x \in [0,1]^2$, the label $Y \in \{0,1\}$ is drawn based on:

$$\mathbb{P}(Y = 1 | X = x) = (1 + \exp(-\alpha(\|x - c\|_2 - r)))^{-1},$$

where $\alpha \in \mathbb{R}$, $c \in \mathbb{R}^2$ and $r > 0$ are fixed parameters. Here, $\|\cdot\|_2$ denotes the Euclidean norm, i.e. $\|x\|_2 = \sqrt{x_1^2 + x_2^2}$ for any $x = (x_1, x_2) \in \mathbb{R}^2$.

Bayes predictor and Bayes risk

1. Answer the questions below and carefully justify your answers

   (a) **(4 pts)** Using the general expression of Bayes predictors for 0–1 loss, show that if $\alpha > 0$, the predictor $f^* : [0,1]^2 \mapsto \{0,1\}$, defined as

   $$f^*(x) = \mathbb{1}_{\|x-c\|_2 \geq r}\,,$$

   is a Bayes predictor.

   (b) **(3 pts)** Derive the expression of a Bayes predictor in case $\alpha < 0$.

   (c) **(3 pts)** What can you conclude about the geometric shape of the decision boundary of the Bayes predictor, i.e. the set of points x at which predicting 0 and 1 yields the same conditional expected cost?

2. Consider the following parameter configurations:

   (i) $c = (0.5, 0)$, $r = 0.5$, $\alpha = 5$,

   (ii) $c = (0.5, 0)$, $r = 0.5$, $\alpha = 10$,

   (iii) $c = (0.5, 0)$, $r = 0.75$, $\alpha = 5$,

   (iv) $c = (0.5, 0)$, $r = 0.75$, $\alpha = 10$.

   For each configuration, the file `assignment_train_sample.csv` (available on Canvas) contains $n = 300$ samples (X, Y) drawn from P.

   Note: The first column of the csv file contains the index (from 1 to 300), the next three columns contain (X_1, X_2, Y) sampled from configuration (i), the next three columns contain (X_1, X_2, Y) sampled from configuration (ii), etc.

(a) **(15 pts)** For each configuration, plot the decision boundary of the Bayes classifier given in question 1(a) in the $[0,1] \times [0,1]$ plane. Shade the regions corresponding to predictions 1 and 0 in red and blue, respectively, and also plot the sampled points using red for $Y = 1$ and blue for $Y = 0$.

Hint: See appendix A.1 or B.1 for a function that you can use for plotting decision boundaries.

(b) **(5 pts)** Compare the resulting figures and give an intuition for the effects of α and r on the decision boundary.

3. There is no closed form expression for Bayes risk in this setting, but we can *approximate* it using Monte Carlo integration. More specifically, Bayes risk $\mathcal{R}_P^* = \mathbb{E}[\mathbb{1}_{f^*(X) \neq Y}]$ can be approximated by the empirical risk

$$\widehat{\mathcal{R}}_{\bar{n}}(f^*) = \frac{1}{\bar{n}} \sum_{i=1}^{\bar{n}} \mathbb{1}_{f^*(X_i) \neq Y_i} ,$$

where $((X_1, Y_1), \ldots, (X_{\bar{n}}, Y_{\bar{n}}))$ is a sample of iid examples $(X_i, Y_i) \sim P$. By the law of large numbers for iid sequences, this quantity will converge to $\mathcal{R}_P^*$ almost surely as $\bar{n} \to \infty$.

(a) **(10 pts)** For each of the four configurations of P in question 2, give an approximation of Bayes risk using this suggested approach, using simulated samples of length $\bar{n} = 1,000,000$. Round your answer to 5 decimals.

(b) **(5 pts)** Provide a careful intuition for the observed effect of α on Bayes risk.

Empirical Risk Minimization: Histogram classifiers

We now apply Empirical Risk Minimization (ERM) to the histogram classifier model introduced in week 3. Let $\mathcal{A}_r$ denote the partition of $\mathcal{X} = [0,1]^2$ into r^2 equally sized squares for $r \in \mathbb{N}_+$:

$$\mathcal{A}_r = \left\{ \, [r^{-1}k_1, r^{-1}(k_1+1)) \times [r^{-1}k_2, r^{-1}(k_2+1)) \, \right\}_{(k_1,k_2) \in \{0,1,\ldots,r-1\}^2} .$$

Write $\mathcal{A}_r = \{A_1, \ldots, A_m\}$ with $m = r^2$. A histogram classifier on this partition is a step-function:

$$f_{\boldsymbol{a}}^{\mathcal{A}_r}(x) = \sum_{i=1}^{m} a_i \mathbb{1}_{x \in A_i} ,$$

where $\boldsymbol{a} = (a_1, \ldots, a_m) \in \{0,1\}^m$. For any integer r, we let S_r denote the model of step functions on the partition $\mathcal{A}_r$:

$$S_r = \left\{ f_{\boldsymbol{a}}^{\mathcal{A}_r} : \boldsymbol{a} \in \{0,1\}^{r^2} \right\} .$$

4. For each $r \in \{2, 4, 8, 16\}$, construct the ERM predictor $\hat{f}_{S_r}(D_n)$ on S_r for each sample D_n in the file `assignment_train_sample.csv`, and provide answers for the following questions.

Hint: See appendices A.2 and B.2 for guidance on constructing the ERM predictor for the model of histogram classifiers.

(a) **(5 pts)** For each r and each sample D_n, report the empirical risk on the training set. Round your answers to three decimals.

(b) **(10 pts)** For the sample corresponding to configuration (ii) only, produce four plots (one per r) showing the decision boundaries/regions of each obtained predictor alongside the training data. Analyse how the decision regions change across these plots. Suggest an improvement to the ERM predictor by dealing with empty cells of the partition in a different way.

(c) **(5 pts)** State whether the empirical risk increases or decreases with r, and carefully explain the intuition behind this.

Empirical Convexified Risk Minimization: linear pseudo-classifiers

Next, we consider Empirical Convexified Risk Minimization (ECRM) for the model of affine pseudo-classifiers. For $b \in \mathbb{R}$ and $\boldsymbol{w} = (w_1, w_2) \in \mathbb{R}^2$, define the pseudo-classifier:

$$h_{\boldsymbol{w},b}(x) = b + w_1 x_1 + w_2 x_2 \,,$$

for any $x = (x_1, x_2) \in [0,1]^2$. Define the model S_{lin} as the set of linear pseudo-classifiers:

$$S_{\text{lin}} := \{h_{\boldsymbol{w},b} : \boldsymbol{w} \in \mathbb{R}^2, b \in \mathbb{R}\} \,.$$

Transform the labels Y to be such that $Y \in \{-1, 1\}$ instead of $Y \in \{0, 1\}$, by transforming $2Y - 1$. Use the squared error surrogate loss function: $\Phi(-yh(x)) := (1 - yh(x))^2$ for $y \in \{-1, 1\}$ and $x \in [0,1]^2$. The plug-in classifier f_h associated to $h \in S_{\text{lin}}$ is:

$$f_h(x) = \begin{cases} 1 & h(x) \geq 0 \,, \\ -1 & h(x) < 0 \,. \end{cases}$$

5. **(5 pts)** Show that for $y \in \{-1, 1\}$, $(1 - yh(x))^2 = (y - h(x))^2$. Use this identity to explain how the ECRM on S_{lin} can be obtained by ordinary least squares regression.

6. Using the result of question 5, construct the ECRM pseudo-classifiers for the model S_{lin} using the squared error surrogate loss function for the four samples from `assignment_train_sample.csv`, and answer the questions below.

   (a) **(10 pts)** Construct the plug-in classifiers corresponding to the ECRM pseudo-classifiers and plot their corresponding decision boundaries alongside the data points (leading to four figures).

   (b) **(5 pts)** Report the empirical risk based on the 0-1 cost function of the obtained plug-in classifiers for each training set. Round your answers to three decimals.

Consider the following more flexible pseudo-classifier for any $\boldsymbol{m} = (m_1, m_2, m_3, m_4, m_5) \in \mathbb{R}^5$ and $b \in \mathbb{R}$:

$$h_{\boldsymbol{m},b}(x) = b + m_1 x_1 + m_2 x_2 + m_3 x_1 x_2 + m_4 x_1^2 + m_5 x_2^2 \,,$$

where $x = (x_1, x_2) \in [0,1]^2$. Let

$$S_{\text{nonlin}} := \{h_{\boldsymbol{m},b} : \boldsymbol{m} \in \mathbb{R}^5, b \in \mathbb{R}\} \,.$$

The pseudo-classifiers in this model are non-linear in x_1 and x_2, but are linear in the parameters $\boldsymbol{m}$ and b. Hence, you can obtain the ECRM using the same approach as above, using the extended feature vector $(x_1, x_2, x_1 x_2, x_1^2, x_2^2)$.

7. (a) **(10 pts)** Repeat question 6, but then for S_{nonlin}. Again, plot the decision regions and report the empirical risk based on 0-1 loss rounded to three decimals.

   (b) **(5 pts)** For each of the four parameter configurations, write down whether empirical risk increases or decreases when moving from S_{lin} to S_{nonlin}, and carefully give the intuitions behind this.

ASSIGNMENT PART II: Evaluating estimation and approximation error, and comparing to learning guarantees + support vector machines

Estimation and approximation error of ERM/ECRM predictors

This is a continuation of Assignment Part I. Please extend the your code from part I. We want to obtain the estimation and approximation error for the ERM and ECRM predictors that we trained in part I.

By the law of large numbers, the risk $\mathcal{R}_P(\hat{f}_S)$ of an ERM predictor $\hat{f}_S$ for some model S can be approximated by evaluating the empirical risk of $\hat{f}_S$ on a very large sample $D_{\bar{n}}$ that is independent of the training sample. In other words, we can use the following approximation for large $\bar{n}$:

$$\widehat{\mathcal{R}}_{\bar{n}}(\hat{f}_S) = \frac{1}{\bar{n}} \sum_{i=1}^{\bar{n}} \mathbb{1}_{\hat{f}_S(X_i) \neq Y_i} ,$$

which converges almost surely to the true generalization error $\mathcal{R}_P(\hat{f}_S) = \mathbb{E}[\mathbb{1}_{\hat{f}_S(X) \neq Y} | D_n]$ as $\bar{n} \to \infty$. Furthermore, we can approximate $\inf_{f \in S} \mathcal{R}_P(f)$ by $\inf_{f \in S} \widehat{\mathcal{R}}_{\bar{n}}(f)$ for a large sample size $\bar{n}$, i.e. by carrying out empirical risk minimization on some large sample and then evaluating the resulting empirical risk on this large sample.

1. For this question, only look at **configuration (ii) of question I.2** and consider the model of histogram classifiers S_r defined in part I of the assignment, and in particular look at the ERM predictors $\hat{f}_{S_r}$ for $r = 2, 4, 8, 16$ that you obtained in question I.4 based on the sample given in `assignment_train_sample.csv` corresponding to configuration (ii).

   (a) **(5 pts)** For each r, approximate $\mathcal{R}_P(\hat{f}_{S_r})$ using the approach suggested above. Use the same sample of length $\bar{n}$ simulated under configuration (ii) as in question I.3 for this. Round your answers to 5 decimals.

   (b) **(5 pts)** For each r, approximate $\inf_{f \in S_r} \mathcal{R}_P(f)$ using the approach suggested above based on the same sample of length $\bar{n}$ as in question I.3. In particular, for each r, construct the ERM predictor on this large sample and evaluate the empirical risk of this predictor on this sample. Also plot the decision boundaries of the obtained predictor for each r (do not plot the sample points). Round your answers to 5 decimals.

   (c) **(5 pts)** Combine the results of (a) and (b) with your approximated Bayes risk of question I.3 to calculate the *approximate* approximation and estimation error for each r. Also report the corresponding excess risks.

   (d) **(8 pts)** Analyse the *approximated* excess risk, estimation error and approximation error for the different values of r. Carefully explain the intuition behind the observed effect of r on these quantities. Based on these results, which model has the optimal performance in terms of generalisation error? How do you expect the excess risk, estimation error and approximation error would change if you would have used a larger sample size n in question I.4 to construct the ERM predictor? Explain your answers.

2. Answer the following questions:

   (a) **(2 pts)** For fixed r, is the model of histogram classifiers S_r a finite model? If so, what is Card S_r?

   (b) **(3 pts)** Select an appropriate learning guarantee from the lecture slides that can be applied in this setting. According to this learning guarantee, what is the upper bound of the estimation error of the ERM predictor of the model S_r for $r = 2, 4, 8, 16$ based on samples of length $n = 300$ in at least 90% of the cases? Show your calculations.

   (c) **(2 pts)** Compare the upper bounds obtained in (b) to the estimation errors you approximated in question 1. Is it surprising that the estimation errors are far from the obtained upper bounds?

3. For the following questions consider the model of nonlinear classifiers $S^f_{\text{nonlin}} := \{f_h : h \in S_{\text{nonlin}}\}$, where f_h and S_{nonlin} are defined in part I of the assignment, and consider the ECRM predictors that you obtained in question I.7 based on the sample given in `assignment_train_sample.csv`.

   (a) **(5 pts)** For each configuration, approximate $\mathcal{R}_P(\hat{f}_{h_{S_{\text{nonlin}}}})$ using the same approach as in 1(a), where $\hat{f}_{h_{S_{\text{nonlin}}}}$ denotes the predictor obtained in question I.7. Use the same samples of length $\bar{n}$ as in question I.3. Round your answers to 5 decimals.

   (b) **(5 pts)** Without doing any calculations, give the approximation error of the model S^f_{nonlin}. Carefully explain your answer.

   (c) **(5 pts)** Combine the results of (a) with your approximated Bayes risk of question I.3 to calculate the *approximate* estimation error and excess risk for each configuration.

   (d) **(5 pts)** For configuration (ii), compare the obtained excess risk, estimation error and approximation error to thoses you found in question 1(c). Which learning rule has a better performance here, the ERM for histogram classifiers or the ECRM based on S_{nonlin}? Explain whether this is surprising or not.

4. Again consider S^f_{nonlin} and also S^f_{lin}, which is the model of plug-in classifiers associated to the models of pseudo-classifiers S_{lin}, i.e. $S^f_{\text{lin}} := \{f_h : h \in S_{\text{lin}}\}$.

   (a) **(5 pts)** Are the models S^f_{lin} and S^f_{nonlin} finite models? If yes, give the number of predictors in each of these models. If not, give the VC-dimension or an upper bound of the VC-dimension of these models. You are allowed to directly use the results given in the lecture slides.

   (b) **(5 pts)** Select an appropriate learning guarantee from the lecture slides that can be applied in this setting. According to this learning guarantee, what is the upper bound of the estimation error of the ERM predictor of the models S^f_{lin} and S^f_{nonlin} based on samples of length $\boldsymbol{n = 3000}$ (so not 300) in at least 90% of the cases? Show your calculations.

   (c) **(5 pts)** Are upper bounds found in question (b) applicable to the estimation errors of the plug-in classifiers associated to ECRM pseudo-classifiers for the models S_{lin} and S_{nonlin} for the squared error surrogate loss function? Carefully explain your answer.

Support vector machines

For this part of the assignment, you will consider a real data set. In particular, you will use a banknote authentication data set[1]. On Canvas, you can find the csv files:

- `data_banknote_authentication_train.csv`
- `data_banknote_authentication_validation.csv`
- `data_banknote_authentication_test.csv`

which contain the training data, validation data and test data.

The goal is to predict whether a given banknote is genuine or forged given certain features. The features have been extracted from photographs of banknotes using so-called Wavelet transformations. There are four features (the first four columns of the csv files):

- `variance`: the variance of the Wavelet transformed image
- `skewness`: skewness of the Wavelet transformed image
- `kurtosis`: kurtosis of the Wavelet transformed image
- `entropy`: entropy of the image

For this assignment, it is not necessary to know how these features have been precisely constructed, you can simply take them as given. The class (last column of the csv files) is 0 for genuine banknotes and 1 for forged banknotes. **Note:** the features have already been rescaled to appropriate sizes, so you do not need to do normalizations yourself.

For this part of the assignment, you will apply support vector machines (SVM) with different kernels. It is recommended to use `libsvm`, which is a C library. If you use Python, you can access this library via the `svm.SVC` class of the scikit-learn package (`sklearn`), see this documentation. If you use R you can access it via the `svm` class of the `e1071` library, see this documentation.

5. **(5 pts)** For each combination of two features, train a support vector machine (SVM) with linear kernel (i.e. $K(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^\top \boldsymbol{x}'$) on the training set where you use the default value $C = 1$. Give the accuracies on the validation set for each pair of features. Identify the feature pairs with the highest and the lowest validation accuracy. For both pairs, plot the SVM decision regions alongside the training data.

   Note: The accuracy of a predictor on a sample is the number of correct predictions divided by the total number of predictions. This is equivalent to 1 minus the empirical risk based on the 0-1 loss on that sample.

[1] Retrieved from UCI Machine Learning Repository

6. Consider the feature pair that achieved the lowest validation accuracy in the previous question.

   (a) **(5 pts)** Train an SVM with RBF kernel on the training set. Perform a grid search over the hyperparameters

   $$C \in \{0.1, 1, 10, 100, 1000\}, \quad \text{and} \quad \gamma \in \{0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000\}.$$

   Evaluate each model on the validation set and select the combination of C and γ that yields the highest validation accuracy. Report the selected values of C and γ, as well as the corresponding validation accuracy (do not report all the other accuracies). Is the accuracy higher than for the linear SVM based on these two features?

   (b) **(8 pts)** Fix C to the value selected in (a). Produce one figure showing training and validation accuracy as functions of γ (use a $\log_{10}$ scale on the x-axis), and separate figures showing the SVM decision boundaries for each value of γ (so in total 9 figures). Analyse the resulting figures and discuss whether there is evidence of overfitting for certain values of γ.

   (c) **(7 pts)** Fix C to the value selected in (a). Plot the number of support vectors of the fitted SVMs from part (a) as a function of γ. Explain how the parameter γ affects the number of support vectors, and the intuition behind this.

   Hint: In Python, the number of support vectors for each class is stored in the `n_support_` attribute of your `sklearn.svm.SVC` object, so summing these gives you the total number of support vectors. In R, use `index` component of your `svm` object, which stores the indices of the support vectors in the input data. The number of indices in `index` is therefore the number of support vectors.

7. Now consider all four features jointly.

   (a) **(6 pts)** Train SVM classifiers with linear, RBF, and polynomial kernels. For each kernel, select hyperparameters based on validation set accuracy using the following grids:

   - Linear kernel: $C \in \{0.01, 0.1, 1, 10, 100, 1000\}$.
   - RBF kernel: same grid for C and γ as in question 6.
   - Polynomial kernel: $C \in \{0.1, 1, 10, 100, 1000\}$ and $d \in \{1, 2, 3, 4, 5, 6\}$, set `coef0` $= 1$.

   All unspecified parameters are left at their default values. If multiple hyperparameter settings achieve the same highest validation accuracy, select the one with the lowest C; if still ambiguous, choose the smallest γ or d. Explain briefly why this selection rule is used. Report the selected hyperparameters and validation accuracies.

   (b) **(4 pts)** For each kernel retrain the SVM with the selected hyperparameters on the merged training and validation set. Compare the test set accuracies of the three resulting SVMs. Which kernel achieves the best test set performance? Do the validation and test results agree? If not, briefly comment on possible reasons.

A Appendix - Coding hints: Python

The Python functions listed below can also be found in the file `histfuncs.py` on Canvas.

A.1 Plotting decision boundaries

To plot the decision boundaries, you can for instance use the `contour` or `contourf` function of `matplotlib.pyplot`. There are also other ways of doing this, for instance `sklearn.inspection.DecisionBoundaryDisplay`.

The function below for example uses `contourf`. Notice that the function `predict_fun` must be a function which takes as input a single feature vector (x_1, x_2) and outputs a prediction (here either 0 or 1). Hence, if you want to use a function that also depends on other parameters, you should use a so-called lambda function. E.g. to use the `hist_predict` function of the next section, you can use `predict_func = lambda x_vec: hist_predict(x_vec, pred_mat, edges)`.

```
import numpy as np
import matplotlib.pyplot as plt

def plot_dec_bound(h, predict_func, x1min, x1max, x2min, x2max, X, y, title, label_1, label_2):
  """
  Purpose:
      Plot decision boundary on a [x1min,x1max] x [x2min,x2max] grid and plot a sample of points.

  Inputs:
      h             double, size of step in the grid that you use for determining decision
                            boundary (choose e.g. 0.001)
      predict_func  function, a function that takes as input an array with two entries and
                              outputs a real value
      x1min         double, lower bound to use for x1
      x1max         double, upper bound to use for x1
      x2min         double, lower bound to use for x2
      x2max         double, upper bound to use for x2
      X             matrix, n x 2 matrix where each row is a feature vector of training sample
      y             vector, n-vector containing the labels corresponding to the rows of X
      title         string, title to put above plot and as file name for the saved plot
      label_1       string, label to use for x-axis
      label_2       string, label to use for y-axis

  Return value:
      -
  """

  # create a mesh to plot in
  xx1, xx2 = np.meshgrid(np.arange(x1min+h, x1max-h, h),
                         np.arange(x2min+h, x2max-h, h))

  Z = np.zeros((xx1.shape[1],xx2.shape[0]))
  i=0
  j=0

  for x1 in xx1[0,:]:
     j=0
     for x2 in xx2[:,0]:
```

```
            Z[i,j] = predict_func([x1,x2])
            j=j+1
        i=i+1

    # Put the result into a color plot. cmap determines the color scheme, alpha determines
    # the opacity
    plt.contourf(xx1, xx2, Z.T, cmap=plt.cm.coolwarm, alpha=0.5)
    plt.contour(xx1, xx2, Z.T, [0.5], colors='black')
    plt.xlabel(label_1)
    plt.ylabel(label_2)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    # Also plot the training points
    plt.scatter(X[:,0], X[:,1], c=y, cmap=plt.cm.coolwarm)

    plt.title(title)
    plt.savefig(title,dpi=600,bbox_inches='tight')
    plt.show()
```

A.2 Implementing the histogram classifier

We proved in the lectures of week 3 that the empirical risk minimizer of the set of step-functions on a partition, is the predictor which predicts the majority class of the sample within each cell of the partition. As in the slides, let's predict 0 whenever there is a draw or a cell is empty.

The ERM predictor on S_r can be constructed straightforwardly using Python. It is convenient to use the function `numpy.histogramdd` (see the documentation here).

Below is an implementation of a 'fit' and 'predict' function in Python which you may use (see also the file `histfuncs.py` on Canvas). The 'fit' function, determines which label should be assigned to each cell based on a sample using majority voting. Its outputs are `pred_mat` and `edges`, which respectively are a matrix containing the labels of each cell and a matrix containing the boundaries of each cell. The 'predict' function, predicts the label corresponding to a feature vector by checking to which cell of the partition it belongs and retrieving the corresponding predicted value.

```
import numpy as np

def hist_fit(X, y, r):
  """
    Purpose:
        Fit a histogram in [0,1]^2 with r*r bins

    Inputs:
        X             matrix, n x 2 matrix with sample of feature vectors
        y             vector, n-dimensional vector containing labels corresponding to X
        r             int, number of bins per feature

    Return value:
        pred_mat      matrix, r x r matrix of predicted label per cell of histogram partition
        edges         matrix, 2 x (r + 1) matrix containing boundaries of cells of partition
```

```
  """
  hist1, edges = np.histogramdd(X[y == 1,:], bins=r, range=([0,1],[0,1]))
  hist0, _ = np.histogramdd(X[y == 0,:], bins=r, range=([0,1],[0,1]))
  pred_mat = (hist1 > hist0)

  return pred_mat, edges

def hist_predict(x_vec, pred_mat, edges):
  """
    Purpose:
        Evaluate ERM histogram classifier for a single feature vector

    Inputs:
        x_vec         vector, 2-dimensional vector (x_1,x_2) for which histogram classifier
                              is to be evaluated
        pred_mat      matrix, r x r matrix of predicted label per cell of histogram partition
        edges         matrix, 2 x (r + 1) matrix containing boundaries of cells of partition

    Return value:
        prediction    int, predicted label, either 0 or 1
  """
  row_index = np.where((edges[0][:-1] <= x_vec[0]) & (x_vec[0] < edges[0][1:]))
  col_index = np.where((edges[1][:-1]<= x_vec[1]) & (x_vec[1] < edges[1][1:]))

  prediction = pred_mat[row_index,col_index]

  return prediction

def hist_predict_mult(X, pred_mat, edges):
  """
    Purpose:
        Evaluate ERM histogram classifier for n feature vectors

    Inputs:
        X             matrix, n x 2 matrix with a feature vector (x_1,x_2) in each row
        pred_mat      matrix, r x r matrix of predicted label per cell of histogram partition
        edges         matrix, 2 x (r + 1) matrix containing boundaries of cells of partition

    Return value:
        predictions   vector, n-dimensional vector containing the predictions of a fitted
                              histogram classifier for each feature vector in X
  """
  predictions = np.zeros(X.shape[0])
  for i in range(X.shape[0]):
    predictions[i] = hist_predict(X[i,:], pred_mat, edges)

  return predictions
```

B Appendix - Coding hints: R

The R functions listed below can also be found in the file `histfuncs.R` on Canvas.

B.1 Plotting decision boundaries

To plot decision boundaries corresponding to a particular predictor, you can use the function below. Notice that the argument `pred_fun` must be a function which takes as input a single feature vector (x_1, x_2). Hence, if you want to use a function that also depends on other parameters, you should use a so-called lambda function. E.g. to use the `hist_predict` function of the next section, you have to use `pred_func <- (function(x_vec) hist_predict(x_vec, pred_mat, edges))`.

```
plot_dec_bound <- function(h, pred_func, x1min, x1max, x2min, x2max, X, y, title, label_1,
                  label_2){
  #' Plot decision boundary
  #'
  #' @description Plot decision boundary on a [x1min,x1max] x [x2min,x2max] grid
  #' and plot some points.
  #' @param h size of step in the grid that you use for determining decision boundary
  #'          (choose e.g. 0.001)
  #' @param predict_func a function that takes as input an array with two entries and
  #'                     outputs a prediction (here either 0 or 1)
  #' @param x1min double, lower bound to use for x1
  #' @param x1max double, upper bound to use for x1
  #' @param x2min double, lower bound to use for x2
  #' @param x2max double, upper bound to use for x2
  #' @param X nx2 dimensional matrix where each row is a feature vector
  #' @param y n-vector containing the labels of the corresponding rows of X
  #' @param title string containing title for the plot
  #' @param label_1 string containing x axis label
  #' @param label_2 string containing y axis label
  #' @return -
  #'
  grid1 <- seq(x1min+h, x1max-h, by=h)
  grid2 <- seq(x2min+h, x2max-h, by=h)
  grid1_len <- length(grid1)
  grid2_len <- length(grid2)

  Y_preds <- matrix(data=NA, nrow=grid1_len, ncol=grid2_len)

  for (i in 1:grid1_len){
    for (j in 1:grid2_len){
```

```
      xx_vec <- matrix(c(grid1[i],grid2[j]), 1, 2)
      Y_preds[i,j] <- pred_func(xx_vec)
    }
  }

  cp <- colorRampPalette(c("cornflowerblue", "coral1"))
  filled.contour(grid1, grid2, Y_preds, xlab = label_1, ylab = label_2,
                 main = title, axes = TRUE, nlevels = 2, color.palette = cp, plot.axes = {
                   contour(Y_preds, add = TRUE)
                   points(X, col = ifelse(y == 1, "red", "blue"), pch = 20)
                 }
  )
}
```

B.2 Implementing the histogram classifier in R

You can use the package `mvmesh` (see the documentation here). First make a `SolidRectangle` object, which defines the histogram partition that you use. Then count the number of points in each cell of the histogram partition using `histSimplex`. This function is quite slow, so it might work better to code a 'histogram fit' function from scratch (i.e. simply define a grid for both x_1 and x_2 and for each class count the number of points in each cell of the corresponding partition).

Below is an implementation of a 'fit' and 'predict' function in R which you may use.

```
library(mvmesh)

hist_fit <- function(y, X, r){
  #' Fit histogram classifier
  #'
  #' Use ERM to find optimal step-function on histogram partition
  #' @param y : n dimensional vector. Contains outputs of each example in sample
  #' @param X : n x 2 matrix. Contains 2-dimensional feature vector for each
  #'           example in sample
  #' @param r : integer. Number of bins per feature to consider.
  #' @return Returns a list with the prediction matrix ('pred_mat') and the
  #'          edges of histogram ('edges')
  #'

  X1 <- X[y==1,]
  X0 <- X[y==0,]

  H <- SolidRectangle(a=c(0,0), b=c(1,1), breaks = r)
```

```
  histY1 <- histSimplex( X1, H$S, plot.type="none")
  histY0 <- histSimplex( X0, H$S, plot.type="none")

  pred_vec <- (as.integer(histY1$counts > histY0$counts))
  pred_mat = matrix(pred_vec, nrow=r,ncol=r,byrow=TRUE)

  out <- list('pred_mat' = pred_mat,'edges' = H$breaks[[1]])
  return(out)
}

hist_predict <- function(x_vec, pred_mat, edges){
  #' Evaluate prediction of fitted histogram classifier
  #'
  #' Given a histogram partition and corresponding label, predict
  #' output value corresponding to single input vector X
  #' @param x_vec : 2 dimensional vector. Contains single feature vector
  #' @param pred_mat : r x r matrix. Contains predicted labels corresponding to
  #'                   each cell of histogram partition
  #' @param edges : (r+1) dimensional vector. Edges of histogram partition
  #' @return Returns the predicted output value according to fitted predictor
  #'

  ind_col <- which((x_vec[1] > head(edges, -1)) & (x_vec[1] < edges[-1]))
  ind_row <- which((x_vec[2] > head(edges, -1)) & (x_vec[2] < edges[-1]))

  return(pred_mat[ind_row,ind_col])
}
```

Note: You might get an error message if you try to load the `mvmesh` package. This has to do with a problem with one of the dependencies of this package (namely the `rgl` package). First calling `options(rgl.useNULL = TRUE)` should solve te problem.