ASSIGNMENT

Machine Learning for EDS

2023/2024

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: Thursday, February 29th, 2024 at 23:59 (week 4)
   - Part II: Monday, March 18th, 2024 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. 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, $c(y', y) = \mathbb{1}_{y' \neq y}$.

Consider examples $(X, Y) \sim P$, where the joint distribution P can be described as follows: $Y \sim \text{Bernoulli}(p)$, i.e. the random variable Y is equal to 1 with probability p and is equal to 0 with probability $1-p$. Conditional on $Y = i$, the elements of the random vector $X = (X_1, X_2)$ are two independent Beta-distributed random variables with parameters $\alpha_i > 0$ and $\beta_i > 0$, in other words:

$$X_j | Y = i \sim \text{Beta}(\alpha_i, \beta_i) \quad \text{for } i \in \{0,1\} \text{ and } j \in \{1,2\}.$$

The Beta-distribution is a continuous probability distribution with support $[0,1]$. For any $z \in [0,1]$, the probability density function of a Beta-distributed random variable with parameters $\alpha > 0$ and $\beta > 0$ is given by:

$$p(z) = \frac{1}{B(\alpha, \beta)} z^{\alpha-1}(1-z)^{\beta-1},$$

where $B(\alpha, \beta) = \Gamma(\alpha)\Gamma(\beta)/\Gamma(\alpha+\beta)$ is the Beta function (in Python `scipy.special.beta` and in R simply `beta`).

Bayes predictor and Bayes risk

1. **(10 pts)** Show that a Bayes predictor in this setting is given by $f^*(X) = \mathbb{1}_{A(X)>0}$, where the function $A : \mathcal{X} \to \mathbb{R}$ is defined as follows for any $x = (x_1, x_2) \in \mathcal{X}$:

$$A(x) = \log\left(B(\alpha_0, \beta_0)^2 / B(\alpha_1, \beta_1)^2\right) + \log(p) - \log(1-p) + (\alpha_1 - \alpha_0)\big[\log(x_1) + \log(x_2)\big] + (\beta_1 - \beta_0)\big[\log(1-x_1) + \log(1-x_2)\big].$$

   You may directly use the general expression of Bayes predictors in the binary classification framework with 0-1 cost derived in week 2.

2. **(20 pts)** Plot the decision boundary of the Bayes classifier given in question 1 in the $[0,1] \times [0,1]$ plane for the following parameter configurations:

   (i) $p = 0.5$, $\alpha_0 = 0.6$, $\beta_0 = 0.6$, $\alpha_1 = 3$ and $\beta_1 = 4$

   (ii) $p = 0.8$, $\alpha_0 = 0.6$, $\beta_0 = 0.6$, $\alpha_1 = 3$ and $\beta_1 = 4$

   (iii) $p = 0.5$, $\alpha_0 = 2$, $\beta_0 = 1$, $\alpha_1 = 1$, and $\beta_1 = 2$

   (iv) $p = 0.8$, $\alpha_0 = 2$, $\beta_0 = 1$, $\alpha_1 = 1$, and $\beta_1 = 2$

In the file `assignment_train_sample.csv` on Canvas, you can find $n = 250$ (X, Y) pairs sampled from P for each configuration. For each configuration, plot the corresponding points in the same axis as the decision boundary plots with a different color for each of the two classes. Briefly comment on the shape of the plotted decision regions. In particular, give an intuition for the effect of p on the decision boundary by comparing (i) vs. (ii), and (iii) vs. (iv).

Note: The first column of the csv file contains the index (from 1 to 250), 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.

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

There is no closed form expression for Bayes risk in this setting, but we can *approximate* Bayes risk 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

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

for a large choice of $\bar{n}$, 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 namely converge to $\mathcal{R}^*_P$ almost surely as $\bar{n} \to \infty$.

3. **(15 pts)** For each of the four distributions P of question 2, give an approximation of Bayes risk using the approach suggested above, based on simulated samples of length $\bar{n} = 1,000,000$. Round your answer to 5 decimals. Does Bayes risk tend to increase or decrease as p moves away from 0.5? Give an intuition for this.

   Hint: See appendix A.2 or B.2 for a function that you can use for simulating from P.

Empirical Risk Minimization: Histogram classifiers

Turn to Empirical Risk Minimization (ERM) for the model of histogram classifiers introduced in week 3. More specifically, consider the partition of $\mathcal{X} = [0, 1]^2$ into r^2 identically sized squares for some integer r:

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

So r is the number of bins per feature and hence r^{-1} is the length of each of the sides of the squares. In the slides of week 3, you can find a visual representation of $\mathcal{A}_r$ for $r = 5$, $r = 10$ and $r = 20$.

Histogram classifiers are step-functions over such partitions $\mathcal{A}_r$. Recall that step-functions $f^{\mathcal{A}}_{\boldsymbol{a}} : \mathcal{X} \to \mathcal{Y}$ for any given partition $\mathcal{A} = \{A_1, \ldots, A_m\}$ of $\mathcal{X}$ are defined as follows for any $x \in \mathcal{X}$:

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

where in this case $\boldsymbol{a} = (a_1, \ldots, a_m) \in \{0, 1\}^m$. For any integer r, we then define the model S_r of histogram classifiers with r bins per feature as the set of step-functions on the partition $\mathcal{A}_r$:

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

Recall from the slides of week 3 that we have a closed form expression for the ERM predictor $\hat{f}_{S_r}(D_n)$ for this type of model, where D_n is a sample of length n.

4. **(10 pts)** Construct the ERM predictor on S_r for each sample D_n of length $n = 250$ from the file `assignment_train_sample.csv`, (denote this ERM by $\hat{f}_{S_r}(D_n)$) where you consider $r = 2, 4, 8, 16$. For configuration (i), plot the decision boundaries of each obtained predictor alongside the data points (so you should have four figures).

   Hint: See appendices A.3 and B.3 of this assignment for guidelines for obtaining the ERM predictor.

5. **(10 pts)** Report the empirical risk on the training sample for each ERM predictor from the previous question (so report four times four empirical risks). Round your answers to 3 decimals. Write down whether empirical risk increases or decreases with r, and explain the intuition behind this pattern. Could this pattern in theory be reversed? Explain your answer.

Empirical Convexified Risk Minimization: linear pseudo-classifiers

The model of histogram classifiers considered above is an exception, because for most models it is not possible to derive the ERM predictor in closed form. In fact, even obtaining the ERM by numerically minimizing the empirical risk is usually problematic in the classification setting, because the 0-1 cost function is non-differentiable. As we saw in the lectures of week 3, a solution is to use Empirical Convexified Risk Minimization (ECRM). In this part of the assignment, we consider ECRM for the model of linear (or technically affine) pseudo-classifiers. For any $b \in \mathbb{R}$ and $\boldsymbol{w} = (w_1, w_2) \in \mathbb{R}^2$, define the following linear 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:

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

For this part of the assignment, transform the target value Y to take values in $\{-1, 1\}$ instead of $\{0, 1\}$, which can be done by multiplying the labels by 2 and subtracting 1 (so $2Y - 1$). Consider the squared error surrogate loss function: $\Phi(-yh(x)) := (1 - yh(x))^2$ for $y \in \{-1, 1\}$ and $x \in [0,1]^2$. A plug-in classifier f_h can be associated to each pseudo-classifier $h \in S_{\text{lin}}$ as follows:

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

6. **(5 pts)** Show that for $y \in \{-1, 1\}$ the squared error surrogate loss function $\Phi(-yh(x)) := (1 - yh(x))^2$, is equivalent to $\Phi^*(-yh(x)) := (y - h(x))^2$. Use this fact to explain how the ECRM on S_{lin} can be obtained using ordinary least squares regression.

7. **(15 pts)** Construct the ECRM pseudo-classifiers for the model S_{lin} using the squared error surrogate loss function for the four samples of length $n = 250$ from `assignment_train_sample.csv`. Construct the plug-in classifiers corresponding to the ECRM pseudo-classifiers and plot their corresponding decision boundaries alongside the data points (so you should have four figures). Also report the empirical risk of the obtained plug-in classifiers based on the 0-1 loss function for these four settings. 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 $(x_1, x_2, x_1 x_2, x_1^2, x_2^2)$.

8. **(15 pts)** Carry out question 7 again, but then for S_{nonlin}. Again, plot the decision regions and report the empirical risks rounded to three decimals. For each of the four parameter configurations, write down whether empirical risk increases or decreases when moving from S_{lin} to S_{nonlin}, and explain the intuition behind this pattern. Could this pattern in theory be reversed? Explain your answer.

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

Estimation and approximation error of ERM predictors

This is a continuation of Assignment Part I, so simply extend the code you wrote for that part of the assignment.

We are interested in obtaining the estimation and approximation error for the ERM predictors $\hat{f}_{S_r}(D_n)$ of question I.4, (D_n is suppressed from now on for notational convenience). To calculate these values, we need to know Bayes risk $\mathcal{R}_P^*$, the risk of the ERM predictor $\mathcal{R}_P(\hat{f}_{S_r}) = \mathbb{E}[c(\hat{f}_{S_r}(X), Y)|D_n]$ and the lowest possible risk over all predictors in S_r: $\inf_{f \in S_r} \mathcal{R}_P(f) = \inf_{f \in S_r} \mathbb{E}[c(f(X), Y)]$. These expectations are typically not available in closed form, but we can approximate them using Monte Carlo simulations. You already obtained an approximation of Bayes risk in question I.3, so only $\mathcal{R}_P(\hat{f}_{S_r})$ and $\inf_{f \in S_r} \mathcal{R}_P(f)$ remain to be approximated.

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

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

which converges almost surely to the true generalization error $\mathcal{R}_P(\hat{f}_{S_r}) = \mathbb{E}[\mathbb{1}_{\hat{f}_{S_r}(X) \neq Y}|D_n]$ as $\bar{n} \to \infty$. Furthermore, we can approximate $\inf_{f \in S_r} \mathcal{R}_P(f)$ by $\inf_{f \in S_r} \widehat{\mathcal{R}}_{\bar{n}}(f)$ 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.

Technical note: the sample size $\bar{n}$ that is needed to get an approximation with a particular accuracy depends for instance on the complexity of the model. Because this outside of the scope of this course, and to keep computation time limited, we use $\bar{n} = 1,000,000$ here for all models under consideration.

1. **(35 pts)** For the following questions, consider **configuration (i) of question I.2** and consider the model of histogram classifiers S_r defined in part I of the assignment, and consider 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`.

   (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 (i) as in question I.3 for this. Round your answers to 5 decimals.

   (b) **(15 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. 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) **(10 pts)** Describe the pattern you see in the *approximated* excess risk, estimation error and approximation error for the different values of r. Briefly explain the intuition behind the pattern you observe. 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 answer.

2. **(10 pts)** Answer the following questions:

   (a) **(3 pts)** For fixed r, is the model of histogram classifiers S_r a finite model? If so, what is Card S_r?

   (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 model S_r for $r = 2, 4, 8, 16$ based on samples of length $n = 250$ 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 the previous question. Is it surprising that the estimation errors are very far from the obtained upper bounds?

3. **(15 pts)** Now turn to the models of plug-in classifiers associated to the models of pseudo-classifiers S_{lin} and S_{nonlin} defined in part I of the assignment, which we define as $S^f_{\text{lin}} := \{f_h : h \in S_{\text{lin}}\}$ and $S^f_{\text{nonlin}} := \{f_h : h \in S_{\text{nonlin}}\}$, where still

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

   (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 no, 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 $\boldsymbol{n = 2500}$ (so not 250) in at least 90% of the cases? Show your calculations.

   (c) **(5 pts)** Can the upper bounds found in question (b) be applied 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.

Note: the remaining questions 4, 5 and 6 are on the following two pages.

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`

which contain the training data and validation 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.

4. **(15 pts)** For each combination of two features, train a support vector machine (SVM) with linear kernel on the training set (use the default value $C = 1$). Give the accuracies on the validation set for each pair of features. Plot the decision boundaries of the SVM predictors for the pairs of features with the highest and the lowest accuracy (so in total provide two plots, one for the best pair and one for the worst pair of features). Also plot the corresponding training data in the figures.

   Note: The accuracy of a predictor is simply 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.

[1] Retrieved from UCI Machine Learning Repository

5. **(10 pts)** Consider the best feature pair from the previous question. Train an SVM with RBF kernel on the training set, where you consider all combinations of $C = [0.1, 1, 10]$ and $\gamma = [0.1, 1, 10, 100, 1000]$. Calculate the accuracies on the validation set for each configuration and select the C and γ that lead to the highest accuracy. Is the accuracy higher than for the linear SVM?

6. **(15 pts)** Let (C^*, γ^*) denote the optimal hyperparameters from the previous question. Fix C to C^*. Plot the accuracies on the training set and on the validation set of the SVM predictors from the previous question as a function of γ for $\gamma = [0.1, 1, 10, 100, 1000]$ (use a $\log_{10}$ scale on the x-axis). So this should be one figure with two lines in it, namely a plot of the validation set accuracy and the training set accuracy. Also plot the decision boundaries of the SVM predictors corresponding to the five different values of γ. Analyse the resulting six figures. Do you see signs of overfitting for certain values of γ?

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). 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 `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, x2min, x1max, x1max, X, y, title, label_1,
                   label_2):
"""
  Purpose:
      Plot decision boundary on a [x1min,x1max] x [x2min,x2max] grid and plot some 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
      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.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 Simulating examples from P

To simulate examples (X_i, Y_i) from P, where P is as described in Part I of the assignment, you can take the following approach. First draw a vector of n outcomes $(y_1, \ldots, y_n)$, where each y_i is sampled from a Bernoulli distribution with probability parameter p. You can use `numpy.random.binomial(1,p,n)` for this. Then, for each $i = 1, \ldots, n$ draw a vector $x_i = (x_{1i}, x_{2i})$, where x_{1i} and x_{2i} are independent draws from a Beta distribution with parameters (α_1, β_1) if $y_i = 1$ and (α_0, β_0) if $y_i = 0$. You can simulate from the Beta distribution using the function `numpy.random.beta`.

```
import numpy as np

def sim_sample(n, p, alpha0, alpha1, beta0, beta1):
  """
  Purpose:
      Simulate a sample with n examples from P

  Inputs:
      n             int, length of sample to generate
      p             float, probability parameter Bernoulli dist
      alpha0        float, parameter alpha_0
      alpha1        float, parameter alpha_1
      beta0         float, parameter beta_0
      beta1         float, parameter beta_1

  Return value:
      y             vector, vector of n outcomes
      X             matrix, n x 2 matrix of simulated features
  """
  y = np.random.binomial(1,p,n)

  X = np.zeros((n,2))
  alpha_vec = [alpha0, alpha1]
```

```
    beta_vec = [beta0, beta1]

    for i in range(n):
        X[i,:] = np.random.beta(alpha_vec[y[i]],beta_vec[y[i]],size=2)

    return y, X
```

A.3 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 real value
  #' @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("blue","red"))
  filled.contour(grid1, grid2, Y_preds, xlab=label_1, ylab=label_2, main=
            title, axes=TRUE, nlevels = 2, color.palette = cp,
            plot.axes={points(X, col=ifelse(y==1, "red", "blue"), pch= 20)})
}
```

B.2 Simulating examples from P

To simulate examples (X_i, Y_i) from P, where P is as described in Part I of the assignment, you can take the following approach. First draw a vector of n outcomes $(y_1, \ldots, y_n)$, where each y_i is sampled from a Bernoulli distribution with probability parameter p. You can use `rbinom(n,1,p)` for this. Then, for each $i = 1, \ldots, n$ draw a vector $x_i = (x_{1i}, x_{2i})$, where x_{1i} and x_{2i} are independent draws from a Beta distribution with parameters (α_1, β_1) if $y_i = 1$ and (α_0, β_0) if $y_i = 0$. You can simulate from the Beta distribution using the function `rbeta`. You can direclty use the function below.

```
  sim_sample <- function(n, p, alpha0, alpha1, beta0, beta1){
  #' Simulate a sample with n examples from P
  #'
  #' @param n int, Length of sample to generate
  #' @param p float, probability parameter Bernoulli dist
  #' @param alpha0 float, parameter alpha_0
  #' @param alpha1 float, parameter alpha_1
  #' @param beta0 float, parameter beta_0
  #' @param beta1 float, parameter beta_1
  #' @return ys vector of n outcomes
  #' @return Xs n x 2 matrix of simulated features
  #'
  y <- rbinom(n,1,p)
  X <- matrix(data=NA, nrow=n, ncol=2)

  for (i in 1:n){
    if (y[i]==0){
      xx = rbeta(2,alpha0,beta0)
    } else {
      xx = rbeta(2,alpha1,beta1)
    }
```

```
    X[i,] <- xx
  }
  out <- list('ys' = y,'Xs' = X)              # Store output in list
  return(out)
}
```

B.3 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.