Machine Learning: Assignment Sheet #9

Due on April 19, 2022 at 10:00

Group HB
Henri Sota, Enis Mustafaj

Problem 9.1

We would like to solve a simple classification problem using paper and pencil. To this end, you are given the training data

$$\mathcal{T} = \{(0.3, 1), (1.8, 1), (1.5, 1), (4.8, 2), (2.6, 2)\}$$

The objective is to predict the class labels for the two evaluation points $x_1 = 2.4$ and $x_2 = 6.2$. Use linear regression on the indicator matrix to build the necessary predictor and evaluate it at x_1 and x_2. In addition, evaluate the *training error* using the 0-1 loss.

Firstly, we build the matrices $\mathcal{X}$ and $\mathcal{Y}$:

$$\mathcal{X} = \begin{pmatrix} 1 & 0.3 \\ 1 & 1.8 \\ 1 & 1.5 \\ 1 & 4.8 \\ 1 & 2.6 \end{pmatrix} \qquad \mathcal{Y} = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} \qquad \mathcal{X}^T = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0.3 & 1.8 & 1.5 & 4.8 & 2.6 \end{pmatrix}$$

By using the formula of the least squares, we have:

$$\beta = (\mathcal{X}^T \cdot \mathcal{X})^{-1} \cdot \mathcal{X}^T \cdot \mathcal{Y}$$

$$\beta = \begin{pmatrix} 5 & 11 \\ 11 & 35.38 \end{pmatrix}^{-1} \cdot \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0.3 & 1.8 & 1.5 & 4.8 & 2.6 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1.19033989 & -0.19033989 \\ -0.26833631 & 0.26833631 \end{pmatrix}$$

Now, we find the class that the input data belongs to by choosing the largest output between the entries of the result vector:

- $x_1 = 2.4$

$$y = \begin{pmatrix} 1 & x^T \end{pmatrix} \cdot \beta = \begin{pmatrix} 1 & 2.4 \end{pmatrix} \cdot \begin{pmatrix} 1.19033989 & -0.19033989 \\ -0.26833631 & 0.26833631 \end{pmatrix} = \begin{pmatrix} 0.54633274 \\ 0.45366726 \end{pmatrix}$$

  So, the input data belongs to the first class $(2.4, 1)$

- $x_2 = 6.2$

$$y = \begin{pmatrix} 1 & x^T \end{pmatrix} \cdot \beta = \begin{pmatrix} 1 & 6.2 \end{pmatrix} \cdot \begin{pmatrix} 1.19033989 & -0.19033989 \\ -0.26833631 & 0.26833631 \end{pmatrix} = \begin{pmatrix} -0.47334526 \\ 1.47334526 \end{pmatrix}$$

  So, the input data belongs to the second class $(6.2, 2)$

Now, we calculate the training error by using the 0-1 loss, where 0-1 loss is given by:

$$L_{0-1}(g, g') = \begin{cases} 0 & g = g' \\ 1 & g \neq g' \end{cases}$$

$$TE = \sum_{i=1}^{5} L_{0-1}(y_1, argmax((1, x^T) \cdot \beta))$$

$$TE = L_{0-1}(1, argmax\begin{pmatrix} 1.109839 \\ -0.109839 \end{pmatrix}) + L_{0-1}(1, argmax\begin{pmatrix} 0.70733453 \\ 0.29266547 \end{pmatrix}) + L_{0-1}(1, argmax\begin{pmatrix} 0.78783542 \\ 0.21216458 \end{pmatrix})$$
$$+ L_{0-1}(2, argmax\begin{pmatrix} -0.09767442 \\ 1.09767442 \end{pmatrix}) + L_{0-1}(2, argmax\begin{pmatrix} 0.49266547 \\ 0.50733453 \end{pmatrix}) = 0$$

Problem 9.2

We again start from the training data set

$$\mathcal{T}_{train} = \{(-2.1, 1), (-0.9, 1), (0.6, 2), (1.5, 2), (2.7, 2)\}$$

for a paper and pencil classification task. In addition, you are given the validation set

$$\mathcal{T}_{val} = \{(-1.2, 1), (0.5, 1), (1.4, 2)\}$$

a) Use linear discriminant analysis to build a classifier based on the training data.

- Firstly, we find the number of members for each class: $N_g = \{2,\ 3\}$.
- Then we find the probability set for each of the classes: $p(g) = \{\frac{2}{5}, \frac{3}{5}\} = \{0.4,\ 0.6\}$
- We find the set of mean values of the input data for each class:

$$\mu_g = \{\frac{-2.1 - 0.9}{2}, \frac{0.6 + 1.5 + 2.7}{3}\} = \{-1.5, 1.6\}$$

- Now, we find the covariance matrix by the formula:

$$\Sigma = \sum_{g=1}^{r} \sum_{(x,g)\in\mathcal{T}} (x - \mu_g) \cdot (x - \mu_g)^T / (N - r)$$

  Since the input data is one dimensional, the covariance matrix will be one dimensional as well.

$$\Sigma = \frac{(-2.1 + 1.5)^2}{3} + \frac{(-0.9 + 1.5)^2}{3} + \frac{(-2.1 - 0.6)^2}{3} + \\ \frac{(0.6 - 1.6)^2}{3} + \frac{(1.5 - 1.6)^2}{3} + \frac{(2.7 - 1.6)^2}{3} = 0.98$$

The produced classifier is expressed as:

$$g = argmax\left(\begin{pmatrix} -0.91629073 \\ -0.51082562 \end{pmatrix} + x \cdot \frac{1}{0.98} \cdot \begin{pmatrix} 0.4 \\ 0.6 \end{pmatrix} - \frac{1}{2} \cdot \begin{pmatrix} 0.4 & 0.6 \end{pmatrix} \cdot \frac{1}{0.98} \cdot \begin{pmatrix} 0.4 \\ 0.6 \end{pmatrix} \right)$$

b) Evaluate the generalization error for the just constructed predictor using the 0-1 loss and the validation set approach.

We evaluate the resulting class for each input data in the evaluation set:

- x_1 = -1.5, $g = argmax \begin{pmatrix} -0.22751522 \\ -3.77613175 \end{pmatrix} = 1$
- $x_2 = 0.5$, $g = argmax \begin{pmatrix} -2.82955604 \\ -1.00062154 \end{pmatrix} = 2$
- $x_3 = 1.4$, $g = argmax \begin{pmatrix} -4.20710706 \\ 0.46876621 \end{pmatrix} = 1$

From the results above, we see that the class predicted for the second input data is different from the initial class. Thus, the 0-1 loss will be 1 the generalization error is: $GE = 1$

Problem 9.3

Prove Lemma 8.2 from the lecture.

Lemma 8.2 - With the setting from Theorem 8.4, the gradient of the functional $J(\mathcal{B})$ with respect to a fixed class label g, which we call "∇_g" is given by

$$\nabla_g J(\mathcal{B}) = \begin{pmatrix} \frac{\partial}{\partial \beta_0^{(g)}} J(\mathcal{B}) \\ \vdots \\ \frac{\partial}{\partial \beta_D^{(g)}} J(\mathcal{B}) \end{pmatrix} = \frac{1}{N} \sum_{i=1}^{N} (p_{\mathcal{B}}(g|\boldsymbol{x}_i) - p(g|\boldsymbol{x}_i)) \begin{pmatrix} 1 \\ \boldsymbol{x}_i \end{pmatrix}$$

Based on the setting of Theorem 8.4 and indirectly Method 12, the following statements are known

$$s_g(\boldsymbol{x}) = \beta_g^{(g)} + \sum_{i=1}^{D} \beta_i^{(g)} x_i \qquad \text{for } g = 1, \dots, r$$

$$p_{\mathcal{B}}(g|\boldsymbol{x}) \approx \frac{\exp(s_g(\boldsymbol{x}))}{\sum_{h=1}^{r} \exp(s_h(\boldsymbol{x}))} \qquad \text{for } g = 1, \dots, r$$

We start from the result of Theorem 8.4, and the goal is to minimize the cost functional with respect to a given class g. This minimization can be found by calculating the gradient and equating it to 0. Since the solution is not dependent on any scaling, we are free to add any scaling.
Therefore, the cost functional that we are minimizing is:

$$\begin{aligned} \frac{1}{N} J(\mathcal{B}) &= -\frac{1}{N} \sum_{i=1}^{N} \left[\left(\sum_{g=1}^{r} p(g|\boldsymbol{x}_i) s_g(\boldsymbol{x}_i) \right) - \log \left(\sum_{h=1}^{r} \exp(s_h(\boldsymbol{x}_i)) \right) \right] \\ &= -\frac{1}{N} \sum_{i=1}^{N} \left(p(1|\boldsymbol{x}_i) \cdot \left[\beta_0^{(1)} + \sum_{j=1}^{D} \beta_j^{(1)} \boldsymbol{x}_{ij} \right] + \ldots + p(r|\boldsymbol{x}_i) \cdot \left[\beta_0^{(r)} + \sum_{j=1}^{D} \beta_j^{(r)} \boldsymbol{x}_{ij} \right] \right) \\ &\qquad - \log \left(\exp \left(\beta_0^{(1)} + \sum_{j=1}^{D} \beta_j^{(1)} \boldsymbol{x}_{ij} \right) + \ldots + \exp \left(\beta_0^{(r)} + \sum_{j=1}^{D} \beta_j^{(r)} \boldsymbol{x}_{ij} \right) \right) \Bigg) \end{aligned}$$

The gradient with respect to the coefficients $\mathcal{B}$, given a fixed class g, can be calculated as:

$$\nabla_g \frac{1}{N} J(\mathcal{B}) = \frac{1}{N} \begin{pmatrix} \frac{\partial}{\partial \beta_0^{(g)}} J(\mathcal{B}) \\ \vdots \\ \frac{\partial}{\partial \beta_D^{(g)}} J(\mathcal{B}) \end{pmatrix}$$

$$\frac{\partial}{\partial \beta_0^{(g)}} \frac{1}{N} J(\mathcal{B}) = -\frac{\partial}{\partial \beta_0^{(g)}} \frac{1}{N} \sum_{i=1}^{N} \left(p(1|\boldsymbol{x}_i) \cdot \left[\beta_0^{(1)} + \sum_{j=1}^{D} \beta_j^{(1)} \boldsymbol{x}_{ij} \right] + \ldots + p(r|\boldsymbol{x}_i) \cdot \left[\beta_0^{(r)} + \sum_{j=1}^{D} \beta_j^{(r)} \boldsymbol{x}_{ij} \right] \right)$$
$$- \log \left(\exp \left(\beta_0^{(1)} + \sum_{j=1}^{D} \beta_j^{(1)} \boldsymbol{x}_{ij} \right) + \ldots + \exp \left(\beta_0^{(r)} + \sum_{j=1}^{D} \beta_j^{(r)} \boldsymbol{x}_{ij} \right) \right)$$
$$\vdots$$
$$\frac{\partial}{\partial \beta_D^{(g)}} \frac{1}{N} J(\mathcal{B}) = -\frac{\partial}{\partial \beta_D^{(g)}} \frac{1}{N} \sum_{i=1}^{N} \left(p(1|\boldsymbol{x}_i) \cdot \left[\beta_0^{(1)} + \sum_{j=1}^{D} \beta_j^{(1)} \boldsymbol{x}_{ij} \right] + \ldots + p(r|\boldsymbol{x}_i) \cdot \left[\beta_0^{(r)} + \sum_{j=1}^{D} \beta_j^{(r)} \boldsymbol{x}_{ij} \right] \right)$$
$$- \log \left(\exp \left(\beta_0^{(1)} + \sum_{j=1}^{D} \beta_j^{(1)} \boldsymbol{x}_{ij} \right) + \ldots + \exp \left(\beta_0^{(r)} + \sum_{j=1}^{D} \beta_j^{(r)} \boldsymbol{x}_{ij} \right) \right)$$

Using the additive property, in this case, partial derivative of a sum is the sum of the partial derivatives. To avoid performing the following steps manually, we are taking the partial derivative of one of the coefficients, β_k, where $k \in \{0, \ldots, D\}$ to represent what we are doing for each partial derivative of the gradient.

$$\frac{\partial}{\partial \beta_k^{(g)}} \frac{1}{N} J(\mathcal{B}) = -\frac{\partial}{\partial \beta_k^{(g)}} \frac{1}{N} \sum_{i=1}^{N} \left(p(1|\boldsymbol{x}_i) \cdot \left[\beta_0^{(1)} + \sum_{j=1}^{D} \beta_j^{(1)} \boldsymbol{x}_{ij} \right] + \ldots + p(r|\boldsymbol{x}_i) \cdot \left[\beta_0^{(r)} + \sum_{j=1}^{D} \beta_j^{(r)} \boldsymbol{x}_{ij} \right] \right)$$
$$- \log \left(\exp \left(\beta_0^{(1)} + \sum_{j=1}^{D} \beta_j^{(1)} \boldsymbol{x}_{ij} \right) + \ldots + \exp \left(\beta_0^{(r)} + \sum_{j=1}^{D} \beta_j^{(r)} \boldsymbol{x}_{ij} \right) \right)$$
$$= -\frac{1}{N} \sum_{i=1}^{N} \frac{\partial}{\partial \beta_k^{(g)}} \left(p(1|\boldsymbol{x}_i) \cdot \left[\beta_0^{(1)} + \sum_{j=1}^{D} \beta_j^{(1)} \boldsymbol{x}_{ij} \right] + \ldots + p(r|\boldsymbol{x}_i) \cdot \left[\beta_0^{(r)} + \sum_{j=1}^{D} \beta_j^{(r)} \boldsymbol{x}_{ij} \right] \right)$$
$$- \frac{\partial}{\partial \beta_k^{(g)}} \log \left(\exp \left(\beta_0^{(1)} + \sum_{j=1}^{D} \beta_j^{(1)} \boldsymbol{x}_{ij} \right) + \ldots + \exp \left(\beta_0^{(r)} + \sum_{j=1}^{D} \beta_j^{(r)} \boldsymbol{x}_{ij} \right) \right)$$

Considering only the first term of the sum:

$$\frac{\partial}{\partial \beta_k^{(g)}} \left(p(1|\boldsymbol{x}_i) \cdot \left[\beta_0^{(1)} + \sum_{j=1}^{D} \beta_j^{(1)} \boldsymbol{x}_{ij} \right] + \ldots + p(r|\boldsymbol{x}_i) \cdot \left[\beta_0^{(r)} + \sum_{j=1}^{D} \beta_j^{(r)} \boldsymbol{x}_{ij} \right] \right)$$

Taking the partial derivative only given the class g will leave only one of the product terms in the sum.

$$\frac{\partial}{\partial \beta_k^{(g)}} \left(p(g|\boldsymbol{x}_i) \cdot \left[\beta_0^{(g)} + \sum_{j=1}^{D} \beta_j^{(g)} \boldsymbol{x}_{ij} \right] \right)$$

Continuing the derivation with respect to the coefficient β_k:

$$\frac{\partial}{\partial \beta_k^{(g)}} \left(p(g|\boldsymbol{x}_i) \cdot \left[\beta_0^{(g)} + \sum_{j=1}^{D} \beta_j^{(g)} \boldsymbol{x}_{ij} \right] \right) = \begin{cases} p(g|\boldsymbol{x}_i) \cdot 1 & \text{if } k = 0 \\ p(g|\boldsymbol{x}_i) \cdot x_{ik} & \text{otherwise} \end{cases} \quad \text{for } k \in \{0, \ldots, D\}$$

Condensing the result to find the gradient of only the first term of them sum given the class g with respect to the set of coefficients:

$$\begin{pmatrix} \frac{\partial}{\partial \beta_0^{(g)}} \left(p(g|\boldsymbol{x}_i) \cdot \left[\beta_0^{(g)} + \sum_{j=1}^{D} \beta_j^{(g)} \boldsymbol{x}_{ij} \right] \right) \\ \vdots \\ \frac{\partial}{\partial \beta_D^{(g)}} \left(p(g|\boldsymbol{x}_i) \cdot \left[\beta_0^{(g)} + \sum_{j=1}^{D} \beta_j^{(g)} \boldsymbol{x}_{ij} \right] \right) \end{pmatrix} = \begin{pmatrix} p(g|\boldsymbol{x}_i) \\ \vdots \\ p(g|\boldsymbol{x}_i) \cdot x_{iD} \end{pmatrix} = p(g|\boldsymbol{x}_i) \cdot \begin{pmatrix} 1 \\ \boldsymbol{x}_i \end{pmatrix}$$

Considering only the second term of the sum:

$$\frac{\partial}{\partial \beta_k^{(g)}} \log \left(\exp \left(\beta_0^{(1)} + \sum_{j=1}^{D} \beta_j^{(1)} \boldsymbol{x}_{ij} \right) + \ldots + \exp \left(\beta_0^{(r)} + \sum_{j=1}^{D} \beta_j^{(r)} \boldsymbol{x}_{ij} \right) \right)$$

Letting z be equal to the sum of exponentials inside of the logarithm, and continuing the derivation:

$$\frac{\partial}{\partial \beta_k^{(g)}} \log(z) = \frac{1}{z} \frac{\partial}{\partial \beta_k^{(g)}} \left(\exp \left(\beta_0^{(1)} + \sum_{j=1}^{D} \beta_j^{(1)} \boldsymbol{x}_{ij} \right) + \ldots + \exp \left(\beta_0^{(r)} + \sum_{j=1}^{D} \beta_j^{(r)} \boldsymbol{x}_{ij} \right) \right)$$

Taking the partial derivative only given the class g will leave only one exponential term in the sum.

$$\frac{1}{z} \frac{\partial}{\partial \beta_k^{(g)}} \exp \left(\beta_0^{(g)} + \sum_{j=1}^{D} \beta_j^{(g)} \boldsymbol{x}_{ij} \right)$$

Continuing the derivation with respect to the coefficient β_k:

$$\frac{1}{z} \frac{\partial}{\partial \beta_k^{(g)}} \exp \left(\beta_0^{(g)} + \sum_{j=1}^{D} \beta_j^{(g)} \boldsymbol{x}_{ij} \right) = \begin{cases} \frac{1}{z} \exp \left(\beta_0^{(g)} + \sum_{j=1}^{D} \beta_j^{(g)} \boldsymbol{x}_{ij} \right) \cdot 1 & \text{if } k = 0 \\ \frac{1}{z} \exp \left(\beta_0^{(g)} + \sum_{j=1}^{D} \beta_j^{(g)} \boldsymbol{x}_{ij} \right) \cdot x_{ik} & \text{otherwise} \end{cases} \quad \text{for } k \in \{0, \ldots, D\}$$

From the setting, we recall that:

$$p_{\mathcal{B}}(g|\boldsymbol{x}) \approx \frac{\exp(s_g(\boldsymbol{x}))}{\sum_{h=1}^{r} \exp(s_h(\boldsymbol{x}))} = \frac{1}{z} \exp \left(\beta_0^{(g)} + \sum_{j=1}^{D} \beta_j^{(g)} \boldsymbol{x}_{ij} \right)$$

Condensing the result to find the gradient of only the second term of them sum given the class g with respect to the set of coefficients:

$$\begin{pmatrix} \frac{\partial}{\partial \beta_0^{(g)}} \left(\frac{1}{z} \exp \left(\beta_0^{(g)} + \sum_{j=1}^{D} \beta_j^{(g)} \boldsymbol{x}_{ij} \right) \right) \\ \vdots \\ \frac{\partial}{\partial \beta_D^{(g)}} \left(\frac{1}{z} \exp \left(\beta_0^{(g)} + \sum_{j=1}^{D} \beta_j^{(g)} \boldsymbol{x}_{ij} \right) \right) \end{pmatrix} = \begin{pmatrix} p_{\mathcal{B}}(g|\boldsymbol{x}_i) \\ \vdots \\ p_{\mathcal{B}}(g|\boldsymbol{x}_i) \cdot x_{iD} \end{pmatrix} = p_{\mathcal{B}}(g|\boldsymbol{x}_i) \cdot \begin{pmatrix} 1 \\ \boldsymbol{x}_i \end{pmatrix}$$

Combining both results:

$$\begin{aligned} \nabla_g \frac{1}{N} J(\mathcal{B}) = \frac{1}{N} \begin{pmatrix} \frac{\partial}{\partial \beta_0^{(g)}} J(\mathcal{B}) \\ \vdots \\ \frac{\partial}{\partial \beta_D^{(g)}} J(\mathcal{B}) \end{pmatrix} &= \frac{1}{N} \begin{pmatrix} -\sum_{i=1}^{N} p(g|\boldsymbol{x}_i) - p_{\mathcal{B}}(g|\boldsymbol{x}_i) \\ \vdots \\ -\sum_{i=1}^{N} p(g|\boldsymbol{x}_i) \cdot x_{iD} - p_{\mathcal{B}}(g|\boldsymbol{x}_i) \cdot x_{iD} \end{pmatrix} \\ &= -\frac{1}{N} \begin{pmatrix} \sum_{i=1}^{N} (p(g|\boldsymbol{x}_i) - p_{\mathcal{B}}(g|\boldsymbol{x}_i)) \cdot 1 \\ \vdots \\ \sum_{i=1}^{N} (p(g|\boldsymbol{x}_i) - p_{\mathcal{B}}(g|\boldsymbol{x}_i)) \cdot x_{iD} \end{pmatrix} \\ &= \frac{1}{N} \sum_{i=1}^{N} (p_{\mathcal{B}}(g|\boldsymbol{x}_i) - p(g|\boldsymbol{x}_i)) \begin{pmatrix} 1 \\ \boldsymbol{x}_i \end{pmatrix} \end{aligned}$$

Programming Problem 9.1

a) Start from Example 8.1 from the lecture notes for which you have the Python source code available as Jupyter notebook. Ignore kNN classification and linear regression on indicator matrix and (only) re-implement the linear discriminant analysis by yourself. Verify the correctness of your implementation by cross-checking it with Example 8.1.

   The implementation of linear discriminant analysis can be found in the file `programming_exercises.ipynb`. After the implementation, the comparison is done and an accuracy of our implementation based on the results of Example 8.1 is given.

b) Now, we would like to compare the performance of the just implemented linear discriminant analysis to the performance of kNN classification (based e.g. on Scikit-learn) on SPAM data. Use as data set the Spambase Data Set from the UCI Machine Learning Repository. To carry out the comparison, implement the validation set approach with the 0-1 loss. Randomly split the data set into $N_{train} = 1000$ training samples and $N_{val} = 100$ validation samples and use the same split to evaluate the generalization error for LDA and kNN (with $k = 3$) classification.

   The solution to this part of the exercise can be found in the file `programming_exercises.ipynb`. The validation set approach is based on the split retrieved by `train_test_split` function. The generalization error based on the validation set approach is calculated using the function `zero_one_loss`, whose implementation is just a simple loop going over each pair of outputs, real and predicted, and summing up the cases when the predicted output is not the same as the test set one. In the end, the result is divided by the size of the test split.