Machine Learning

Exercise Sheet 4

Winter Term 2023/2024
Prof. Dr. Niels Landwehr
Dr. Ujjwal

Available: 23.11.2023
Hand in until: 30.11.2023 11:59am
Exercise sessions: 04.12.2023/06.12.2023

Task 1 – Linear Discriminant Analysis [10 points]

In this exercise, we will study a linear discriminate analysis model for the toy data set given in Task 1 of Exercise Sheet 3. We extend the toy data set by one additional positive and one additional negative example, and remove the constant attribute:

$$\mathbf{X} = \begin{pmatrix} 1 & 3 \\ 2 & 4 \\ 1 & 4 \\ 2 & 2 \\ 3 & 3 \\ 3 & 2 \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} 2 \\ 2 \\ 2 \\ 1 \\ 1 \\ 1 \end{pmatrix}. \tag{1}$$

Here, we have assumed that the negative class is class one and the positive class is class two.

a) For training data $\mathbf{X}, \mathbf{y}$ as given, compute the maximum-likelihood parameters $\boldsymbol{\pi} \in \mathbb{R}^2$, $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2 \in \mathbb{R}^2$, and the (shared) covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{2\times 2}$. You are welcome to use a programming environment such as Numpy to perform matrix calculations, but please state clearly the mathematical formulae that you use and give intermediate results.

b) How would the new data point $\mathbf{x}_{new} = (1.5, 3)$ be classified by the linear discriminant analysis model? For computing the multivariate normal density, it is advised to use a programming environment such as *scipy.stats*.

Task 2 – Normal Equations for Ridge Regression [15 points]

Assume a linear regression model $f_{\boldsymbol{\theta}} : \mathbb{R}^M \to \mathbb{R}$ given by

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = \theta_1 x_1 + ... + \theta_m x_m \tag{2}$$

$$= \mathbf{x}^\mathsf{T}\boldsymbol{\theta} \tag{3}$$

Assume we learn the model from data $\mathbf{X} \in \mathbb{R}^{N\times M}$, $\mathbf{y} \in \mathbb{R}^N$ (where the rows of $\mathbf{X}$ contain the training instances $\mathbf{x}_1, ..., \mathbf{x}_N$ and $\mathbf{y}$ contains the labels $y_1, ..., y_N$) by minimizing the squared loss plus an L2-regularization term,

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) \tag{4}$$

$$L(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N}(f_{\boldsymbol{\theta}}(\mathbf{x}_n) - y_n)^2 + \lambda\|\boldsymbol{\theta}\|_2^2 \tag{5}$$

where $\lambda > 0$ is the regularization weight. We want to show that the optimal model parameters $\boldsymbol{\theta}^*$ are given by

$$\boldsymbol{\theta}^* = (\mathbf{X}^\mathsf{T}\mathbf{X} + N\lambda\mathbf{I})^{-1}\mathbf{X}^\mathsf{T}\mathbf{y} \tag{6}$$

where $\mathbf{I}$ is the $M \times M$ identity matrix, that is, a diagonal matrix with value one everywhere on the diagonal. To show that a minimum is obtained at the $\boldsymbol{\theta}^*$ of Equation **??**, show that at $\boldsymbol{\theta}^*$

a) the gradient $\frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$ is zero and

b) the Hessian matrix is positive definite (note: positive semidefinite is not sufficient here).

You may use the results from Slide 36 in the lecture on linear regression and Slide 36 in the lecture on regularization.

Task 3 – Programming: Variable Selection [15 points]

In this task we implement the forward and backward search algorithms for a linear regression model on the Ccalifornia Housing data set in Python.
The ipython notebook *Ex04_Task_3.ipynb* linked in the Learnweb provides a template that can serve as a starting point for the solution and contains additional building blocks and advice to solve this exercise. You can load this notebook for example in Google colab (`https://colab.research.google.com/`) or any other environment for Python notebooks.
First run the code below to load the California Housing data set and split into a training and validation set.

```
import numpy as np
from sklearn.datasets import fetch_california_housing
np.random.seed(123)
dataset = fetch_california_housing()
Xdata = dataset["data"]
Ydata = dataset["target"]
N, M = Xdata.shape
# Split into training and validation sets
ridx = np.random.permutation(N)
split = int(0.8*N)
Xtrain = Xdata[ridx[:split]]
Ytrain = Ydata[ridx[:split]]
Xvalid = Xdata[ridx[split:]]
Yvalid = Ydata[ridx[split:]]
```

A linear regression model for this data set can be implemented with the scikit-learn library as follows:

```
from sklearn.linear_model import LinearRegression
reg = LinearRegression().fit(Xtrain, Ytrain) # fit model
intcp = reg.intercept_ # get intercept
coefs = reg.coef_  # get coefficients
```

The training and validation squared error can be calculated with scikit-learn as follows:

```
# Get Predictions for training and validation sets
pred_train = reg.predict(Xtrain)
pred_val = reg.predict(Xvalid)
```

```
# Calculate Loss on training and validation sets
from sklearn.metrics import mean_squared_error
train_loss = mean_squared_error(pred_train, Ytrain)
val_loss = mean_squared_error(pred_val, Yvalid)
```

a) Implement your own learning algorithm for linear regression by solving the normal equations and calculate its loss on training and validation set. Your solution should be based on basic Numpy functions for matrix multiplication, solving sets of linear equations etc. (that is, without using Scikit-learn). Print the coefficient values, intercept, training and validation loss to show that they match the results from the scikit-learn implementation.

b) Implement the forward and backward search algorithms to select a subset of attributes for the linear regression model. Use the validation loss as scoring function. In each outer loop print the selected variables and training and validation loss.