Supervised Learning: Linear Regression

Introduction

- Before you start, be sure to read well the slides of the course.
- **Objectives:**
  1. Understand linear regression using the formulas of the course.
  2. Use `Python` libraries to apply regression to simulated and real-world datasets. In particular, we will use `numpy`, `pandas` and `scikit-learn`.
- Be curious : Play with the parameters (for example the number of training data, the variance of the noise, ...), change the true underlying function, ... and try to understand.
- Avoid the use of chatGPT, or other large language models, to solve the TP. You are here to make mistakes, scratch your head and learn!
- Implement all the TP exercises in one or several `Python` notebooks. Make extensive use of comments and markdown blocks in order to explain your code.

1 Linear regression

1. Create a function `simple_linear_regression(X,y)` that receives a vector of a descriptor variable `X`, a response vector `y` and implements the simple linear regression [2, p.211]:

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i,\ \bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i,$$

$$\hat{\beta_0} = \bar{y} - \hat{\beta_1}\bar{x} \quad \text{and} \quad \hat{\beta_1} = \frac{\sum_{i=1}^{N}(y_i - \bar{y})(x_i - \bar{x})}{\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

2. A hypermarket has 20 checkouts. We focus on the average waiting time for clients expressed in minutes, denoted y, versus the number of available checkouts, denoted x. The dataset of size $N = 7$ is given in the following table:

x	Number of available checkouts	3	4	5	6	8	10	12
y	Average waiting time	16	12	9.5	8	6	4.5	4

The objective is to perform a linear regression to predict the average waiting time for clients given the number of available checkouts.

(a) **Data visualization:** Define the input x and the output y as `numpy` arrays and visualize the scatter plot corresponding to the dataset $\{(x_i, y_i)\}_{i=1}^{N}$.

Remark. Use the function `scatter` from `matplotlib.pyplot` module. For instance:

```
>>> import matplotlib.pyplot as plt
>>> # X and y are numpy arrays
>>> plt.scatter(X,y)
```

(b) **Data statistics estimation:** Estimate the means, the variances, the covariance and the correlation coefficient of x and y using the `numpy` functions `mean` and `var`. Are the number of available checkouts and the average waiting time correlated ?

(c) **Training / Learning:** Apply the function `simple_linear_regression(X,y)` to estimate the coefficients $\hat{\beta}_0$ and $\hat{\beta}_1$ of the regression line (with equation: $y = \hat{\beta}_0 + \hat{\beta}_1 x$) from the training set $\{(x_i, y_i)\}_{i=1}^N$. Plot the regression line on top of the scatter plot of the data.

(d) **Test / Prediction:** Give a prediction of the average waiting time if the hypermarket has only 1 available checkout, 7 available checkouts, 20 available checkouts.
What do you think about the linear model for this problem ? Is it appropriate ?

2 Polynomial regression

1. Using `numpy`, create a function that generates the following artificial dataset:

   (a) $y_i = 10 + 5x_i + 4\sin(10x_i) + \epsilon_i,\ \epsilon_i \sim \mathcal{N}(0,4), x_i \sim \mathcal{U}(0,1),\ i = 1, \ldots, 80.$

2. Create a function `linear_regression(X,y)` that implements the multidimensional solution for linear regression with

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T y. \tag{1}$$

   Since the vector $\hat{\boldsymbol{\beta}} \in \mathbb{R}^{d+1}$ *includes* the intercept $\hat{\beta}_0$, the matrix $\mathbf{X}$ should be

$$\mathbf{X} = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \ldots & x_{1,d} \\ 1 & x_{1,1} & x_{2,2} & \ldots & x_{2,d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N,1} & x_{N,2} & \ldots & x_{N,d} \end{bmatrix}.$$

   Remark. The matrix $\mathbf{X}^+ = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$ is called the Moore-Penrose pseudo-inverse [1], and can be easily computed using the function `numpy.linalg.pinv()`. Additionally, remember that matrix multiplication is implemented by the `@` operator in `Python`.

3. Create a function `phi(X,order)` that implements

$$\phi(x) : \mathbb{R} \to \mathbb{R}^{\text{order}+1},\ \phi(x) = [1, x, x^2, \ldots, x^{\text{order}}].$$

4. Apply the function `linear_regression(X,y)` to the dataset created in Exercise 1a. Then using the function `phi(X,order)`, apply `linear_regression(phi(X, order=20),y)` to the same dataset. Plot both solutions on top of a scatter plot of the data. Which solution overfits and which one underfits?

3 Regression with `scikit-learn`

Elements from `scikit-learn`. The table below shows the modules of `scikit-learn` and the functions/ classes that are used in the following sections.

Module	Function / class
`sklearn.linear_model`	`LinearRegression`, `Ridge`, `Lasso`
`sklearn.metrics`	`mean_squared_error`, `r2_score`,
`sklearn.model_selection`	`KFold`, `train_test_split`
`sklearn.datasets`	`load_diabetes`
`sklearn.feature_selection`	`SequentialFeatureSelector`

Data Partition. Often we will need to fix parameters of our model, such as the order of the polynomial to fit, the regularization parameter, the number features to use, etc. As seen in the course, determining our model using all the available data results in severe overfitting and poor generalization. In order to appropriately choose a regularization parameter one could follow these steps:

1. Separate the data into **disjoint sets**, that we call **train** and **test** (or **validation**) sets.
2. Fix the parameter and fit the model using the train set.
3. Evaluate the performance of the model on the test set using some **score** or **performance metric**.
4. Repeat steps 2 and 3 for several values of the parameter and save the score for each value.
5. Finally, choose the parameter value with the best score.

`scikit-learn` offers the convenient function `train_test_split()` that can easily separate the data into disjoint sets for us:

```
>>> from sklearn.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

The input argument `test_size` fixes the proportion of data used in the test set. Make sure to give the `random_state` input argument some fixed value to ensure **repeatability** of your experiments.

Cross-validation. K-fold cross-validation is a generalization of the previous procedure that separates the data in K disjoint subsets, and repeats all the previous steps K times. Each time, a train set is built using $K-1$ subsets and the test set is the remaining subset. In the end, we get K values of the estimated parameter that we can average to have a final estimation.
Cross-validation can be easily implemented using the `Kfold` class. For instance:

```
>>> from sklearn.model_selection import KFold
>>> kf = KFold(n_splits=5,shuffle=True,random_state=0)
>>> for train, test in kf.split(X):
>>>     X_train, y_train = X[train], y[train] # Train the model with these.
>>>     X_test, y_test = X[test], y[test] # Test the model with these.
```

3.1 Simple linear regression

1. **Data reading and visualization:** Using `pandas.read_csv()` function import the dataset `house_rent.csv` which stores, for 545 flats in Paris, the rent and the surface of each flat. The objective is to perform a linear regression to predict the rent of a flat given its surface. Plot a scatter plot of the dataset. Do you think that a linear model is appropriate for the regression ? To ensure the validity of the model, you can remove outliers, for example by choosing a range of values for the rent or the surface. Keep only data with a rent lower than 10000.
2. **Data partitioning:** Split the dataset into train and validation sets using `train_test_split()`. Use 20% of the data for the test set.
3. **Training:** Using `'surface'` as descriptor and `'loyer'` as response variable, fit the linear model `LinearRegression` to the training set. Plot a scatter plot of the dataset and the line corresponding to the solution on top.

   Remark. All regression (and classification) models from `scikit-learn` implement a class method called `fit()` for the **training** or **learning** stage. An example in this case would be:

   ```
   >>> from sklearn.linear_model import LinearRegression
   >>> regressor = LinearRegression()
   >>> regressor.fit(X_train,y_train) # X: descriptors, y: response variable.
   ```

4. **Prediction:** Recover the estimated coefficients $\hat{\boldsymbol{\beta}}$ and the intercept $\hat{\beta}_0$ as attributes of the `LinearRegression` object, e.g. `regressor.coef_` and `regressor.intercept_`. These can be useful to compute **predictions** on the test set as $y = \hat{\boldsymbol{\beta}}^T \mathbf{x} + \hat{\beta}_0$, or using the `predict()` class function as:
   ```
   >>> y_pred = regressor.predict(X_test).
   ```
   Plot on the same figure the predicted values `y_pred` and the true output values `y_test` (use different colors).

5. **Analysis of the performance:**

   (a) Respectively on training and test data, compute:
   - the mean squared error (MSE) between the predicted output values and the true output values using the `mean_squared_error()` function. Derive the root mean squared error (RMSE).
   - another regression metrics called R^2 score using the `r2_score()` function from module. It equals the square of the correlation coefficient between the predicted and true output values. An R^2 of 1 indicates that the regression predictions perfectly fit the data.

   (b) When working with real data, it is important to check the residuals and their distribution. Plot the residuals for the `house_rent.csv` data. Draw conclusions about their distribution. How should the residuals be distributed according to our assumptions?

3.2 Ridge and lasso regression

Now we use a bigger dataset `house_price.csv` which stores, for houses in United States, the sale price versus a great number of features which are described in the file `data_description.txt`. The objective is to predict the sale price of a house given its features.

1. **Data reading and visualization:** Read the csv file into a `DataFrame` using the `read_csv()` function from `pandas` library. Visualize and analyze the dataset. Give the size of the dataset : number of houses and number of features per house.

2. **Data preprocessing:** Use the code provided in the file `data_preprocess.py` to preprocess the data. Analyze the code to identify the different transformations performed on data and the decomposition of data into an input matrix `X_data` and an output vector `y_data`.

3. **Data partitioning:** Split the dataset into training and test subsets using the `train_test_split` function.

4. Since the number of features is high, it is interesting to introduce **regularization** in the model to avoid overfitting. Therefore, we need to choose an appropriate value of the regularization parameter α. For this, create a list of possible values for α, e.g.: $[10^{-4}, 10^{-3}, \ldots, 10^{2}, 10^{3}]$, and repeat the following steps for each value:

   (a) **Training** Fit the `Ridge` model to the train subset. For each value of α, store the values of the estimated regression coefficients given by the attribute `coef_`.

   Remark. Both `Ridge` and `Lasso` models can receive a value for the regularization parameter α at the moment of instantiation:
   ```
   >>> from sklearn.linear_model import Ridge
   >>> regressor = Ridge(alpha=0.001) # Give a value to alpha
   ```

   (b) **Prediction:** Predict the output values of the test data `X_test` using the method `predict()` of the `Ridge` class. For each value of α, store the predicted values.

   (c) **Analysis of the performance:** Compute the RMSE between the predicted test values and the true test values using the `mean_squared_error()`. For each value of α, store the RMSE.

5. Now, plot the following curves:

(a) **Regression coefficients vs α:** Plot the evolution of the different coefficients versus the regularization parameter α using the `semilogx()` function from `matplotlib.pyplot` module.

(b) **RMSE vs α:** Plot the RMSE versus the regularization parameter α using the `semilogx()` function from `matplotlib.pyplot` module. Based on this plot, **determine the best value for α.**

6. Repeat exercises 4 and 5 but now using the `Lasso` model. Once you find an appropriate value of α for Lasso regularization:

   (a) Identify the regression coefficients equal to 0. How many features have been eliminated by the Lasso regularization ?

   (b) Compare the performance obtained with a Lasso regularization to the performance obtained with the Ridge regression.

Optional Exercises

1. Load the diabetes dataset using `load_diabetes(return_X_y=True, as_frame=True)`. Explore the `DataFrame` and get a list of the 10 covariate names. What is the response variable?

   Hint. Read the documentation of the dataset.

2. Fit a `SequentialFeatureSelector` model, using `LinearRegression`, to the complete diabetes dataset (i.e. use the 10 features). Reduce the model to only two features. What descriptors are left? Do you get the same features for both backward and forward approaches? What is the default score used to quantify the performance of the models?

   Remark. Example:

```
>>> from sklearn.feature_selection import SequentialFeatureSelector as SFS
>>> regressor = LinearRegression()
>>> sfs_forward = SFS(regressor,n_features_to_select=2,direction="forward")
>>> sfs_forward.fit(X,y)
>>> selected_features = sfs_forward.get_support()
```

3. Implement a function `my_aic(regressor,X,y)` that computes the Akaike information criterion (AIC) for a regressor (for instance the `Ridge` model). Repeat the previous exercise using AIC as score. What features are selected in this case?

   Remark. The `SequentialFeatureSelector` can also receive a scoring function via the `scoring` input argument, for instance your function `my_aic(regressor,X,y)`.

References

[1] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[2] Larry Wasserman. *All of statistics: a concise course in statistical inference*. Springer Science & Business Media, 2013.