Assignment IV: Bayes'Theorem and Ridge Regression

Exercises in Machine Learning (190.013), SS2022
Stefan Nehl[1]

[1]*stefan-christopher.nehl@stud.unileoben.ac.at, MNr: 00935188*, *Montanuniversität Leoben, Austria*

May 8, 2022

In the fourth assignment, I had to solve three different task. First, calculate the probability for a positive test results with the Bayes' Theorem, next describe the ridge regression and derive the weight update for with the least squares regression and last implement the ridge regression. The implementation of the ridge regression also includes testing the model and plotting it's results.

1 Task 1: Bayes'Theorem

The Bayes'Theorem is a mathematical formula which describes the probability of an event. Furthermore, it is used for calculating conditional probabilities. (**bayesTheoremHist**)

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)}$$

(Rueckert, 2022)

I used this formula to calculate the probability to be infected with SARS CoV2 and having a positive test result of an antigen test. Let A $\in$ [infected, non-infected] the event, which defines if a person is infected or not, and B $\in$ [+,-] the event, which defines the result of the antigen test.

1.1 Implementation

First, I created the following variables with the values.

Table 1: *Variables and Values*

name	value
populationAustria	9095538
activeCases	441098
covTestSensitivity	0.971
covTestSpecific	0.995

First, I set the variable for *p(+|infected)* to the value of *covTestSensitivity* and the variable for *p(-|non-infected)* tp the value of *covTestSpecific*. Next, I calculated the value for *p(infected)*, *p(non-infected* and stored the values in the variables *pInfected* and *pNotInfected*.

$$p(infected) = \textit{activeCases}/\textit{populationAustria}$$

$$p(non-infected) = \textit{activeCases}/\textit{populationAustria}$$

Next, I initialized the variable *pInfectedAndPositive* and calculated the value with the following formula.

$$p(infected\&positive) = p(infected) * p(+|infected)$$

$$p(infected\&negative) = p(infected) - p(infected\&positive)$$

set the value *p(B)* to *covTestSpecific* and *p(B|A)* to *covTestSensitivity* and used the Bayes'Theorem to calculate *p(A|B)*.

1.2 Result

The result for *p(A|B)* was $0.04732 \approx 4.73\%$. Which means there is a 4.73% chance to be infected with covid and get a positive test result.

2 Task 2: Ridge Regression

Ridge regression is used for parameter estimation to address the collinearity problem in multiple linear regression. (**ridgeRegression**) The Ridge Regression adds the quadratic regularization term $\frac{\lambda}{2}$ $(\omega^{\mathrm{T}}\omega)$.

3 Task 3: Implementation of Ridge Regression

4 Gauss Distribution

The class *GaussDistribution* has a constructor with the parameters *dimension*, sets the dimension of this gauss distribution and the optional parameters *fileName*, for importing a *CSV* file, *numberOfSamplesToGenerate*, number of samples generated by the class, *mean* and *variance*. Important to mention here is, that the importing of a file and the generating of samples is excluding each other. Only one parameter can be set otherwise the class throws an exception. The construction also sets the data and calculate the needed values like mean and standard deviation and generates the gauss distribution.

4.1 Generating Samples

As already mentioned, the generating of samples needs to be implemented for each class separately. For the generation I used the *random()* function from the *numpy* library with the values of the mean and the variance for the generation and the dimension with the number of samples for the amount of data.

4.2 Calculation

For the calculation I implemented two different methods. One for the one dimensional calculation, *generateGaussen1D*, and for the two dimensional calculation, *calculateGaussen2D*. For the one dimensional implementation I used the following formula.

$$N(x|\mu,\sigma) = \frac{1}{(2\pi\sigma^2)^{1/2}}\, exp^{-\frac{1}{2\sigma^2}(x-\mu)^2}$$

Where μ stands for the mean and σ for the standard deviation. For the two dimensional implementation I used the following formula.

$$N(x|\mu,\sigma) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}}\, exp^{-\frac{1}{2}(x-\mu)^{T}\Sigma^{-1}(x-\mu)}$$

Where D is the dimension, Σ the covariance and $|\Sigma|$ the determinant of the covariance.For the covariance I used created a vector with the mean and zeros. $\begin{pmatrix} \sigma & 0 \\ 0 & \sigma \end{pmatrix}$ Both formulas are from the book An Introduction to Probabilistic Machine Learning. (Rueckert, 2022)

4.3 Plotting

The implementation for the one dimensional plot was straightforward. I plotted a histogram of the generated data and the gauss distribution as a line above the histogram. Additional, I add the raw the of the generated samples. With the two dimensional plots I had some issues. I was able to create a 3d model of the raw data and the 3d bar chart of the distribution. I wanted to plot the surface of the two dimensional gauss distribution following the paper (Roelants, 2018), but unfortunately I failed to create the surface.

5 Beta Distribution

The class *BetaDistribution* has analogue to the class *GaussDistribution* also a constructor for handling the initialization for the parameters. Also the limitation for file name for *CSV* or generation of samples is the same. The difference is, that the beta distribution needs the parameter a and b and not the dimension for the distribution.

5.1 Generating Samples

The generation for the samples was created again with the *random()* function from the *numpy* library. However, I changed the distribution to the beta distribution.

5.2 Calculation

For the calculation I used the following formula.

$$Beta(x|a,b) = B(a,b)x^{a-1}(1-x)^{b-1}$$

where a,b are in the constructor given parameters as a scalar and *B(a,b)* the Beta function.

$$B(a,b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}$$

where Γ is the gamma function. The formulas are again from book An Introduction to Probabilistic Machine Learning. (Rueckert, 2022)

5.3 Plotting

For plotting the results I used a histogram with the distribution and the raw data. In addition I created a plot with the beta distribution with different values for the parameters *alpha* and *beta*. The values are displayed in Table 1. For the plotting of the parameters with different values I added a method named *plotDataWithDifferentAlphasAndBetas* with a list of the different settings as a parameter. These function sets the values and generates the plots.

Table 2: *Values for alpha and beta*

alpha	beta
0.5	0.5
5	2
2	5

6 Results

Figure 1 shows the gauss distribution for one dimension. It displays the values distribution and the frequencies of those values. The orange line displays the gauss distribution itself.

7 Conclusion

The implementation of the abstract class was straightforward. For the other classes I made some changes. I moved the classes to separate files to handle them better. Unfortunately, I was not able to create a satisfying plot for the gauss distribution for two dimensions. The whole code is in the appendix of this paper.

Bibliography

Roelants, Peter (2018). *Multivariate normal distribution*. URL: `https://peterroelants.github.io/posts/multivariate-normal-primer/`.

Rueckert, Elmar (2022). *An Introduction to Probabilistic Machine Learning*. Elmar Rueckert.

APPENDIX