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 9, 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 Bayes'Theorem

The Bayes'Theorem is a mathematical formula which describes the probability of an event. Furthermore, it is used for calculating conditional probabilities. (Joyce, 2021)

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

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

$$p(inf) = \mathit{activeCases/populationAustria}$$

$$p(nInf) = 1 - p(inf)$$

The variable *p(inf)* defines the value for the probability to be infected with covid and *p(nInf)* not. Furthermore, the abbreviation for infected is *inf* and for non infected *nInf*.The abbreviation for having a positive test result is $+$ and for a negative test result $-$. Next, I initialized the following variables and calculated there values with the following formulas.

$$p(nInf\&-) = p(nInf) * p(-|non - infected)$$

$$p(-) = p(inf\&-) + p(nInf\&-)$$

$$p(nInf\&+) = p(nInf) - p(nInf\&-)$$

$$p(+) = p(inf\&+) + p(nInf\&+)$$

$$p(-|inf) = \frac{p(inf\&-)}{p(-)}$$

$$p(+|nInf) = \frac{p(nInf\&+)}{p(nInf)}$$

Last, I used the Bayes'Theorem to calculate the $p(infected|+)$ value.

$$p(inf|+) = \frac{p(+|inf) * p(inf)}{p(nInf)}$$

1.2 Result and Conclusion

The results of the calculation is displayed in Table 2. The result for *p(inf|+)* is $0.630525 \approx 63.05\%$. Which means there is a 63.05% chance to be infected with covid and get a positive test result. The implemented code can be found in the appendix of this paper.

Table 2: *Results*

name	value
p(-|inf)	00.15%
p(+|nInf)	02.90%
p(inf)	04.85%
p(nInf)	95.15%
p(+)	07.47%
p(inf|+)	63.05%

2 Ridge Regression

Ridge regression is used for parameter estimation to address the collinearity problem in multiple linear regression. (McDonald, 2009) The Ridge Regression adds the quadratic regularization term $\frac{\lambda}{2}\left(\boldsymbol{\omega}^{\mathrm{T}}\boldsymbol{\omega}\right)$ to the objective $\mathbf{J}_{LS}$. (Rueckert, 2022)

2.1 Derivation of the Least Squares Solution

For the derivation of the least squares I defined the vectors $\mathbf{y} \in \mathbb{R}$, $\mathbf{A} \in \mathbb{R}^{nxM}$ and $\omega \in \mathbb{R}^{M}$ where M is the dimension and n the number of samples.

$$\frac{\partial J_{LS}}{\partial \boldsymbol{\omega}} = \frac{\partial}{\partial \boldsymbol{\omega}}\{1/2\sigma^{-2}(\mathbf{y}-\mathbf{A}\boldsymbol{\omega})^{T}(\mathbf{y}-\mathbf{A}\boldsymbol{\omega})\}$$

After the partial deviation we receive the following equation.

$$\frac{\partial J_{LS}}{\partial \boldsymbol{\omega}} = 1/2\sigma^{-2}(-2\mathbf{y}^{T}\mathbf{A} + 2\boldsymbol{\omega}^{T}\mathbf{A}^{T}\mathbf{A})$$

I set this equation to zero to calculate the least square solution.

$$\begin{gathered}
\frac{\partial J_{LS}}{\partial \boldsymbol{\omega}} = 0, \\
1/2\sigma^{-2}(-2\mathbf{y}^{T}\mathbf{A} + 2\boldsymbol{\omega}^{T}\mathbf{A}^{T}\mathbf{A}) = 0, \\
1/2\sigma^{-2}(-2\mathbf{y}^{T}\mathbf{A} + 2\boldsymbol{\omega}^{T}\mathbf{A}^{T}\mathbf{A}) = 0, \\
-\mathbf{y}^{T}\mathbf{A} + \boldsymbol{\omega}^{T}\mathbf{A}^{T}\mathbf{A}\boldsymbol{w} = 0, \\
\omega = (\mathbf{A}^{T}\mathbf{A})^{-1}\mathbf{A}^{T}\mathbf{y}.
\end{gathered}$$

(Rueckert, 2022) Important here is, that the matrix **A** has a full rank and is invertible. If this is not the case I would use the Moore–Penrose inverse which is described in the following formula.

$$\mathbf{A}^{+} = (\mathbf{A}^{T}\mathbf{A})^{-1}\mathbf{A}T.$$

(Rueckert, 2022)

3 Implementation of Ridge Regression

The last task was to implement the ridge regression for a given dataset. The dataset includes longitude and latitude of a map with the corresponding temperature data.

3.1 Import Data and Implementation

For the implementation I first created the abstract class *Regression* which includes the abstract methods *importData*, *generateTrainingSubset*, *computeLinearRidgeRegression*, *testModel*, *computeError*, *plotError*, *plotHeatMap*, *computMeanOfError* Then i created the class *RidgeRegression* which implements those methods and has the parameter *trainStep* which indicates the size of the training data. For importing the data I used the scipy.io library which is able to read *MatLab* files and load this data. I used only the first dataset of the time series data to create the model. The method *generateTrainingSubset* creates the training data with the parameter *trainStep*. The parameter *trainStep* defines the size of the training set. For example, if the value is set to 4 every 4. values is used for the calculation of the weight values.

3.2 Ridge Regression

For the Implementation of the Ridge Regression calculation I added the method *computeLinearRidgeRegression* and made the calculation. I followed the formula from chapter 2 and used the *numpy* library to do the matrix calculations. I added a helper method which I used to create the feature vector for the calculation. The method is name *createFeatureVector* and takes the parameter *x*. The parameter *x* is the vector of the current y-value. In our case it's a two dimensional vector with the longitude and latitude. I augmented this vector and created the new vector *featureVector* which is a three dimensional vector. The first dimension contains a 1, the second the latitude and the third the longitude. This vector was then returned from the method. After the creation of the feature vectors, the method *computeLinearRidgeRegression* finishes it's calculations and returns the weight vector. The weight vector is used in the method *testModel* where the vector is multiplied with the given test values in the dataset.

3.3 Calculate Error

After testing the model I calculated the error between the model and the provided test data. For the error computed I added the method compute error, which takes a list of y values, stored in the variable *yStar*, to compute the difference between the calculated result and the parameter *yStar*. Next, I sorted the values descending for plotting and created a *panda data frame* with the *pandas* library. Last, I computed also the mean of the error values with *numpy*.

3.4 Plotting

For plotting I used the *seaborn* library for the heatmap and the descending bar chart and *matplotlib* for the error differences between different lambdas and train steps. Both libraries, *seaborn* and *matplotlib* are working great together, because *seaborn* is build on top of *matplotlib*.Because of the native support of *pandas* with the *seaborn* I used the data frame type for preparing the data. For the descending error plot I just created a data frame and passed the value to the function *barplot* in the *seaborn* library. The heat map was a little bit more difficult to create. I created a data frame and used the *pivot* function and sorted the values descending in respect of the longitude value. The sort operation was used, because of the ascending longitude values in the dataset. The prepared data frame was passed to the *heatmap* function of the *seaborn* library.

3.5 Results

The error of the model between the is displayed in Figure 1. This figure shows a descending error plot with a λ from 0.1. Raising λ does not made any changes in my model. Even raising the value to 50, displayed in Figure 2, only changes the value slightly. The train step was at 4 with both plots. The small changes of the error with a different λ is also displayed in Figure 3 and 4. The highest error is exactly on the same position in the heat map, only the value itself changed slightly. After I made some tests with the different λ values

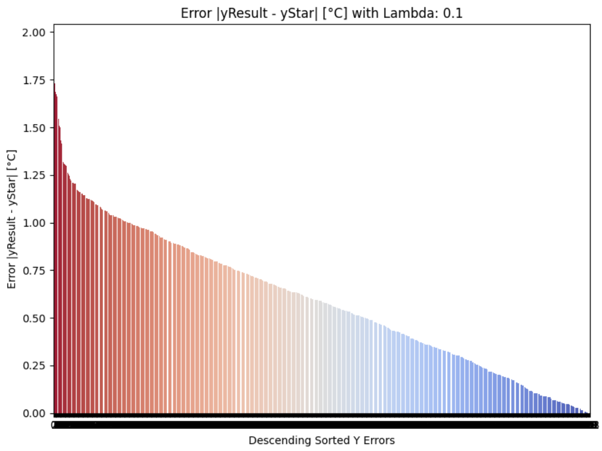

Figure 1: *Descending error plot with λ:0.1*

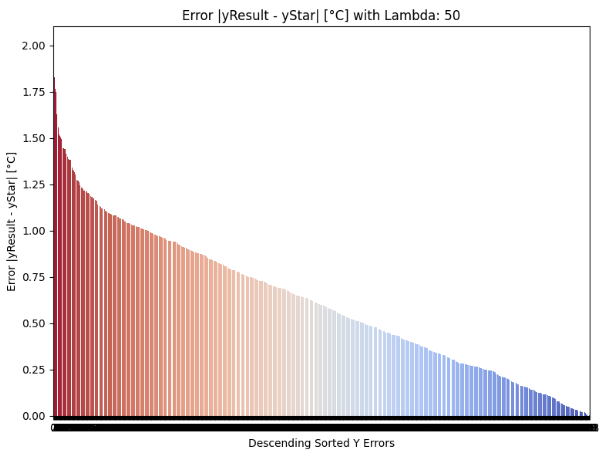

Figure 2: *Descending error plot with λ:50*

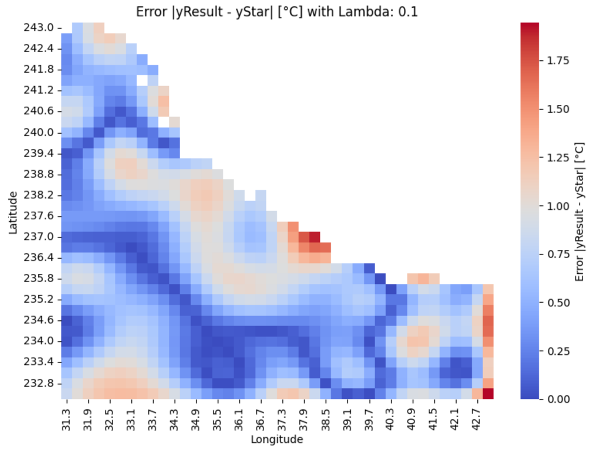

Figure 3: *Heatmap with λ:0.1*

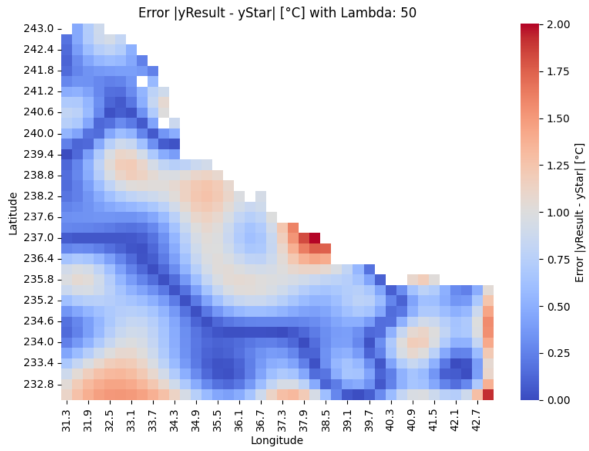

Figure 4: *Heatmap with λ:50*

I also started to adjust the train step. I reduced the train step to 1 and repeated the tests with different λ values. I displayed the results in Figure 5. Figure 5 shows, that the changes of the train step and λ values only made small adjustment to the results itself. Also the mean values of every tests has only small changes in the value itself. The result of the mean values is displayed in Table 3.

3.6 Conclusion

I had some difficulties with the Implementation of the ridge regression. Especially the augmentation of the values gave me some headache. I even tried to use a *Gaussian Basis Function* with the *Polynomial Basis Function* to create a feature vector. However, all the tries didn't result in a better model. Furthermore, the results were much worse than the straight forward implementation of the feature vector with 1, Latitude and Longitude. One explanation of the error could be the small size of the data set. A larger data set could improve the accuracy of the model. The code of the implementation is in the appendix of this paper. Small side node, I had to remove the correct temperature unit from the code because it resulted in an *UTF-8* issue.

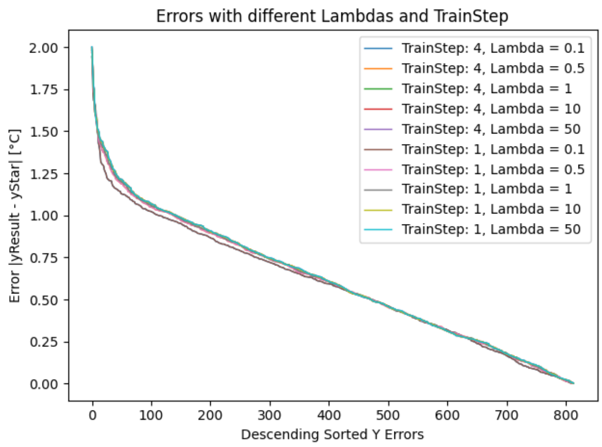

Figure 5: *Descending error plot with different lambdas and train steps*

Bibliography

Joyce, James (2021). "Bayes' Theorem". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Fall 2021. Metaphysics Research Lab, Stanford University.

McDonald, Gary C. (2009). "Ridge regression". In: *WIREs Computational Statistics* 1.1, pp. 93–100. DOI: https://doi.org/10.1002/wics.14. eprint: https://wires.onlinelibrary.wiley.com/doi/pdf/10.1002/wics.14. URL: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/wics.14.

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

Table 3: *Mean Errors*

Train Step	λ	Mean
4	0.1	0.5938
4	0.5	0.6117
4	1.0	0.6156
4	10.0	0.6195
4	50.0	0.6198
1	0.1	0.5938
1	0.5	0.6117
1	1.0	0.6156
1	10.0	0.6195
1	50.0	0.6198

APPENDIX

```python
import numpy as np

populationAustria = 9095538
activeCases = 441098
covTestSensitivity = 0.971
covTestSpecific = 0.995

#A...[infected, not-infected]
#B...[positive, negative]

pInfected = activeCases / populationAustria
pNotInfected = 1 - pInfected

pPositiveInfected = covTestSensitivity
pNegativeNotInfected = covTestSensitivity

pInfectedAndPositive = pInfected * pPositiveInfected
pInfectedAndNegative = pInfected - pInfectedAndPositive

pNotInfectedAndNegative = pNotInfected * pNegativeNotInfected
pNegative = pInfectedAndNegative + pNotInfectedAndNegative

pNotInfectedAndPositive = pNotInfected - pNotInfectedAndNegative
pPositive = pInfectedAndPositive + pNotInfectedAndPositive

pNegativeInfected = pInfectedAndNegative / pNegative
pPositiveNotInfected = pNotInfectedAndPositive / pNotInfected

#BayersTheorem
pInfectedPositive = pPositiveInfected * pInfected / pPositive

print("p(-|inf): " + str(pNegativeInfected))
print("p(+|nInf): " + str(pPositiveNotInfected))
print("p(inf): " + str(pInfected))
print("p(nInf): " + str(pNotInfected))
print("p(+): " + str(pPositive))
print("p(inf|+): " + str(pInfectedPositive))
```

```python
import sys
import seaborn as sbr
import matplotlib.pyplot as plt

sys.path.insert(0, '../modules')
from RidgeRegression import RidgeRegression

def doTest(lambdaValue, trainStep, plot=False):
    ridgeRegression = RidgeRegression(100)
    print("import data")
    ridgeRegression.importData()
    print("generate Trainingset")
    ridgeRegression.generateTrainingSubset()
    print("Train")
    weightVector = ridgeRegression.computeLinearRidgeRegression(lambdaValue)
    ridgeRegression.testModel(weightVector)
    print("Calculate Error")
    yTest = ridgeRegression.getYTestData()
    ridgeRegression.computeError(yTest)
    ridgeRegression.computMeanOfError()
    print("Mean Error with Lambda " + str(lambdaValue) + ": " + str(
        ridgeRegression.getMeanError()))

    if plot:
        print("plot error")
        ridgeRegression.plotError()
        print("plot heatmap")
        ridgeRegression.plotHeatMap()

    return ridgeRegression.getDescSortedError()

firstTrainSetErrors = []

trainStep = 4
firstTrainSetErrors.append((0.1, trainStep,(doTest(0.1, trainStep, plot=True))))
firstTrainSetErrors.append((0.5, trainStep, (doTest(0.5, trainStep))))
firstTrainSetErrors.append((1, trainStep, (doTest(1, trainStep))))
firstTrainSetErrors.append((10, trainStep, (doTest(10, trainStep))))
firstTrainSetErrors.append((50, trainStep, (doTest(50, trainStep, plot=True))))

trainStep = 1
firstTrainSetErrors.append((0.1, trainStep,(doTest(0.1, trainStep))))
firstTrainSetErrors.append((0.5, trainStep, (doTest(0.5, trainStep))))
firstTrainSetErrors.append((1, trainStep, (doTest(1, trainStep))))
firstTrainSetErrors.append((10, trainStep, (doTest(10, trainStep))))
firstTrainSetErrors.append((50, trainStep, (doTest(50, trainStep))))

for testSet in firstTrainSetErrors:
    lambdaValue = testSet[0]
    trainStep = testSet[1]
    errors = testSet[2]
    x = errors["values"]
    y = errors["error"]
    plt.plot(x, y, linewidth=1, label=f"TrainStep: {trainStep}, Lambda = {
        lambdaValue}")

plt.title("Errors with different Lambdas and TrainStep")
plt.xlabel("Descending Sorted Y Errors")
plt.ylabel("Error |yResult - yStar| [C]")
```

```
plt.tight_layout()
plt.legend()
plt.show()
```

```python
import matplotlib.pyplot
import scipy.io as sio
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import seaborn as sbr
import pandas as pd

from inference import Regression
from GaussDistribution import GaussDistribution

class RidgeRegression(Regression):

    def __init__(self, trainStep):
        self.trainStep = trainStep
        self.hasGenerated1DTestData = False

    def importData(self):
        dataDictonary =  sio.loadmat("AssignmentIV_data_set.mat")

        tempTimeData = dataDictonary.get("TempField")
        self.tempData = np.array(tempTimeData[:,:, 0])

        longData = dataDictonary.get("LongitudeScale") #x-value: Long
        latData = dataDictonary.get("LatitudeScale") #y-value: Lat

        self.x_test = np.array(dataDictonary.get("x_test")) # Testdata from DataSet
        self.y_test = np.array(dataDictonary.get("y_test")) # Testdata from DataSet

        longLatData = []
        arrayValues = []

        for y in range(len(self.tempData)):
            for x in range(len(self.tempData[y])):
                value = self.tempData[y,x]
                if np.isnan(value):
                    continue

                arrayValues.append(value)
                longLatData.append((latData[y][0], longData[x][0])) #[0] needed because of strange import of long/latData

        self.inputValues = np.array(longLatData)
        self.outputValues = np.array(arrayValues)
        self.numberOfSamples = len(self.inputValues)

    def generateTrainingSubset(self):
        self.trainSubsetInput = np.array(self.inputValues[0:len(self.inputValues):self.trainStep])
        self.trainSubsetOutput = np.array(self.outputValues[0:len(self.outputValues):self.trainStep])

    def createFeatureVector(self, x):
        featureVector = []
        featureVector.append(1)

        for i in range(len(x)):
```

```
            newXVector = x[i]
            featureVector.append(newXVector)

        return featureVector

    def computeLinearRidgeRegression(self, lambdaValue):
        self.lambdaValue = lambdaValue
        X = np.vstack(([self.createFeatureVector(x) for x in self.trainSubsetInput]))
        Y = np.vstack(([y for y in self.trainSubsetOutput]))

        XT = np.transpose(X)
        XTX = np.matmul(XT, X) + self.lambdaValue * np.identity(X.shape[1])
        self.weightVector = np.matmul(np.matmul(np.linalg.inv(XTX), XT), Y)
        return self.weightVector

    def testModel(self, weight):
        self.yResult = np.hstack([x @ self.weightVector for x in self.x_test])

    def getYTestData(self):
        return self.y_test

    def computeError(self, yStar):
        self.yError = np.transpose(abs(self.yResult - yStar))
        reversedArray = np.flip(np.sort(self.yError, 0))
        self.errorDataFrame = pd.DataFrame({
            'values': range(len(reversedArray)),
            'error': [error[0] for error in reversedArray]
        })

    def getDescSortedError(self):
        return self.errorDataFrame

    def computMeanOfError(self):
        self.meanError = np.mean(self.yError)

    def getMeanError(self):
        return self.meanError

    def plotError(self):
        plt.figure(figsize=(8, 6))
        errorPlot = sbr.barplot(data=self.errorDataFrame, x="values", y="error", palette="coolwarm_r")

        plt.xlabel("Descending Sorted Y Errors")
        plt.ylabel("Error |yResult - yStar| [C]")
        plt.title("Error |yResult - yStar| [C] with Lambda: " + str(self.lambdaValue))
        plt.tight_layout()
        matplotlib.pyplot.show()

    def plotHeatMap(self):
        tempErrorData = \
            {
                'Lat': [round(long,2) for long in self.x_test[:, 1]],
                'Long': [round(lat,2) for lat in self.x_test[:, 2]],
                'error':[error[0] for error in self.yError]
            }

```

```python
        tempDataFrame = pd.DataFrame(tempErrorData)
        tempDataFrame = tempDataFrame.pivot("Long", "Lat", "error")
        reversedTempErrorData = tempDataFrame.sort_values(("Long"), ascending=False)

        def fmt(x, y):
            return '{:,.2f}'.format(x)

        plt.figure(figsize=(8,6))
        errorHeatMap = sbr.heatmap(reversedTempErrorData, vmin=0.0, cmap="coolwarm", cbar_kws={"label":"Error |yResult - yStar| [C]"})
        ax = errorHeatMap.axes

        plt.xlabel("Longitude")
        plt.ylabel("Latitude")
        plt.title("Error |yResult - yStar| [C] with Lambda: " + str(self.lambdaValue))
        plt.tight_layout()
        matplotlib.pyplot.show()
```