FYS-STK4155: Project 2

Simen Nyhus Bastnes

10. October 2020

Abstract

1 Introduction

currently entire document just copied from project 1, so fix this later

Logistic regression with softmax check aurelion geron page 195ish

2 Theory

2.1 Linear regression

Linear regression is a method of fitting a set of p *predictors* $\boldsymbol{X}$ to a data set $\boldsymbol{y}$, while minimizing the error between the *response* $\tilde{\boldsymbol{y}}$ and the actual data $\boldsymbol{y}$. For each of the n samples y_i in the data set the relationship between the response and the predictors $\boldsymbol{X}_i$ is modeled in a linear fashion, giving us the following matrix equation

$$\mathbf{y} = \mathbf{X}\beta + \boldsymbol{\epsilon}$$

where $\beta = (\beta_0, \beta_1, ..., \beta_{p-1})^\mathsf{T}$ are the regression parameters we are trying to estimate, one for each predictor, and $\boldsymbol{\epsilon}$ is the error in our approximation. The matrix $\mathbf{X}$ is often called the design matrix, and the equation can be written a bit more explicitly as

$$y_i = \beta_0 + X_{i,1}\beta_1 + ... + X_{i,p-1}\beta_{p-1} + \epsilon_i$$

Exactly what each predictor is can vary a lot from case to case, and how the design matrix is set up is important for the accuracy of the fit. In our case, we will focus on a form of linear regression where the predictors is on the form of a polynomial in the input parameters. In the case where we have a data set $\boldsymbol{y}(\boldsymbol{x})$, the design matrix can for example be written on the form of

$$\mathbf{X} = (\boldsymbol{x}^0, \boldsymbol{x}^1, ..., \boldsymbol{x}^{p-1})$$

With that said, we still need some way to find the β's that fit the data best, and we will now look at three ways to try to do this.

2.1.1 Ordinary least squares

Following Chapter 2.3 of Hastie et al. [1], in order to find the optimal regression parameters β, the OLS method minimizes the residual sum of squares

$$\text{RSS}(\beta) = \sum_{i=1}^{N} (y_i - x_i^T \beta)^2$$

With $\boldsymbol{y}$ as the vector containing all N y_i, and X an $N \times p$ matrix as shown in section 2.1, this can be written as

$$\text{RSS}(\lambda) = (\boldsymbol{y} - \mathbf{X}\beta)^\mathsf{T}(\boldsymbol{y} - \mathbf{X}\beta)$$

Differentiating with respect to β we get

$$\frac{\partial \text{RSS}}{\partial \beta} = X^\mathsf{T}(\boldsymbol{y} - \mathbf{X}\beta)$$

In order to find an optimal β, this has to be zero

$$\begin{aligned} \mathbf{X}^\mathsf{T}(\boldsymbol{y} - \mathbf{X}\beta) &= 0 \\ \mathbf{X}^\mathsf{T}\mathbf{X}\beta &= \mathbf{X}^T\boldsymbol{y} \end{aligned}$$

Finally giving us the expression for the optimal regression parameters

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

assuming that $\mathbf{X}^\mathsf{T}\mathbf{X}$ is invertible.

2.1.2 Ridge regression

Ridge regression is an example of a so-called shrinkage method, which shrinks the regression coefficients by adding a small penalty proportional to their size.

$$\beta^{\text{Ridge}} = \min_{\beta \in \mathbb{R}^p} \frac{1}{n} ||\boldsymbol{X}\beta - \boldsymbol{y}||_2^2 + \lambda ||\beta||_2^2$$

where λ is a regularization parameter that controls the amount of shrinkage, and we $||\beta||_2^2 \leq t$ where t is a finite number larger than zero. The higher λ, the more shrinkage occurs. This can be shown to give the Ridge solution

$$\beta^{\text{Ridge}} = (\mathbf{X}^\mathsf{T}\mathbf{X} + \lambda I)^{-1}\mathbf{X}^\mathsf{T}\boldsymbol{y}$$

The aim with Ridge regression is to limit the potential problems with singularities when computing the inverse of $\mathbf{X}^\mathsf{T}\mathbf{X}$, which can be a problem when there are many correlated variables.

2.1.3 Lasso regression

Lasso regression is another shrinkage method, with a slightly different optimization equation compared to Ridge regression

$$\beta^{\text{Lasso}} = \min_{\beta \in \mathbb{R}^p} \frac{1}{n} ||\boldsymbol{X}\beta - \boldsymbol{y}||_2^2 + \lambda ||\beta||_1$$

where $||\beta||_1$ is the L_1 norm.

2.2 Bias-Variance decomposition

Starting from the so-called cost function, we want to derive an expression for the variance and bias for the predicted values $\tilde{\boldsymbol{y}}$

$$C(\mathbf{X}, \beta) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 = \mathbb{E}[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2]$$

assuming that the true data is generated from a noisy model $\boldsymbol{y} = f(\boldsymbol{x}) + \boldsymbol{\epsilon}$ where ϵ is normally distributed with zero mean and standard deviation σ^2, we get

$$\mathbb{E}[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2] = \mathbb{E}\Big[(f + \boldsymbol{\epsilon} - \tilde{\boldsymbol{y}})^2\Big]$$

We add and subtract $\mathbb{E}[\tilde{\boldsymbol{y}}]$ to the expression

$$\begin{aligned}
&= \mathbb{E}\Big[(f + \boldsymbol{\epsilon} - \tilde{\boldsymbol{y}} + \mathbb{E}[\tilde{\boldsymbol{y}}] - \mathbb{E}[\tilde{\boldsymbol{y}}])^2\Big] \\
&= \mathbb{E}\Big[(f - \mathbb{E}[\tilde{\boldsymbol{y}}]) - (\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}}]) + \boldsymbol{\epsilon})^2\Big]
\end{aligned}$$

Expanding the expression gives us

$$\begin{aligned}
&= \mathbb{E}\Big[(f - \mathbb{E}[\tilde{\boldsymbol{y}}])^2 + (\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}}])^2 + \boldsymbol{\epsilon}^2 - 2(f - \mathbb{E}[\tilde{\boldsymbol{y}}])(\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}}]) \\
&+ 2(f - \mathbb{E}[\tilde{\boldsymbol{y}}])\boldsymbol{\epsilon} - 2(\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}}])\boldsymbol{\epsilon}\Big]
\end{aligned}$$

We can use that the mean of $\boldsymbol{\epsilon}$ is zero, thus making the last two terms vanish $\mathbb{E}[\boldsymbol{\epsilon}] = 0$

$$= \mathbb{E}\Big[(f - \mathbb{E}[\tilde{\boldsymbol{y}}])^2 + (\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}}])^2 + \boldsymbol{\epsilon}^2 - 2(f - \mathbb{E}[\tilde{\boldsymbol{y}}])(\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}}])\Big]$$

For this to give the results we want, the last term should also be zero, but I don't see how, and often the derivation is just skipped over, or with confusing notation. Assuming it is zero, we get

$$\mathbb{E}[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2] = \mathbb{E}[(f - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] + \mathbb{E}[(\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] + \mathbb{E}[\boldsymbol{\epsilon}^2]$$

Using that $\mathbb{E}[\boldsymbol{\epsilon}^2] = \sigma^2$, we get the final expression for the bias-variance decomposition

$$\mathbb{E}[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2] = \mathbb{E}[(f - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] + \mathbb{E}[(\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}}])^2] + \sigma^2$$

Where the left-hand side is the so-called mean squared error (henceforth referred to as MSE), the first term is the bias term, which measures how much the our model $\tilde{\boldsymbol{y}}$ differs from the true model $f(\mathbf{x})$. The second term is the variance of our model, while the third term is the noise. High bias indicates underfitting, and the model is not complex enough to capture the relevant features, while high variance can indicate that the model is overfitting, and adjusting to the smallest variations in the data set.

Thus, our goal in this project is to try to find the middle ground between high bias and high variance, and that will be the point where the MSE is the lowest, and thus it makes a lot of sense to use it as a measure for how good our model is.

The MSE can be written more explicitly as

$$\text{MSE}(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2$$

Another way of assessing the results would be the R^2 score

$$R^2(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \bar{\boldsymbol{y}})^2}$$

where

$$\bar{\boldsymbol{y}} = \frac{1}{n}\sum_{i=0}^{n-1} y_i$$

2.3 Confidence intervals

For the OLS and Ridge regression cases, it is possible to derive the variance of β (a proper derivation is given in [2]), and thus the confidence intervals as well. For the OLS method, the variance is given by

$$\text{Var}(\beta) = \sigma^2(\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}$$

where σ^2 is the estimated variance of y given by

$$\sigma^2 = \frac{1}{N-p-1}\sigma_{i=1}^{N}(y_i - \tilde{y}_i)^2$$

Taking the square root of the diagonal of $(\mathbf{X}^\mathsf{T}\mathbf{X})^{-1}$ gives us an estimate of the variance of the j-th regression coefficient

$$\sigma^2(\beta_j) = \sigma^2\sqrt{[\mathbf{X}^\mathsf{T}\mathbf{X}]^{-1}_{jj}}$$

Letting us construct the 95% confidence intervals by

$$\text{CI}(\beta_j) = \left[\beta_j - 2\sqrt{\sigma_2(\beta_j)}\,,\ \ \beta_j + 2\sqrt{\sigma_2(\beta_j)}\right]$$

Similarly, the variance for β for Ridge regression can be found to be

$$\text{Var}[\beta^{\text{Ridge}}] = \sigma^2[\mathbf{X}^\mathsf{T}\mathbf{X} + \lambda\mathbf{I}]^{-1}\mathbf{X}^\mathsf{T}\mathbf{X}[(\mathbf{X}^\mathsf{T}\mathbf{X} + \lambda\mathbf{I})^{-1}]^\mathsf{T}$$

and confidence interval can be constructed following the same steps as done above for OLS.

2.4 Resampling methods

In order to assess our models properly, we will be using some form of resampling. Specifically, we will be looking at the Bootstrap, and the standard k-fold Cross-validation resampling methods.

2.4.1 Bootstrap

The basic concept of the Bootstrap resampling method is to create new data sets by drawing samples (with replacement) from the training data set. Algorithm 1 shows the Bootstrap method given a data set $\mathcal{L}$ consisting of the data $\mathbf{X}_\mathcal{L} = \{(y_j, \boldsymbol{x}_j), j = 0 \dots n-1\}$

Algorithm 1 Bootstrap

1: Split the data set $\mathbf{X}_\mathcal{L}$ into training $\mathbf{X}_{\mathcal{L},\text{train}}$ and test data sets $\mathbf{X}_{\mathcal{L},\text{test}}$
2: **for** $i = 0$, $N_\text{bs} - 1$ **do**
3: Create a new data set $\mathbf{X}_{\mathcal{L},i}$ by drawing samples with replacement from $\mathbf{X}_{\mathcal{L},\text{train}}$.
4: Fit the model using $\mathbf{X}_{\mathcal{L},i}$.
5: Evaluate the model on the test set $\mathbf{X}_{\mathcal{L},\text{test}}$ and store the results.
6: **end for**
7: Assess the model by looking at the distribution of computed quantities, for example looking at the mean of the MSE.

2.4.2 Cross-validation

Cross-validation is a resampling method where (in the case of k-fold CV) the data set is split into k-folds of training and test data sets. Algorithm 2 shows the standard k-fold cross-validation.

Algorithm 2 k-fold Cross-Validation

1: Shuffle the data set $\mathbf{X}_\mathcal{L}$ randomly.
2: Split the data set $\mathbf{X}_\mathcal{L}$ into k folds/subsets $\{\mathbf{X}_{\mathcal{L},i}, i = 0 \dots k-1\}$.
3: **for** $i = 0$, $k - 1$ **do**
4: Set $\mathbf{X}_{\mathcal{L},i}$ as the test data set and the rest of the folds as the training set.
5: Fit the model using the training data set as defined above.
6: Evaluate the model on the i-th test set $\mathbf{X}_{\mathcal{L},i}$ and store the results.
7: **end for**
8: Assess the model by looking at the distribution of computed quantities.

3 Code and Data sets

figure out naming

3.1 Code

All scripts used to generate the results in this project can be found in the Github repository [3]. **DONT THINK WE NEED MUCH MORE THAN THIS, BUT CHECK FEEDBACK + HOW WE DID IN COMPPHYS**

The code can be found in the `src` folder, while figures can be found in the `figures` folder. The data files for the terrain map as well as calculated results can be found in `datafiles`. The `benchmark` folder contains a data file for each section of the code with information about the parameters used, in order to easier verify that the program works.

The main program `main.py` is split into several different run modes, where the specific details of each one is detailed in table 1. When starting the program, it prompts you to input one of

the following arguments (a,b,c,d,e,f,g), where (a-e) relates to the Franke function, and (f,g) to the terrain data.

Table 1: Explanation of each of the different run modes for the main program `main.py`.

Run mode	Function
a	Performs an OLS regression on the Franke function with a constant degree p. Prints out the MSE and R^2 score, and plots the variance of the regression coefficients β
b	Performs OLS with Bootstrap and CV on the Franke function. Saves to file the error, bias, and variance for each p, and plots all relevant plots.
c	Performs a Cross-validation and compares it to the one in Scikit-learn.
d	Runs Ridge regression with Bootstrap and CV for a set of λ values over a set of p values. Saves the results to file, and plots the results.
e	Same as c, expect done with Lasso regression
f	Creates plots of the patch we are looking at for the terrain map
g	Depending on what is set to the `reg_str` variable, performs one of the three regression methods with Bootstrap and CV on the terrain data. Results are saved to file and plotted.

The program first defines common variables needed for all the different parts of the program, and then goes into specific sections of the code relating to either the Franke function or the terrain data in order to initialize the data set that is to be used.

Then, there is a fairly long function that performs the actual regression over the polynomial degree p and hyperparameter λ for all parts excluding `a` and `f`. This function performs Bootstrap and CV with the regression method specified by the variable `reg_str`, returning a huge amount of arrays with all the results. This function is a result of restructuring in order to minimize the amount of duplicate code (where the regression method used is the major difference), in order to easier find and fix some bugs that were present in the code.

After that, each run mode has its own section of code to call the regression function, and plots/saves the results.

3.2 Data sets

add something referring to project 1. In this project, we will be using two different data sets. The first, is the Franke function shown in appendix A.1. This function is a two-dimensional function that has been widely used for testing implementations for regression and interpolation.

MNIST, show example plot of some of the 8x8 images with labels

4 Results and discussion

WE USE CROSS-VALIDATION FOR ALL RESULTS SINCE PROJECT 1 SHOWED THAT IT PERFORMED WELL, AND SINCE IT IS CHEAPER THAN BOOTSTRAP (since less resamples), IT ALLOWS US TO TEST FOR MORE COMBINATIONS OF HYPERPARAMETERS

My understanding is that bootstrapping is a way to quantify the uncertainty in your model while cross validation is used for model selection and measuring predictive accuracy. FROM STATSEXHANGE, check link in sublime. k-fold cross validation is used for two main purposes, to tune hyper parameters and to better evaluate the performance of a model.

Hyperparameter search is definitely something that would benefit a lot from parallelizing.

In this section, we will be showing some select results from running the code described in Chapter 3. More figures and data files, as well as some simpler benchmark runs can be found within their respective folders in the Github repository [3]. The results will be split into two sections, the first one pertaining to the Franke function, while the second is the analysis of the terrain data.

4.1 Franke function

4.1.1 Stochastic Gradient Descent

4.1.2 Feed-forward Neural Network

test different activation functions

4.2 NOT MNIST, or reduced number recognition

MEOW

for simplicity we will be looking at symmetrical hidden layers, as in with identical amount of nodes. This is not necessary, but studying other variations would almost be a project in itself. (maybe find some sources regarding performance) 10, 20, 30, 40, 50, 60, 70 nodes in each hidden layer, and (1,2,3,4,5) hidden layers should be plenty. **could also have tested with various activation functions throughout, but also a lot of extra work, and would ideally look closer at literature for that in order to see what has already been discovered regarding it**

4.2.1 Classification with Neural Network

4.2.2 Logistic regression

5 Conclusion

In this project, we set out to investigate how different methods of linear regression performs on the Franke function, as well as terrain elevation data. To assess our models, we employed Bootstrap and Cross-validation as resampling methods.

For the Franke function, we found that all three methods, OLS, Ridge and Lasso performed very similarly, giving errors on the scale 10^{-3} for the best-fit model. Ridge was however the best model, both in accuracy and stability. Lasso yielded almost as good results as Ridge, but was significantly slower as it has no analytical solution. Cross-validation gave the lowest errors, with roughly 30% improvement over Bootstrapping for the best-fit models.

For the terrain data, we found that the OLS performed the best, with Ridge being a decent

bit behind, and Lasso roughly as much behind Ridge. The best fit for OLS was at $p = 20$, which gave an MSE of $9.58 \cdot 10^{-5}$. In this case, Cross-validation was only marginally better for OLS and Lasso regression, and slightly worse for Ridge, but the difference is minor. As all the best fits was the highest p-value we tested, testing higher polynomial degrees would be necessary.

For future work, looking closer at the terrain data would probably be a good idea. Ideally, more parts of the maps should be checked instead of just one small patch, in order to see how more complex and noisy regions affect the results, as well as increasing the degrees that are tested (and the hyperparameter λ). This however would take significantly more time, especially for Lasso, so looking into parallelizing the analysis would be relevant. Since all p and λ computations are independent, doing so should not be too difficult.

References

[1] Hastie et al. *The Elements of Statistical Learning - Data Mining, Inference, and Prediction.* Springer, second edition edition, 2017.

[2] Wessel N. van Wieringen. Lecture notes on ridge regression, 2020.

[3] Github repository, project 1. `https://github.com/simennb/fysstk4155-project1`.

[4] Richard Franke. A critical comparison of some methods for interpolation of scattered data, 1979.

Appendix

A Data sets

A.1 Franke function

The Franke function as given in [4]

$$
\begin{aligned}
f(x,y) = \frac{3}{4}\exp\left(-\frac{(9x-2)^2}{4} - \frac{(9y-2)^2}{4}\right) + \frac{3}{4}\exp\left(-\frac{(9x+1)^2}{49} - \frac{(9y+1)^2}{10}\right) \\
+ \frac{1}{2}\exp\left(-\frac{(9x-7)^2}{4} - \frac{(9y-3)^2}{4}\right) - \frac{1}{5}\exp\left(-(9x-4)^2 - (9y-7)^2\right)
\end{aligned}
$$

where $x, y \in [0, 1]$.