

# FYS-STK4155 Project 1

Bendik Steinsvåg Dalen & Gabriel Sigurd Cabrera

September 25, 2019

## Abstract

## Introduction

## Data

## Method

### Generalization of Multidimensional Polynomials

If we wish to construct a  $p$ -dimensional polynomial of degree  $d$ , we need to know what terms need to be included to give us a completely generalized polynomial. For a 1-D polynomial of second degree, we would have three terms:

$$f(x) = \beta_1 + \beta_2x + \beta_3x^2$$

Where  $\beta$  is a vector containing each coefficient. For a 2-D polynomial of second degree, we would have 6 terms:

$$f(x, y) = \beta_1 + \beta_2x + \beta_3y + \beta_4xy + \beta_5x^2 + \beta_6y^2$$

And for a 3-D polynomial of second degree, we would have 10 terms:

$$f(x, y, z) = \beta_1 + \beta_2x + \beta_3y + \beta_4z + \beta_5xy + \beta_6xz + \beta_7yz + \beta_8x^2 + \beta_9y^2 + \beta_{10}z^2$$

There are many possible combinations of  $p$  and  $d$ , and the number of terms blows up significantly as these values increase. We can, however, create a general expression [1] for any  $p$  and  $d$  using summation notation:

$$f(\mathbf{x}) = \sum_{\sum_{j=1}^d i_j \leq p} \left( \beta_{i_1, i_2, \dots, i_d} \prod_{k=1}^d x_k^{i_k} \right) \quad (1)$$

Alternatively, a simple `python` script can be used to find all the terms' exponents by calculating all permutations of the natural numbers from zero to  $d$  in sets of length  $p$ , then removing all results whose sum is greater than  $d$ .

```

1  def get_exponents(p,d):
2      powers = np.arange(0, d+1, 1)
3      powers = np.repeat(powers, p)
4      exponents = list(permutations(powers, p))
5      exponents = np.unique(exponents, axis = 0)
6
7      if p != 1:
8          expo_sum = np.sum(exponents, axis = 1)
9          valid_idx = np.where(np.less_equal(expo_sum, d))[0]
10         exponents = np.array(exponents, dtype = np.int64)
11         exponents = exponents[valid_idx]
12     else:
13         exponents = np.array(exponents, dtype = np.int64)
14
15     return len(exponents)

```

## Ordinary Least-Squares (OLS) Regression

We are given a  $p+1$ -dimensional dataset<sup>1</sup> consisting of  $N$  datapoints per feature such that:

$$\mathbf{X} = \begin{bmatrix} X_{1,1} & X_{1,2} & \cdots & X_{1,p} \\ X_{2,1} & X_{2,2} & \cdots & X_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ X_{N,1} & X_{N,2} & \cdots & X_{N,p} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \quad (2)$$

Where the  $N \times p$  matrix  $\mathbf{X}$  contains the dataset's *input data*, and the  $N$ -vector  $\mathbf{y}$  contains its *output data*, such that each row in  $\mathbf{X}$  corresponds to a single output in  $\mathbf{y}$ .

Now, we wish to find a  $p$ -dimensional polynomial of degree  $d$  which most closely matches our dataset. We will need a design matrix  $\mathbf{A}$ ; this will require using the knowledge presented in (1), since a design matrix should contain each polynomial term as an individual column.

Next, we will be using the method of *least squares* [2], whereby we attempt to minimize the *residual sum of squares*:

$$RSS(\beta) = \sum_{i=1}^N (y_i - \mathbf{A}_i^T \beta)^2$$

Where  $\mathbf{A}_i$  represents the  $i^{\text{th}}$  row in  $\mathbf{A}$ , and  $\beta$  is the set of coefficients in the aforementioned polynomial; in matrix form, this can be written more concisely:

$$RSS(\beta) = (\mathbf{y} - \mathbf{A}\beta)^T (\mathbf{y} - \mathbf{A}\beta)$$

To minimize the  $RSS$ , we can differentiate it with respect to  $\beta$  and set the right-hand side equal to zero – this allows us to solve for  $\beta$ , which will give us the coefficients to the polynomial that best matches our dataset:

$$\mathbf{A}^T (\mathbf{y} - \mathbf{A}\beta) = 0 \iff \mathbf{A}^T \mathbf{y} = \mathbf{A}^T \mathbf{A} \beta$$

Solving for  $\beta$  then gives us our desired result:

$$\beta = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{y} \quad (3)$$

---

<sup>1</sup>Meaning a set of  $p$  input features and 1 output.

Using the set of coefficients given by (3), we can then match the dataset from (2) as effectively as possible.

## Ridge Regression

The solution for  $\beta$  given in (3) can be used without issue in many cases, but if the matrix  $\mathbf{A}$  is singular<sup>2</sup>, we run into an issue – namely, we cannot take the inverse of a singular matrix! As a result, we must look to more robust methods; one such method is called *ridge regression*.

The process of obtaining our vector of coefficients  $\beta$  via ridge regression is functionally very similar to that of OLS. The main difference is that we include an extra term in the residual sum of squares:

$$RSS(\beta) = \sum_{i=1}^N (y_i - \mathbf{A}_i^T \beta)^2 + \lambda \sum_{i=1}^N \beta^2$$

In matrix form, this can be rewritten:

$$RSS(\beta) = (\mathbf{y} - \mathbf{A}\beta)^T (\mathbf{y} - \mathbf{A}\beta) + \lambda \beta^T \beta$$

Where  $\lambda$ , known as the *hyperparameter*, is a scalar value. Performing the same process as in the previous subsection, we are left with a solution similar to that in (3):

$$\beta = (\mathbf{A}^T \mathbf{A} + \lambda \mathbf{I})^{-1} \mathbf{A}^T \mathbf{y} \quad (4)$$

In cases where  $\mathbf{A}$  is singular, it is therefore possible to make very few changes to the OLS algorithm and still get a good result, one must simply optimize the hyperparameter and find a  $\lambda$  that minimizes the *mean squared error* (yet to be introduced) of our polynomial approximation.

## LASSO Regression

The *least absolute shrinkage and selection operator*, commonly abbreviated as *LASSO*, is a method that implements the **L1** norm in place of the **L2** (or Euclidian) norm used in ridge regression. The residual sum of squares is therefore given by:

$$RSS(\beta) = \sum_{i=1}^N (y_i - \mathbf{A}_i^T \beta)^2 + \lambda \sum_{i=1}^N |\beta| \quad (5)$$

Unfortunately, differentiating the above with respect to  $\beta$  will not work as intended, since we cannot take the matrix-form derivative of  $\lambda \sum_{i=1}^N |\beta|$ . As a result, we must use an iterative *gradient descent* method to minimize the right-hand side of (5).

---

<sup>2</sup>Meaning that  $\det(\mathbf{A}) = 0$

---

**Algorithm 1** The LASSO algorithm, over the course of 500 iterations.

---

```

1:  $z = \sum_i A_i^2$ 
2:  $i = 0$ 
3: while  $i \leq 500$  do
4:    $i = i + 1$ 
5:    $j = 0$ 
6:   while  $j \leq p$  do
7:      $\hat{y} = \sum_{k \neq j} \beta A_{*,k}$ 
8:      $\rho = \sum_k A_{*,k}(\mathbf{y} - \hat{\mathbf{y}})$ 
9:     if  $\rho < -\lambda/2$  then
10:       $\beta_j = (\rho + \lambda/2)/z_j$ 
11:     else if  $\rho > \lambda/2$  then
12:       $\beta_j = (\rho - \lambda/2)/z_j$ 
13:     else
14:       $\beta_j = 0$ 
15:     end if
16:   end while
17: end while

```

---

## Mean Squared Error

To get a measure of success with respect to the implemented method and parameters, we can calculate the mean difference in the squares of each measured output  $y_i$  and their respective predicted outputs  $\hat{y}_i$ :

$$MSE(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2 \quad (6)$$

The lower the  $MSE$ , the closer the polynomial approximation is to the original dataset. If it is too low, however, we run the risk of overfitting our dataset, which is not desirable either – fortunately, this not an issue within the scope of this report.

## $R^2$ Score

Another measure of success is the *coefficient of determination*, colloquially known as the  $R^2$  score, is given by the following expression:

$$R^2 = 1 - \frac{\sum_{i=1}^N (y_i - \hat{y}_i)^2}{\sum_{i=1}^N (y_i - \bar{y})^2} \quad (7)$$

The closer  $R^2$  is to one, the closer the polynomial approximation is to the input/output dataset, although a perfect score can once again arise due to overfitting just as in the case of the  $MSE$ .

## Results

## Discussion

## Appendix

### References

- [1] M. (<https://math.stackexchange.com/users/58320/macavity>), “What is the general form of a polynomial of degree  $n$  and with  $m$  variables?.” Mathematics Stack Exchange. URL:<https://math.stackexchange.com/q/2482654> (version: 2017-10-21).
- [2] T. Hastie, R. Tibshirani, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2013.