Project 1 Report: Polynomial Curve-Fitting Regression for Working-Age Data

Jordan S-H

https://github.com/jstebner/481-P1

[0] Overview

The aim of this project is to implement all components necessary to perform generalized linear regression on the data found in the '**data/**' directory.

The implementations for all models and transformers can be found in the '**src/impls.py**' file, and the usage of implementations is done in the '**src/main.py**' file which can be run with:

```
$ python3 ./main.py
```

Note that the main file can be run from any location as it can locate the positions of supporting files.

All figures used in this document and data collected from testing the model are also stored in the '**out/**' directory with names indicative of their content.

The model to be tested (in '**src/main.py**') is given as follows:

```
model = OutputScalingWrapper(
    Pipeline(
        StandardScaler(),
        PolynomialFeatures(degree=d),
        StandardScaler(),
        LinearRegressor()
    )
)
```

A rough outline of how the model seeks to perform polynomial curve fitting is given as follows:

1. The target vector is scaled prior to fitting the model (for prediction accuracy).
2. The input vector is scaled and transformed with Polynomial Features.
3. The transformed input vector is scaled to create interpretable weights.
4. A linear regressor is fit using the transformed input and target vectors.
5. When creating predictions, the model applies the weights found when predicting to the new transformed input (using the same transformations as the training data) and un-scales the resulting vector (using the same transformations as the training target vector).

[1] 6-Fold Cross Validation for varying degrees

We'll be using 6-fold cross validation to select an optimal degree (d*) by evaluating the average RMSE† across each fold for each d ∈ [0, 12]. The splits made are done in order (fold 1 validation set uses indices [0, 7), fold 2 uses indices [7, 14), ...) without shuffling the data as was specified in the project description.

The data for each degree and its associated average RMSE can be found in '**out/cv_errors_data.dat**', and the error curve to select d* is given below:

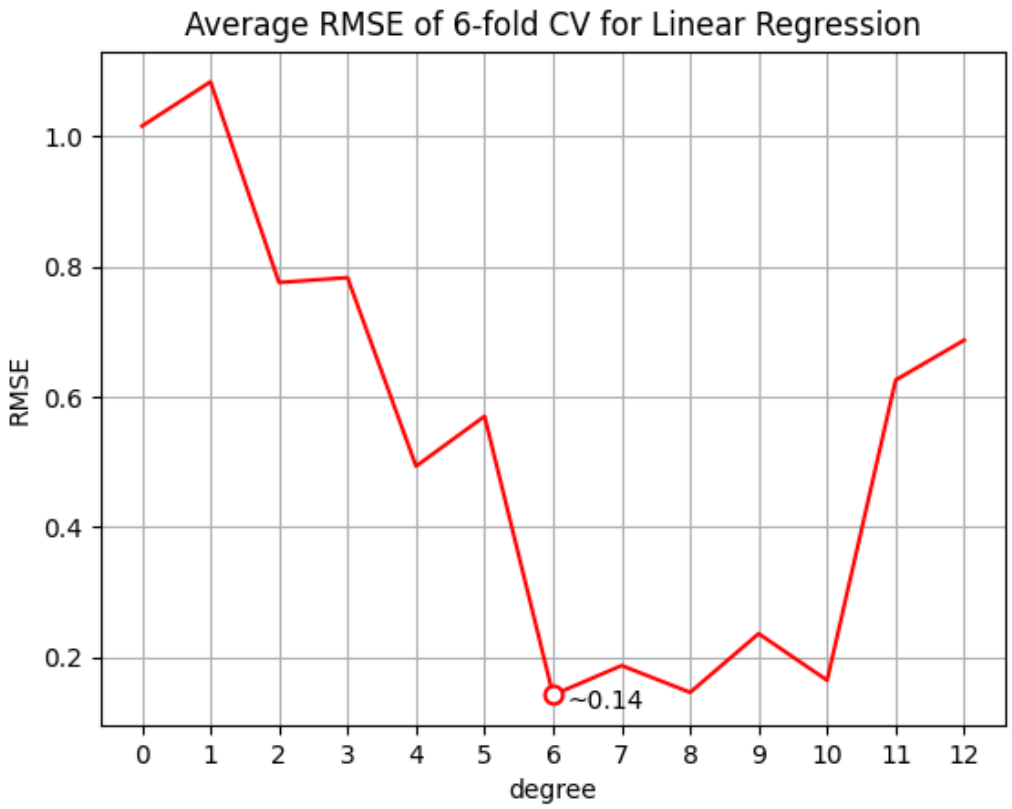

Figure 1.1

[2] Optimal degree of model - d*

Figure 1.1 shows the error decline as model complexity increases, until it begins to rise again at higher degrees (which suggests overfitting). Also shown in the figure is the degree with minimal error, which is degree 6 with an error of roughly 0.14. Degree 6 will be used for d* moving forward.

† See appendix 8.1 for details

[3] Weights obtained using d*

The weights obtained fitting a model with degree 6 on all training data are stored in '**out/d6_weights.dat**', and the formula for predicting output values (with rounded weights, and assuming $x \in R$) is given below:

$$f(x) \approx -0.113 + 0.357x + 3.370x^2 + 0.136x^3 - 2.946x^4 + 0.008x^5 + 0.538x^6$$

Since the training data was scaled after transforming with polynomial features, the generated weights can be compared relative to each other[†]. We can see that the weights for the x^2 and x^4 have a larger effect on the output, while the weight for x^5 has a very small effect.

[4] Scores of the selected model

When evaluating the model with d* on the training and testing datasets, the following RMSE values (which will be referred to as Loss) are produced:

Training Loss: 0.10540106673270466

Testing Loss: 0.11432570919500114

*(These values can also be seen when running '**src/main.py**')*

While the testing error is still greater than the training error (which is to be expected), it's surprisingly close[‡] (which was unexpected).

My interpretation of this result is that the training RMSE is the expected loss for any data consistent with the trend existing in the training data set, and given that the relative error percent of the testing RMSE from the training RMSE is:

$$RE(\%) = \frac{|\text{measured} - \text{expected}|}{\text{expected}} * 100\% = \frac{|\text{TestLoss} - \text{TrainLoss}|}{\text{TrainLoss}} * 100\% \approx \mathbf{8.47}\%$$

One could (and I will) see this value as a **generalization error**, meaning this model was ~8.5% away from the expected loss from the training set (though this view may be naïve). I believe this interpretation is at least somewhat valid, as a test loss lower than the associated training loss could be due to a small test set or luck (though this view may also be naïve).

[†] I am not entirely confident in this statement 😊.

[‡] See addendum 7.1 for details and further analysis.

[5] Prediction Curve

Below is the graph of the model fit using all training data transformed with degree 6 polynomial features superimposed on all data in the project.

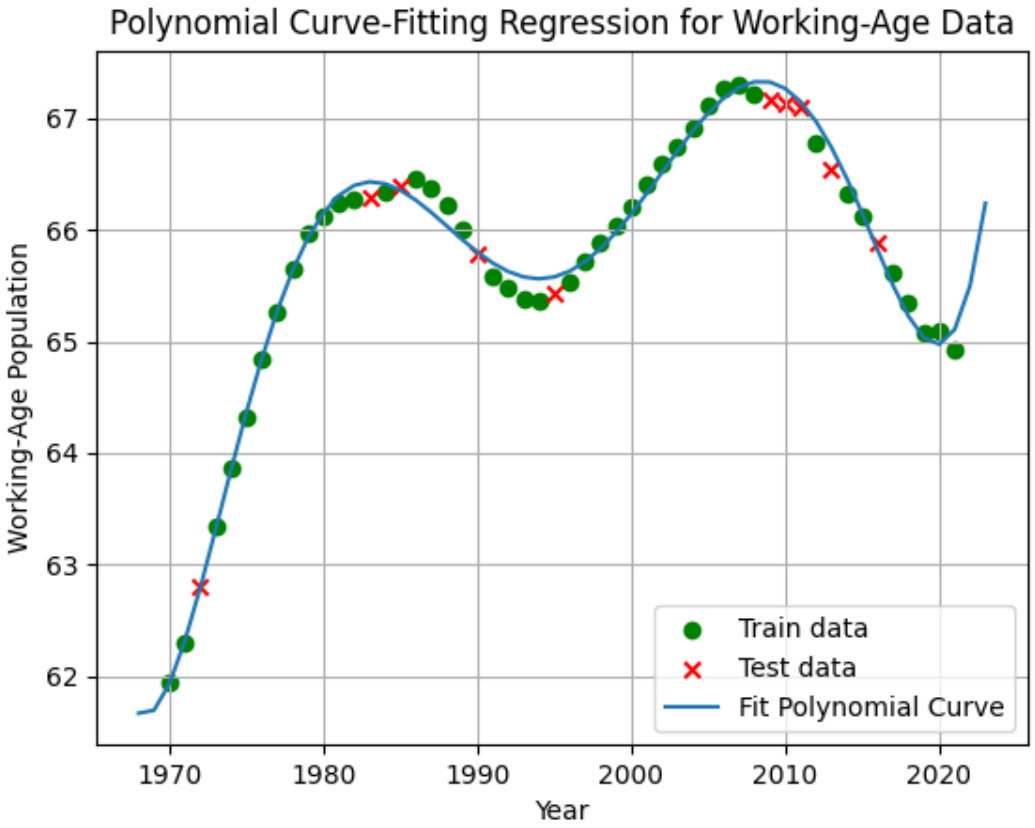

Figure 5.1

We can see in figure 5.1 that the model is able to fit a polynomial curve well to the data. There are places in the curve where it may have benefitted from a slightly higher degree (namely the peaks of the data). This can be supported by figure 1.1 where degree 8 is comparable in accuracy to degree 6 in cross validation, and further exploration could be conducted to test that claim[†].

[†] See addendum 7.2 for details

[6] Reflection

I'm happy with the curve my program was able to fit as seen in figure 5.1. The results discussed in section 4 also indicate that the program was able to generalize (roughly) to the training data with how low and close the training and testing losses were, though this could be due to the size of the dataset. I might want to try this model on a larger dataset to see if my program can perform as well on more/higher dimension data and to see if my implementation of polynomial features works on data with more than one feature[†].

I would also like to try testing a ridge regression model, as linear regression is equivalent to ridge regression parameterized by $\lambda=0$, meaning the results may be improved with different values of λ[‡]. An added hyperparameter to tune also gives an additional "degree of freedom" for testing, which could be fun to explore and create pretty visualizations for analysis.

At this point, the formal report for the project concludes; however, the following sections contain additional exploration of the model discussed (as well as one built on ridge regression) and technical details on motivations for how some aspects of the project are implemented (i.e., I added them for completeness, so you don't need to read them).

[†] See addendum 7.3 for further analysis on datasets, and appendix 8.2 for details on the implementation of polynomial feature creation.

[‡] See addendum 7.4 for details and further analysis.

[7] Addendum

** Code for this section can be found in '**src/addendum.ipynb**'*

7.1 – Test error across degrees

TODO: test err on cv graph

7.2 – Changes in architecture

TODO: remove initial scaling

7.3 – Larger datasets

TODO: this

7.4 – Ridge regression

TODO: this

[8] Appendix

8.1 – RMSE implementation

The formula used in the RMSE implementation for this project is:

$$RMSE(a,b) \equiv \sqrt{\frac{1}{n}}\|a-b\|_2$$

I use this formula instead of calculating RMSE using its definition because the python library used for linear algebra operations is (probably) more optimized than any implementation which iterates over each vector performing the necessary calculations that I could write (and I find the code for this formula more aesthetically pleasing). The proof on why the above formula is true is as follows:

Given that:

$$RMSE(a,b) \coloneqq \sqrt{\frac{1}{n}\sum_{i=1}^{n}(a_i-b_i)^2} \quad \forall\, a,b \in \mathbb{R}^n$$

The n^{-1} term can be moved out of the root to make:

$$= \sqrt{\frac{1}{n}}\sqrt{\sum_{i=1}^{n}(a_i-b_i)^2}$$

(Eq. 1)

Given that:

$$\|c\|_2 \coloneqq \sqrt{\sum_{i=1}^{m}c_i^2} \quad \forall\, c \in \mathbb{R}^m$$

We can add a term to each side to make:

$$\sqrt{\frac{1}{m}}\,\|c\|_2 \coloneqq \sqrt{\frac{1}{m}}\sqrt{\sum_{i=1}^{m}c_i^2}$$

(Eq. 2)

Let c = a – b. This implies that:

$$m = n \quad and \quad c_i = a_i - b_i \quad \forall\, i \in [\,1, n\,] \cap \mathbb{Z}$$

We can use these equalities to substitute terms in equation 2 to make:

$$\sqrt{\frac{1}{n}}\,\|c\|_2 \coloneqq \sqrt{\frac{1}{n}}\sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$$

Which is equivalent to equation 1. We can substitute c for (a – b) to yield:

$$\therefore\ RMSE(a,b) \equiv \sqrt{\frac{1}{n}}\|a-b\|_2 \quad \forall a, b \in \mathbb{R}^n$$ 😊

It's a simple proof, but I felt that it would be a good exercise to work through and wanted to justify my decision for how RMSE was implemented. I haven't had any formal instruction on writing proofs, so please let me know if there are any ways I can improve on it regarding use of notation, logical steps taken, presentation, etc.

8.2 – Polynomial features implementation

The implementation for polynomial features follows the fit/transform pattern used throughout scikit-learn to stay consistent with common practice, so the fit method creates and stores list of exponents for each transformed feature which is used by the transform method to calculate each transformed feature using the input features.

The transform method is relatively straightforward to describe, and can be defined as such:

$$Transform(x, E) \coloneqq [\,\Theta(x, E_1), \Theta(x, E_2), \ldots, \Theta(x, E_t)\,] \quad s.t.\ x \in \mathbb{R}^m, E \in \mathbb{R}^{t \times m}$$

$$\Theta(x, \varepsilon) \coloneqq \prod_{i=1}^{m} x_i^{\varepsilon[i]} \quad s.t.\ x, \varepsilon \in \mathbb{R}^m$$

Where:

- **m** is the number of features in the data,
- **t** is the number of features in the transformed space
- **x** is a m-dimensioned vector representing a data point,
- **E** is a **t** by **m** matrix over the integers that holds the exponents used to map an input vector to the transform space,
- **ε** is a **t**-dimensional vector for each feature in the transform vector containing the exponents for each term in the input vector.

E is created when fitting the polynomial features object, which is described below.

The process for creating polynomial features from an input vector can be represented

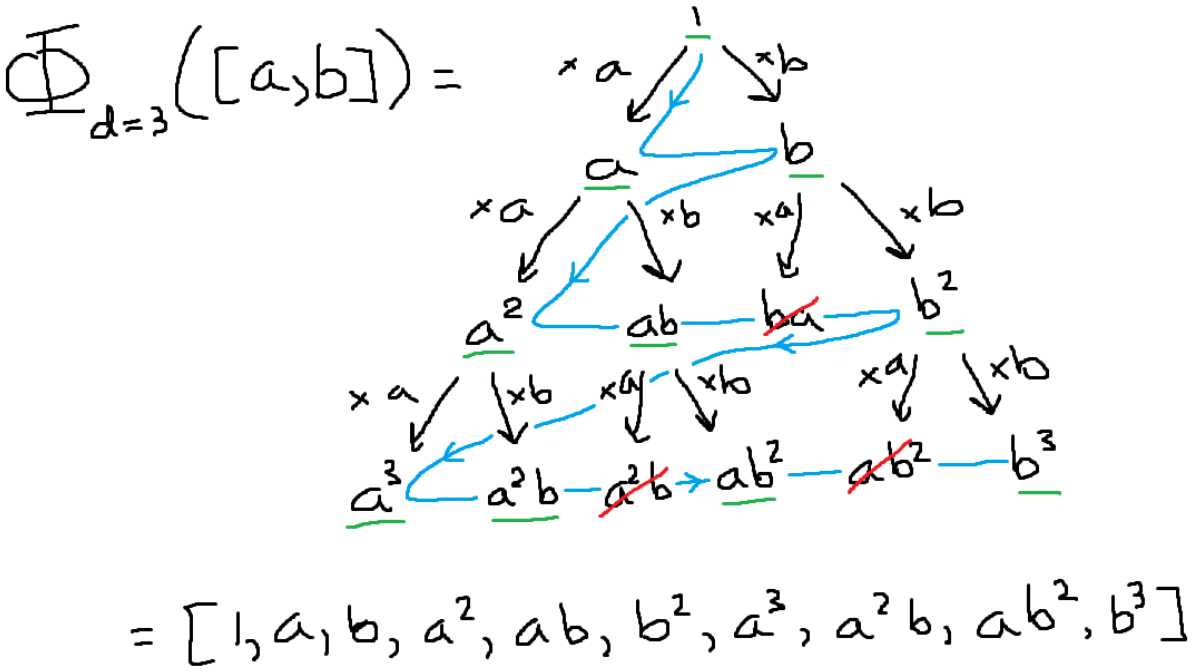

Figure 8.2.1