Technical University of Denmark

Hand in - 1

Course: 02582 - Computational Data Analysis

Authors:

Andreas Sønderbo Patscheider - s185034
Jonatan Hauge Steffensen - s230368
Sofus Laub Erdal - s240398

March 18, 2024

Contents

1 Introduction

The objective of this case is to select and train a well suited predictive model based on a training set, use this model to predict on a test set and estimate the prediction error. In order to do so we will test and compare the performance of various suited regression models using a nested cross validation loop, which will also provide an estimate of the generalization error (root mean squared error in our case). To select the parameters of the best model found in the nested cross validation we will do one final cross validation to find the optimal parameters of our chosen model and use this model to predict on the test set.

Throughout the report we will discuss our choices regarding both preprocessing, imputation of missing data, model selection and estimate of prediction error.

2 Data

The provided data consists of two data sets, one being our training set, upon which we train our models and the other our test set, upon which we use our trained model to predict.
The training data consist of 100 observations of the continuous outcome variable y and 100 features x_i, 95 of these being continuous and 5 being categorical. An inspection of the categorical variable revealed that the 'C_2' column had the categorical value 'H' in all entries, so we decided to remove this feature since a constant feature value does not provide any information about the target values.

Describe how you handled missing data
Since both our training and test data have missing values we need to implement some kind of imputation scheme as part of the preprocessing. In the figure below we have displayed the missing values. In the training data around 14.1 % of the data is missing and in the test data around 14.9% of the data is missing. The data seems to be missing completely at random in both cases except that in the training data we have no missing values in the last 4 categorical features.

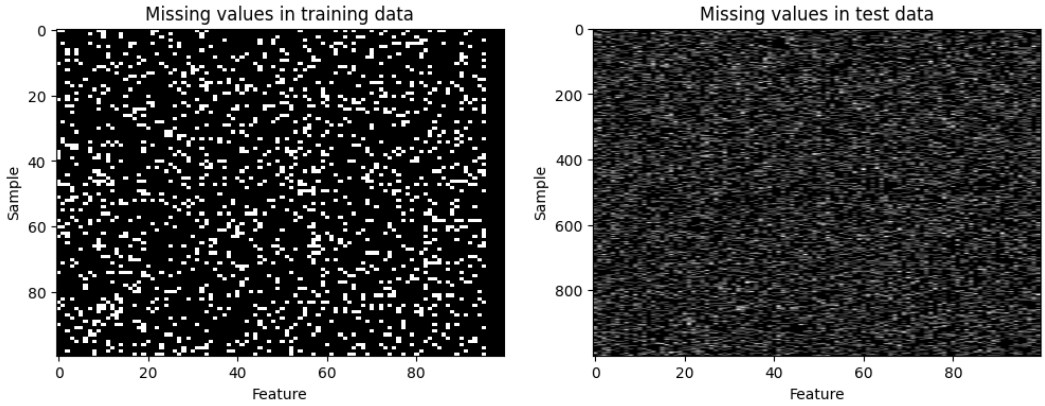

Figure 1: Missing values (indicated by whitespace) in both training and test set

Describe how you handled factors in the features (categorical variables)
We have utilized the method of 'One-hot encoding' of these features which will be elaborated in the following section.

In figure 2 the scale of our raw data is presented. What we can observe is that the features, as well as the one-hot encoded categorical variables, seem to fairly similarly distributed.
In figure 3 the variance and mean of our datasets is visualized. Each blue dot represents the coordinate of values $(\bar{x}, \bar{x}_{new})$. We can see that the mean, sorted by value, aligns almost linearly, indicating a symmetrical distribution of the feature-values in the 2 datasets. The variance boxplot suggests that the smaller dataset X is more affected by outliers, and therefore has a higher variance. Assuming random sampling for the

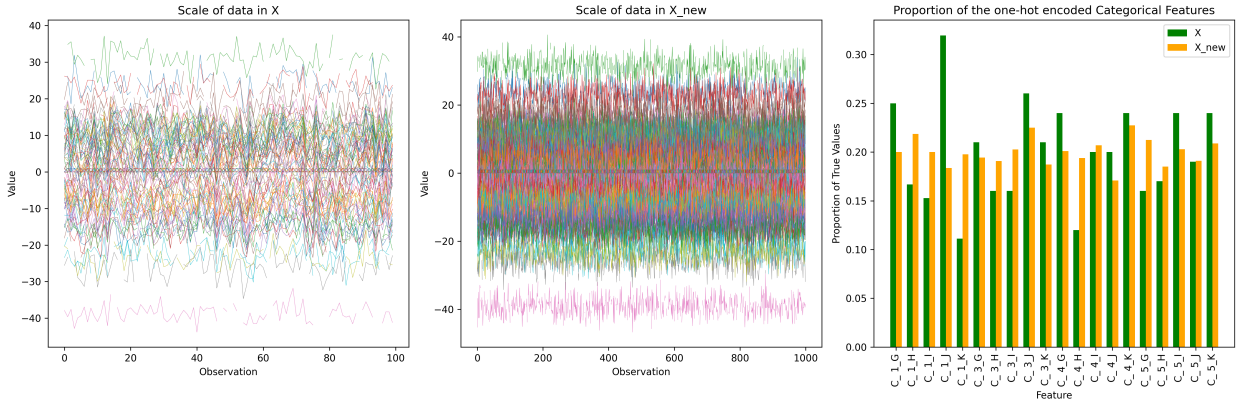

Figure 2: Scale of features in data

2 datasets, Xnew captures a broader range of the natural variability within the data, leading to a more accurate (potentially lower) estimate of the variance.

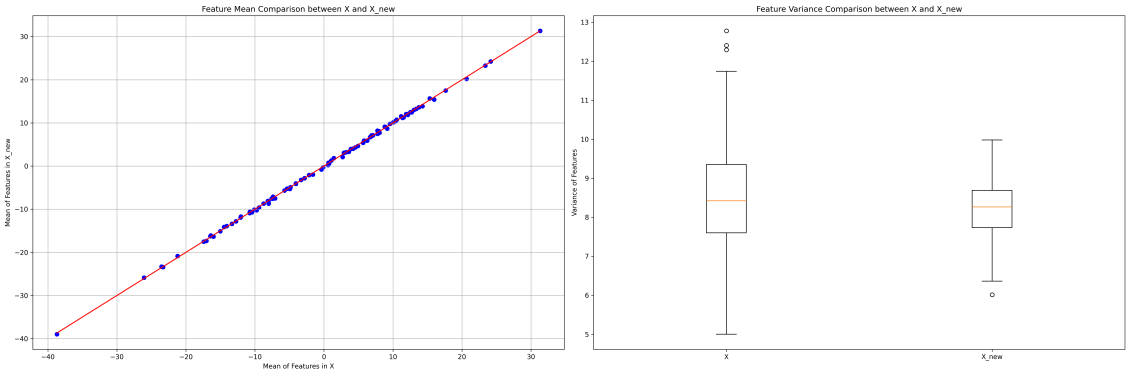

Figure 3: Mean and Variance of data

3 Models

3.1 Elastic Net

Based on our dataset, which contains 100 features for 100 observations, as well as considering the potential for multicollinearity among the features, a choice of model could be Elastic Net. This is because Elastic Net is adept at handling datasets where the relationships between variables are not entirely clear or where variables may be highly correlated. It effectively combines the feature selection capabilities of Lasso (L1 penalty), which can zero out less important features, with the regularization strength of Ridge (L2 penalty), which distributes the coefficient values more evenly across correlated features. This combination allows Elastic Net to maintain model complexity at an optimal level, reducing the risk of overfitting while ensuring that important features are not overlooked.

3.2 Random Forest

As explained earlier the provided data is 100 by 100 and with the categorical features handled, explained in the next section, our training sample consists of fewer observations n than features p, $n << p$. As explained the multicollinearity among the features is a risk when dealing with the case with $n << p$ but also the

risk of having redundant or irrelevant features. A applicable model to implement for a data set like this is the Random Forest model, which utilizes the method of bagging, bootstrapping, random feature selection and CART. By utilizing bootstrapping it introduces diversity into each bootstrap sample, allowing the algorithm to capture different aspects of the underlying data distribution, thus reducing bias. By utilizing the random feature selection bootstrapping the model becomes more diverse, capturing different patterns and relationships in the data and by randomly selecting features at each split helps to decorrelate the trees in the forest, reducing the overall variance of the ensemble. Though if the dataset contains irrelevant features the performance of random forest will drop due to to many splits trying to fit to irrelevant features. Therefore in the implementation we also tested random forest with a feature selection before the fit based on a Lasso regression.

4 Methods

4.1 One-hot encoding

As briefly presented we've utilized the method of 'One-hot encoding' to handle the categorical variables in our training data. By utilizing One-hot encoding we convert each column of categorical variables into a binary feature vector where a value of 1 indicates the presence of the corresponding category and 0 otherwise. In our training data, we have four columns of categorical values (we removed C_2 since it was constant with value 'H') each having the value of either 'G', 'H', 'I', 'J', or 'K meaning five binary feature vectors will be created for each column, a total of 20 new columns.

4.2 Normalization

As presented in figure 2 our raw training data both shows high variance in scale of our features while also showing missing values. The issue with the missing values will be handled in the following section.
Having features with values of a wide range can cause issues for distance-based imputation algorithms like K-nearest neighbors, which we'll introduce later, and for algorithms using regularization like Ridge, Lasso, and Elastic net as just described. To mitigate these issues and have models perform more stable we'll standardize our data using the standard-scaling which scales each feature to have a mean of 0 and a standard deviation of 1.

4.3 Imputation

As described in the previous section we've utilized imputation to handle missing data in both our training data and test data.
In this case we have investigated the performance of two different imputation schemes presented below.

4.3.1 Iterative imputation

The iterative imputation replaces missing values by modeling a separate model for predicting the missing values based on the values of other features in the data set. This method ensures dependencies between variables, especially for data sets with missing values across multiple features. The assignment is done iteratively using a robin-hood fashion meaning that each feature with missing values is imputed in sequence, and the imputation for each feature is performed iteratively until convergence.

4.3.2 KNN imputation

The KNN imputation assigns missing values based on the values of neighboring observations. The parameter of k controls how many neighbors to consider when imputing values. Once the k-nearest neighbors are identified, the missing value is imputed by aggregating the values of the corresponding feature from these neighbors. The weights of the neighboring observations can be uniformly distributed or distributed based

on the distance to the other features of the imputed datapoint.

Both methods apply to our case as our provided data contains missing values that need to be replaced and not dropped to preserve sample size as our initial sample size only counts 100 observations. Furthermore, we avoid adding additional bias by utilizing these methods instead of using simple imputation methods such as replacing missing values with the mean or most frequent value.

4.4 Nested Cross Validation

To ensure the selection of the most appropriate model for predicting on our actual test data, we employ the methodology of Nested Cross Validation (NCV), a technique designed to evaluate the performance of different predictive models with a higher degree of reliability and robustness. NCV operates on a two-level cross-validation process: an outer loop for assessing the model's performance and an inner loop for hyperparameter tuning.
In the outer loop, the dataset is partitioned into K_{outer} distinct folds. In each iteration of this loop, one fold is reserved as the test set while the remaining folds are combined to form the training set. This partitioning ensures that each fold serves as the test set exactly once, allowing for the comprehensive evaluation of the model's performance.
In each iteration, the test set is "untouched", which enables a reliable performance evaluation of the model. The training sets are then again split into several folds, to optimally tune the hyperparameters. Essentially what this means, is that for each K_{outer} iteration of training-test-split, we do K_{inner} iterations of cross validation splits for hyperparameter tuning.
After identifying the optimal hyperparameters for a given model, for that specific training-test-split, the RMSE of the model's prediction on the test-set is evaluated. This RMSE is averaged out for all K_{outer} iterations, to more accurately determine a *true* RMSE for a given model. Note that different hyperparameters are chosen in each inner loop, which is exactly the point of the NCV setup: we wish to estimate how well different models predict on an unknown test set when fitted on a training set with optimal parameters, but given the small amount of data available we do not want to set aside a single test set to begin with, but instead do this several times via the nested cross validation.
Thus by applying NCV to multiple models, we are able to perform a comparative analysis in a fair and unbiased manner, as each model is evaluated based on the same data partitions and under the same conditions.

4.5 Implementation

In our implementation we have used 10 outer folds and 5 inner folds for the NCV. Usually with a larger training set available we would have chosen the imputation method and its hyperparameters beforehand by comparing performance on a test set where we artificially mask out known value, impute them and compute the error. But given the small amount of data we have available we have chosen to include the choice of imputer and its hyperparameters as part of the nested cross validation loop. Furthermore we have chosen to include the test set 'Xnew' (with unknown target values) in the imputation fit to make the imputation better. This also means that the test set 'Xnew' has to be normalized. Here we have also chosen to include the test set such that we in each inner loop normalize based on the inner training data and the test data 'Xnew' (not the test data from any of the CV loops).[1] We are aware that this might introduce a small bias since the parameters of the imputer is chosen also based on 'Xnew' but we consider this trade-off reasonable since we in this way get a more stable imputation and thereby less variance in our predictions. As our goal is to achieve the highest prediction only on this particular data-set, the can allow a slight increase in bias for the model's generalized prediction on unseen data.

When we normalize the data we exclude the 20 columns of one-hot-encoded categorical values and instead keep those as 0's or 1's. Then we do the imputation which is then followed by a transformation of the

[1] As displayed visually in figures 2 and 3 the distribution of the features in the 2 datasets are pretty similar, indicating that it could be beneficial to include both in the imputation and normalization for this particular task.

one-hot-encoded columns such that only one of every 5 columns contains a 1 (The argmax after impuation) and the rest contains 0's. The rest of the loop follows the procedure of nested cross validation as described in the previous section.

At last we choose the model and imputer based on the RMSE estimated from the nested cross validation and report this as our RMSE estimate. To make our final predictions we do a single cross validation of this model, choose the best parameters based on this, then we fit this model on the whole training set and use this to make our final predictions.

5 Model Results

As presented in table 1, the results from our implementation show that the model minimizing the RMSE is the Lasso model using the Iterative Imputer. The same results are also visually presented in figure 4. Derived from the initial implementation we observe the Elastic Net model being the best performative model and all models perform best under the Iterative Imputer. However, we observe from the optimal parameters that the optimal penalization method is full L1 penalty, indicating that the Lasso Model outperforms Ridge. Thus we additionally implemented a Lasso model with the Iterative Imputer. We observe a slight improvement in implementing the Lasso model. Furthermore we observe that the random forest models with a feature selection before the fit outperform random forest fits without feature selection, which indicates that we indeed have redundant features like the full weight on L1 penalty by Elastic Net also indicates. In figure 5 both the predictions, residuals and a Q-Q plot of the residuals are plotted. The residuals do indeed look like white noise and the Q-Q plot confirms this, so it seems that our Lasso regression models the dataset well without fitting to the noise.

	Elastic Net	Lasso	RF w.o. feature selection	RF w. feature selection
Iterative	16.21	16.18	34.57	32.46
KNN	27.33	x	34.53	32.67

Table 1: RMSE results from implemented models

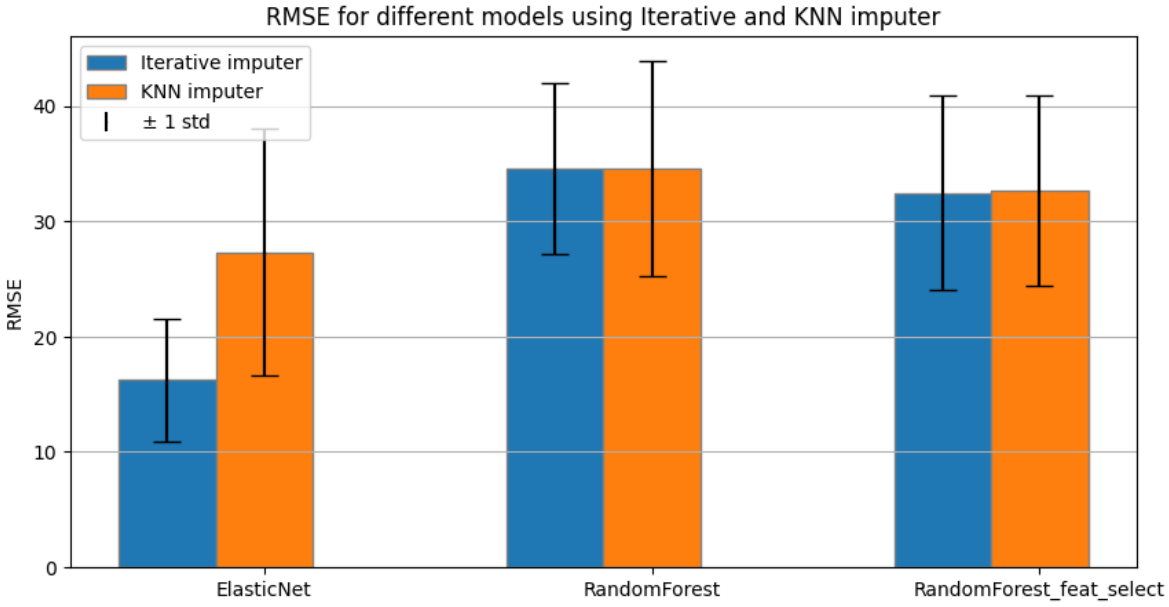

Figure 4: RMSE estimates for different models using either KNN imputation or iterative imputation.

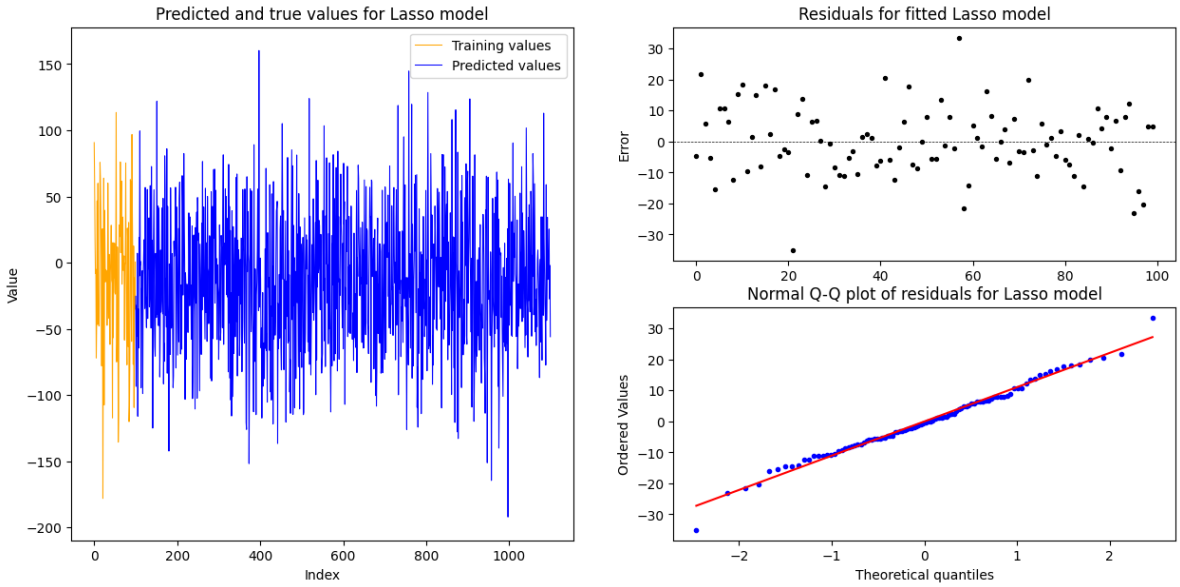

Figure 5: Left: Predictions using optimal Lasso parameters. Right top: Residuals on training data for optimal Lasso model. Right bottom: Q-Q plot of residuals.

With our best-performing model in terms of RMSE, the Lasso model, we have predicted $\hat{y}_{new}$ with the provided x_{new} test sample. The predicted values of $\hat{y}_{new}$ are provided in the enclosed file "predictions_s185034s230368s240398.txt"

The estimated $\hat{\text{RMSE}}$ from our best-performing model, the Lasso model, is enclosed in the file "estimatedRMSE_s185034s230368s240398.txt"