Karlsruher Institut für Technologie

Fakultät für Wirtschaftswissenschaften

Institut für Operations Research

Lehrstuhl Analytics and Statistics

Prof. Dr. Oliver Grothe

Variable selection in regression tasks

Seminararbeit

Kai Reffert

Matrikelnr.: 2279216

16.01.2022

Betreut von:

Fabian Kächele

Contents

1 Introduction

This seminar thesis presents the topic of variable selection in regression tasks, which is particularly relevant to solve the problem of overfitting that emerges from including too many regressors. Furthermore, this issue emerges directly from the bias-variance trade-off and is widely present in cases, working with high dimensional data. The second chapter introduces different metrics to indirectly estimate the testing error, which tries to access the model's prediction power on data it has not worked with so far. Additionally, this chapter also covers the three subset selection methods: best subset selection, forward stepwise selection and backward stepwise elimination. The third chapter contains the shrinkage methods lasso and ridge regression, which shrink the coefficient estimates towards zero in order to decrease the variance and increase the bias. The fourth chapter includes the implementation of the previously explained methods into python. Here, we primarily discuss the linear model, although some methods are also applicable for different model types. Moreover, we are going to use real and created datasets to elaborate and illustrate certain effects. In the end, we compare the different methods results and try to determine which model works best on the real dataset.

2 Subset Selection

In this section we discuss the possibilities of selecting subsets with the best subset approach as well as the forward stepwise selection and the backward stepwise elimination.

2.1 Information criterion

Before we can discuss the subset selection methods below, we first need to understand the different information criterion, which are goodness-of-fit measurements that, in contrast to the R^2 or residual sum of squares, account for overfitting by implementing a penalty term for using too many regressors.

Mallows' $C_P = \frac{RSS_p}{S^2} + 2 \cdot (p+1) - n$ (Source: Zach,2021)

The *p* variable represents the model dimensionality (all included regressors without the Intercept), *n* implements the number of observations and $\frac{RSS_p}{S^2}$ contains the residual sum of squares (*RSS* is its abbreviation) of the model divided by its estimated variance S^2. The mallows' C_P determines the best model, as the model with the minimal C_P out of all C_P-values lower than *p* (Zach,2021). The C_P metric will always equal *p* for the full model and is therefore not able to evaluate the full model. Additionally, if only the full model has a C_P-value close to *p* and all other models have higher C_P values, it suggests that some important regressors are not included in our dataset (Dr. Iain Pardoe et al.,2018). The Akaike Information criterion and the Bayesian Information criterion are probabilistic statistical measurements, which account for model complexity and model performance.

$$AIC = 2p - 2\,ln(L)$$ (Source: Bevans,2021)

$$BIC = ln(n)p - 2\,ln(L)$$(Source: Bevans,2021)

Here, *p* is the number of independent variables used, n represents the quantity of observations, and L is the model's log-likelihood estimate. Ideally, we select the minimal *AIC* and/or BIC values, furthermore a model is significantly better than another model, when its *AIC* is two units lower (Bevans,2021). The *BIC* metric pays more attention to the complexity term, as for *n* larger than seven the *ln(n)* of the *BIC* is larger than two, therefore the *BIC* penalty for adding additional parameters *k* is higher, compared to the *AIC* penalty. In conclusion, the *BIC* usually selects smaller, less complex models than the *AIC* (James G. et al., 2021, p.234). The $\bar{R}^2$ is another way of calculating the goodness-of-fit, this metric is, as the name suggests, computed out of the R^2, and a $\bar{R}^2$ close to one is also optimal. The R^2 is modified in a way so that adding additional regressors can also decrease the $\bar{R}^2$, this is the case when they do not provide additional significant information to the model (Nair,2019).

$$\bar{R}^2 = 1 - (1 - R^2)\frac{(n-1)}{(n-p-1)}$$ (Source: Nair,2019)

This formular implements *n* as the number of observations, *p* as the number of regressors and R^2 as the models R^2.

2.2 Best Subset Selection

In the Best Subset Selection, we are searching for a certain subset of active regressors X_A, which leads to a similarly good model compared to the full model, while having fewer regressors in order to eliminate unrelated variables (Weisberg, 2014, p.237-238). We perform the Best Subset Selection by following the algorithm illustrated in Fig. 2.2.1, here we fit the least squares model for all possible combinations of the *p* predictors, resulting in a comparison of 2^p models. The first and second steps are to fit all models containing exactly *k*-regressors, with *k∈[0, ..., p],* comparing their training error with *RSS* or R^2 and choosing the subset that minimizes these criteria. This gives us the resulting subsets for M_0, ..., M_p models. Out of these *p+1* models, we then select the model with the minimum test error, more in Chapter 2.5. If we also use *RSS* related measurements in this step we would always end up with the full model, which minimizes the training error, however the possibility of overfit is relatively high for this selection (James G. et al., 2021, p.227-228).

The best subset approach is also applicable for other model types, for example the logistic regression. The main drawback of this subset selection method is its computational expenses, as those scale with 2^p, which becomes infeasible when *p* is larger than 40 (James G. et al., 2021, p.229). That is why we have to consider computational feasible alternatives, like the stepwise methods of 2.3-2.4. However, by using the stepwise methods we cannot guarantee to find the best subset anymore. Alternatively, we can use the *leaps and bounds* algorithm for *p* <31, which also computes a fraction of the possible subsets (Weisberg, 2014, p.239).

Algorithm 6.1 *Best subset selection*

1. Let $\mathcal{M}_0$ denote the *null model*, which contains no predictors. This model simply predicts the sample mean for each observation.
2. For $k = 1, 2, \ldots p$:
   (a) Fit all $\binom{p}{k}$ models that contain exactly k predictors.
   (b) Pick the best among these $\binom{p}{k}$ models, and call it $\mathcal{M}_k$. Here *best* is defined as having the smallest RSS, or equivalently largest R^2.
3. Select a single best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

Fig. 2.2.1 The general best subset selection algorithm and its steps are displayed here. (Source: James G. et al., 2021, p.227)

2.3 Forward Stepwise Selection

The starting point for the forward stepwise selection is the null model, containing only the Intercept as a regressor, however it is possible to start with other models involving some regressors, this is especially reasonable when we have regressors that are necessary for the model and should therefore always be included (Weisberg, 2014, p.240). In each iteration of the algorithm deployed in Fig. 2.3.1 we look at the training error of all models which add one additional parameter to the current model. We choose the subset which minimizes the training error by having the lowest *RSS* or highest R^2 value. At the end, we can choose from the *p+1* models, again more in Chapter 2.5.

Algorithm 6.2 *Forward stepwise selection*

1. Let $\mathcal{M}_0$ denote the *null* model, which contains no predictors.
2. For $k = 0, \ldots, p-1$:
   (a) Consider all $p-k$ models that augment the predictors in $\mathcal{M}_k$ with one additional predictor.
   (b) Choose the *best* among these $p-k$ models, and call it $\mathcal{M}_{k+1}$. Here *best* is defined as having smallest RSS or highest R^2.
3. Select a single best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

Figure 2.3.1 This algorithm displays the steps of the forward stepwise selection. (Source: James G. et al., 2021, p.230)

The stepwise methods only look at a fraction of the possible combinations, therefore they have a clear computational advantage over the best subset selection, as they only look at $(1 + \frac{(p+1)}{2} \cdot p)$– models. Another major upside of the forward stepwise selection is its availability in cases where *p* is larger than *n*, as it starts with the null model and is therefore able to also compute models up to *p* equal to *n* regressors. The main issue of the forward stepwise selection is that it is tied to the choice of a regressor, therefore we have no guarantee to find the model with the smallest training error (James G. et al., 2021, p.229-230).

2.4 Backward Stepwise Elimination

The backward stepwise elimination method is similar in functionality to the forward stepwise selection, but we start with the full model, containing all regressors. The algorithm for this variable elimination process is shown in Fig. 2.4.1, where we subtract one parameter from the model in each iteration. Both stepwise algorithms look at the same number of models, therefore their computational expenses are also the same $(1 + \frac{(p+1)}{2} \cdot p)$, moreover we also cannot guarantee to find the best subset by using the backward stepwise elimination. In contrast to the forward stepwise selection, we cannot use the backward stepwise selection in cases where *n* is smaller than *p*, because the full model cannot be fit using least squares in these dimensions (James G. et al., 2021, p.231).

Algorithm 6.3 *Backward stepwise selection*

1. Let $\mathcal{M}_p$ denote the *full* model, which contains all p predictors.
2. For $k = p, p-1, \ldots, 1$:
   (a) Consider all k models that contain all but one of the predictors in $\mathcal{M}_k$, for a total of $k-1$ predictors.
   (b) Choose the *best* among these k models, and call it $\mathcal{M}_{k-1}$. Here *best* is defined as having smallest RSS or highest R^2.
3. Select a single best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

Figure 2.4.1 This algorithm displays the steps of the backward stepwise elimination process. (Source: James G. et al., 2021, p.231)

As an outlook for better stepwise methods, we can regard hybrid stepwise selection algorithms, where it is possible to perform iteration steps in either the forward or backward direction. These hybrids mimic the best subset selection closer than the forward or backward stepwise selection methods, while maintaining their low computational effort (James G. et al., 2021, p.232). However, hybrid selection methods are not discussed further in this seminar thesis.

2.5 Choosing the optimal subset model

In order to determine which of the selected subsets leaves us with the best model, we can use two different ways to determine the test error. The first way is to indirectly estimate the test error by using the different information criteria discussed in 2.1 on each subset. In the end, we choose the model after the One Standard Error Rule, which selects the subset with the fewest parameters from all subsets that are within one standard error of the optimal test error estimate (James G. et al., 2021, p.232 ff.). Figure 2.5.1 elaborates the impact of the One Standard Error Rule, as we end up with smaller models that are easier to interpret compared to the optimal information criteria selections.

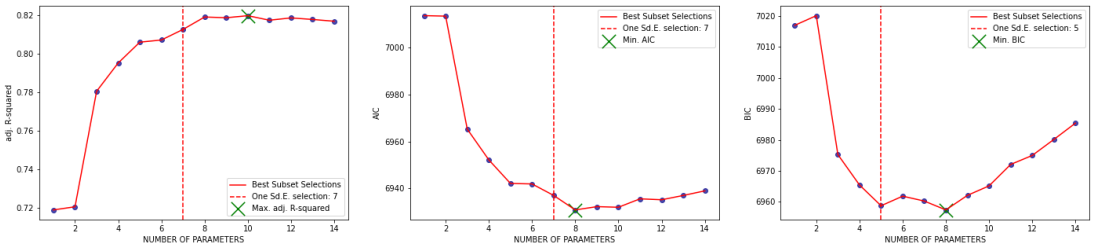

Figure 2.5.1 The best subset selections for each subset size are depicted with their according AIC, BIC and $\bar{R}^2$ values. The green crosses show the optimal value for each different criterion. The dashed vertical lines are the corresponding results of the One Standard Error Rule selection. (Source: Author's illustration)

The alternative to using information criteria is to directly determine the test error by using cross validation. Here we randomly split the data into two parts, one testing data set and one training data set. Then we use the training data set to determine the model and its coefficients, in this context we would determine the coefficients for each chosen subset. In the end we calculate the Mean Squared Error of the true testing target variable and the estimated values of our model (Payam R. et al., 2009, p.532 ff.).

2.6 Limitations

The first limitation of our subset results is that the output of subset selections is biased, this applies to the coefficients and the R^2, which are both too large, as well as the confidence intervals and the p-values, which are both too low. However, by using cross validation we can estimate the output with the training data set, resulting in a likely unbiased output. Another drawback of the stepwise selection methods is that they are highly unstable, so for different training datasets there is a high possibility that the output contains a different subset each time. However, this effect is reduced when we work with smaller dimensions of data and observations of over fifty. In addition, we can also use methods like bootstrapping to choose the variables that are frequently selected (Choueiry G., 2019).

3 Shrinkage Methods

In this chapter, we will look at shrinkage models, which implement a penalty term to shrink the coefficient estimates. This implementation accounts for overfitting, because the shrunken coefficients result in a decrease of variance and increase in bias.

3.1 The Ridge Regression

The ridge regression is based on the L^2-regularization and has the form of

$$\hat{\beta}^R = \min_{\beta} \left\{ RSS + \lambda \sum_{j=1}^{p} {\beta_j}^2 \right\}$$ (Source: James G. et al., 2021, p.237)

with λ >=0, a tuning parameter. The term on the right is called a penalty term, which gets smaller when the coefficient estimates (β_j) get smaller, the Intercept is not included here. When we set λ to zero, the ridge regression estimate equals the least squares estimate, and when λ approaches infinity the β_j estimates shrink towards zero. However, these estimates only equal exactly zero, when λ equals infinity. In this case, we receive the null model, where the only parameter that does not equal zero is the Intercept. Thus, the ridge regression is unable to perform variable selection, resulting in a worse model interpretability compared to subset selection methods (Dinov I., 2018, p.576 ff.). When the tuning parameter λ increases the model has less flexibility, therefore the variance decreases and the bias increases, moreover we can use the bias-variance trade-off to optimize the ordinary least squares model in cases where its variance is relatively high. This is the case in Figure 3.1.1 (Left), where the test mean squared error gets lower at first because the variance decreases more than the bias increases. Additionally, we can see that the test mean squared error has an optimal point, which is dependent on the tuning of the λ parameter (James G. et al., 2021, p.239 ff.).

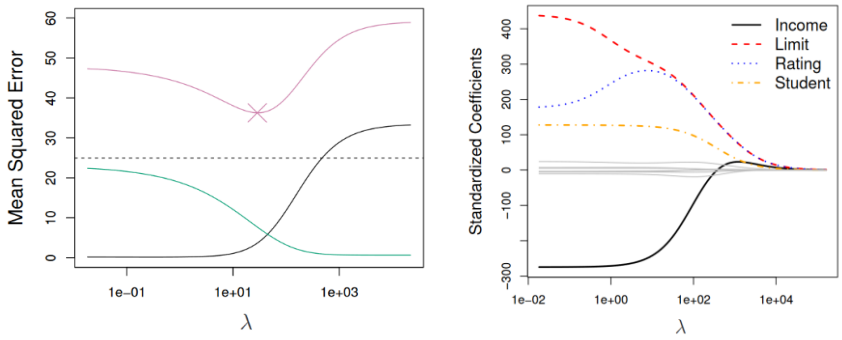

Figure 3.1.1 Left: The test mean squared error (purple), bias(black) and variance(green) are depicted for according λ values. Here, the data used is simulated. (Source: James G. et al., 2021, p.240) Right: The standardized $\hat{\beta}_\lambda^R$ coefficients (of a Credit data set) are plotted against λ. (Source: James G. et al., 2021, p.238)

In the right graph of Fig. 3.1.1, we can see that the coefficients $\hat{\beta}_\lambda^R$ are dependent on λ and shrink towards zero when λ increases. Additionally, we work with standardized coefficients, because the ridge regression is not scale equivariant, which the least squares model is, so different regressor scales can change the results significantly. The change of scale in the least squares model is cancelled out because of the direct multiplication of β_j and x_j (the regressor vector). In contrast to this, the ridge regression cannot account for a scale change, as it additionally includes the penalty term that contains only β_j (James G. et al., 2021, p.238-239). Consequently, it is reasonable to standardize the regressors, using the z-score to reach a mean of 0 and a standard deviation of 1 across all regressors. Lastly, we transform the results back to have a better interpretability (Dinov I., 2018, p.582).

3.2 The Least Absolute Shrinkage and Selecting Operator

The "Least Absolute Shrinkage and Selection Operator", written as LASSO or lasso, has a similar setup to the ridge regression model. However, the lasso uses the L^1-norm for its penalty term:

$$\hat{\beta}^L = \min_{\beta}\left\{RSS + \lambda\sum_{j=1}^{p}|\beta_j|\right\}$$ (Source: James G. et al., 2021, p.241)

with λ >=0, a tuning parameter. This change of norm solves ridge regressions main problem, the lack of variable selection, as the LASSO model can in fact set predictors exactly to zero if λ is large enough. This improvement results in models that are often easier to interpret. Obviously, the lasso and ridge regression return the same outputs, when λ equals zero and λ equals infinity, but the model selection between these values is much different (James G. et al., 2021, p.241 ff.).

3.3 Ridge vs. Lasso

In order to understand the difference in variable selection, we first have to look at different ways of defining the ridge and the Lasso model:

Ridge: $\min_{\beta}\left\{\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2\right\} s.t. \sum_{1}^{P}{\beta_j}^2 \leq s$ (Source: James G. et al., 2021, p.243)

Lasso: $\min_{\beta}\left\{\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2\right\} s.t. \sum_{1}^{P}|\beta_j| \leq s$ (Source: James G. et al., 2021, p.243)

As we can see both shrinkage methods can be written as an optimization problem with a constraint *s*, which limits our variables to be within a certain threshold. If the constraint is large enough, we receive the ordinary least squares estimator, analogically to the case where λ equals zero. Similarly, we can define a constraint *s* to return an equal model for every value of λ (James G. et al., 2021, p.243).

Figure 3.3.1 shows the restriction areas of the ridge and lasso models, for the scenario where *p* equals two. The intersection of the restriction area with the closest contour line is the value of the shrunken estimator, as this point has the minimal RSS out of the possible estimators in the restriction area. However, we can see that the intersection for the ridge regression cannot be fit on one of the axes, where one regressor would equal zero, as it always encounters the circular restriction first (James G. et al., 2021, p.244-245).

In general, the ridge regression does better when there are many predictors related to the response, and all of them are similar in coefficient size. In contrast, the lasso model is better suited, when many regressors equal zero, while only a few have values exceed zero. Accordingly, the lasso model implicitly assumes that there are regressors which are not related to the response at all, while only few regressors are truly related to the response (Oleszak, 2019).

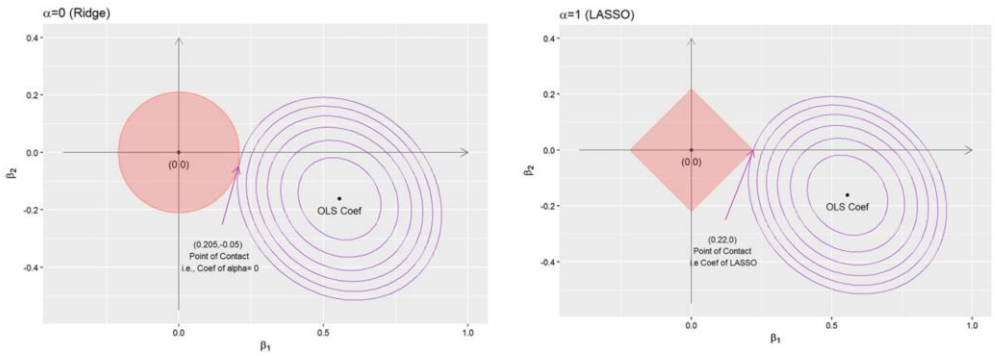

Figure 3.3.1 Two Plots of β_1 and β_2 against each other, the purple RSS contour lines surround the OLS coefficient. Left: The red area represents all combinations for β, which are within the ridge constraint. (Source: Dinov I., 2018, p.596) Right: The red area represents all combinations for β, which are within the lasso constraint. (Source: Dinov I., 2018, p.596)

4 Implementation

This chapter deals with the implementation of the subset and shrinkage methods discussed in chapter 2 and 3. At first, we will introduce the datasets, then model algorithms in python, which follow the previously introduced subset and shrinkage methods. At the end, we will discuss and interpret the results of the different methods.

4.1 The Dataset

In this modelling task we are going to use a real-life dataset containing player statistics from the National Basketball Association[1], short NBA, as well as their salary[2], which is our target variable in the linear model. The data includes eighteen predictors and 290 observations, from the 2021-2022 NBA season and is current to the sixth of December 2021. Because of representability reasons, all players had to play at least in 70% of their Team's games in order to make this dataset. A potential objective for our variable selection methods on this dataset is to detect the player stats that have a good prediction power on the salary, when they are included in a linear regression model while maintaining a simple form.

+/-	Box-Plus-Minus per Game	FG%	Field Goal Percentage	PTS	Points per Game
3P%	3-Pointer Percentage	FP	Fantasy Points per Game	REB	Rebounds per Game
AGE	Player's age	FT%	Freethrow Percentage	STL	Steals per Game
AST	Assists per Game	GP	Games Played	TD3	Triple Doubles Total
BLK	Blocks per Game	MIN	Minutes Played per Game	TOV	Turnovers per Game
DD2	Double-Doubles Total	PF	Personal Fouls per Game	salary	Salary for 21/22 season

Table 1 All eighteen variables used in the NBA dataset are explained in this Table. (Source: Author's illustration)

[1] (Data source: https://www.nba.com/stats/players/traditional/?PerMode=Totals&sort=PTS&dir=-1, Last accessed: 6th December 2021)
[2] (Data source: http://www.espn.com/nba/salaries, Last accessed: 6th December 2021)

In addition, we also use generated datasets, to elaborate effects that were introduced in the previous Chapters 1 -3. The explicit data generating processes are performed with the following python-method depicted in Fig. 4.1.1.

```
def dataset_declaration(observations,mu=0,related=50,sigma=1,noise=20):
    np.random.seed(42)#define the seed to asure compatibility
    X=np.random.normal(mu, sigma, (observations, related))#related regressormatrix
    X_noise=np.random.normal(mu, sigma, (observations, noise))#unrelated regressormatrix
    if (related==0):
        y=np.random.normal(mu,sigma,(1,observations))#normally distributed y if related regressors aren't provided
    else:
        y=np.zeros((1,observations))
    for i in range(X.shape[1]):
        y+=X[:,i]#sum the related regressors->true coefficients are 1
    X=np.append(X, X_noise,axis=1)#combine unrelated and related regressors
    y=y.transpose()
    return X,y
```

Figure 4.1.1 The method depicted creates normally distributed datasets, which fit certain inserted criteria. (Source: author's illustration)

In this self-defined method, we can determine the number of observations, related regressors and unrelated regressors (*noise*), in addition, we can also define the mean (*mu*) and the standard deviation (*sigma*) of the generated data. Finally, we receive a regressor matrix *X*, which follows a normal distribution, and a target vector *y*, which is dependent on the related regressors.

4.2 Best Subset Selection

This chapter deals with the implementation of the best subset selection method, discussed in Chapter 2.2. The code for the python realisation of the best subset selection is depicted in the following Figure 4.2.1, here our input consists of a pandas DataFrame object and its target variable(s) column name(s). We iterate over each possible subset selection by using two for-loops, here we calculate the *RSS* and R^2 values for each subset by using the "*lrg()*"-command, which is another self-defined function that outputs the *RSS* and R^2-values for a linear regression of a given *X* and *y*. In this process we save the results *RSS*, R^2, number of variables and names of included variables to temporary lists. In the end we create a DataFrame to return all subsets and their corresponding *RSS* and R^2 values, we also create two new columns, where all subsets are filtered by the minimal *RSS* and the maximum R^2.

The data used in Figure 4.2.2 consists of the first fourteen variables of our NBA dataset, as the full data set is costly in terms of time. Moreover, this shows a problem of the best subset selection method in praxis, as it struggles to feasibly calculate the optimal subset for large numbers of regressors. In this case, we have to shorten the number of regressors while leaving others out. In Fig. 4.2.2, we can also see why the *RSS* and R^2 values are not good measurements for the testing error, as we would always select the full model, which comes with a high possibility of overfitting. The eventual subset selection will be discussed in 4.5, when we compare the three methods and their selections.

```python
def best_subset(df,output='pred'):
    '''This function calculates all subsets of a given DataFrame-object'''
    X,y=df[df.columns.difference([output])],df[output]#defining the target variable and the regressormatrix
    #initializing empty lists to save the RSS,R-squared,subset variables and the number of variables
    RSS_list,R_squared_list,variable_list,number_variable=[],[],[],[]
    #running through all possible counts of subsets
    for k in range(1,len(X.columns)+1):
        #looping over all possible subsets with size k
        for combo in itertools.combinations(X.columns,k):
            model=lrg(X[list(combo)],y)#returns the RSS- and the R-squared value of the linear regression
            #appending the results of the current subset
            RSS_list.append(model[0])
            R_squared_list.append(model[1])
            variable_list.append(list(combo))
            number_variable.append(len(combo))
    #saving the results in a pandas DataFrame-object
    df_best_subset = pd.concat([pd.DataFrame({'variables':variable_list}),pd.DataFrame({'number of variables':number_variables}),
                        pd.DataFrame({'RSS':RSS_list, 'R_squared': R_squared_list})], axis=1, join='inner')
    #saving the best RSS and the best R-squared results for each different subset size in new columns
    df_best_subset['min_RSS']=df_best_subset.groupby('number of variables')['RSS'].transform(min)
    df_best_subset['max_R_squared']=df_best_subset.groupby('number of variables')['R_squared'].transform(max)

    return df_best_subset
```

Figure 4.2.1 This method performs the best subset selection process when given DataFrame object. (Source: Author's illustration)

	number_of_variables	RSS	R_squared	variables
0	1	1.089929e+16	0.497275	[FP]
1	2	8.291898e+15	0.617540	[AGE, FP]
2	3	7.826176e+15	0.639021	[AGE, FG%, FP]
3	4	7.631272e+15	0.648011	[3P%, AGE, FG%, FP]
4	5	7.487532e+15	0.654641	[+/-, 3P%, AGE, FG%, FP]
5	6	7.383353e+15	0.659446	[+/-, 3P%, AGE, FG%, FP, PTS]
6	7	7.284740e+15	0.663995	[+/-, 3P%, AGE, DD2, FG%, FP, PTS]
7	8	7.247789e+15	0.665699	[+/-, 3P%, AGE, DD2, FG%, FP, GP, PTS]
8	9	7.225291e+15	0.666737	[+/-, 3P%, AGE, DD2, FG%, FP, GP, PF, PTS]
9	10	7.217586e+15	0.667092	[+/-, 3P%, AGE, DD2, FG%, FP, FT%, GP, PF, PTS]
10	11	7.203990e+15	0.667719	[+/-, 3P%, AGE, AST, BLK, DD2, FG%, FP, GP, PF...
11	12	7.193943e+15	0.668183	[+/-, 3P%, AGE, AST, BLK, DD2, FG%, FP, FT%, G...
12	13	7.185583e+15	0.668568	[+/-, 3P%, AGE, AST, BLK, DD2, FG%, FT%, GP, M...
13	14	7.183632e+15	0.668658	[+/-, 3P%, AGE, AST, BLK, DD2, FG%, FP, FT%, G...

Figure 4.2.2 This is the output DataFrame of the best_subset method (Fig. 4.2.1) filtered after the minimal RSS of each subset. (Source: Author's illustration)

4.3 Forward Stepwise Selection

The implementation of the forward stepwise selection method is visualized in Figure 4.3.1, its input is equal to the best subset input (Fig. 4.2.1). This function works with two lists that represent the variables already included and the variables that have not been selected yet. These two lists are updated in each iteration of the outer for-loop, with the inner for-loop selecting the next regressor based on the smallest *RSS* value. Finally, the results are again put together in a DataFrame with the columns number of features, features of the subset, *RSS* and R^2, each row represents the selected subset for the different numbers of predictors, see Fig. 4.3.2.

```python
def forward_stepwise(df,output='pred'):
    y_train,df=df[output],df[df.columns.difference([output])]#defining the output and regressors
    k=df.shape[1]#saving the count of variables available in the Regressormatrix
    #2 lists for the not included and included regressors
    remaining_features,selected_features = list(df.columns.difference(['Intercept']).values),[]
    #empty lists and an empty dictionary to save results in each iteration of the following for loop
    RSS_list, R_squared_list,features_dict = [np.inf], [np.inf],dict()
    #Start with the linear regression of the null model containing only the Intercept
    nullmodel = lrg(df[['Intercept']],y_train)
    #saving the results at the first positions of our lists
    RSS_list[0],R_squared_list[0],features_dict[0]=nullmodel[0],nullmodel[1],['Intercept']
    #the outer for loop iterates over the number of available regressors
    for i in range(1,k):
        best_RSS = np.inf#defining an upper limitation for the possible RSS value
        #the inner for loop is iterating over each combination, of one regressor, out of the remaining features
        for combo in itertools.combinations(remaining_features,1):
            #calculating the linear regression model for the combination plus the already selected features
            model = lrg(df[list(combo) + selected_features],y_train)
            #checking whether the current selection model has a lower RSS, than other selections
            if (model[0] < best_RSS):
                #overwriting the 'best' values
                best_RSS,best_R_squared,best_feature = model[0],model[1],combo[0]
        #updating the remaining and included regressor lists, after each combination has been looked at
        selected_features.append(best_feature)
        remaining_features.remove(best_feature)
        #appending the current selection to the lists
        RSS_list.append(best_RSS)
        R_squared_list.append(best_R_squared)
        features_dict[i] = selected_features.copy()
    #creating the output DataFrame-Object
    df1 = pd.concat([pd.DataFrame({'features':features_dict}),
                     pd.DataFrame({'RSS':RSS_list, 'R_squared': R_squared_list})], axis=1, join='inner')
    df1['number of features'] = df1.index
    return df1
```

Figure 4.3.1 This method performs the forward stepwise selection process on a given DataFrame object. (Source: Author's illustration)

	variables	RSS	R_squared	number of variables
1	[FP]	1.089929e+16	0.497275	1
2	[FP, AGE]	8.291898e+15	0.617540	2
3	[FP, AGE, FG%]	7.826176e+15	0.639021	3
4	[FP, AGE, FG%, 3P%]	7.631272e+15	0.648011	4
5	[FP, AGE, FG%, 3P%, +/-]	7.487532e+15	0.654641	5
6	[FP, AGE, FG%, 3P%, +/-, PTS]	7.383353e+15	0.659446	6
7	[FP, AGE, FG%, 3P%, +/-, PTS, DD2]	7.284740e+15	0.663995	7
8	[FP, AGE, FG%, 3P%, +/-, PTS, DD2, GP]	7.247789e+15	0.665699	8
9	[FP, AGE, FG%, 3P%, +/-, PTS, DD2, GP, PF]	7.225291e+15	0.666737	9
10	[FP, AGE, FG%, 3P%, +/-, PTS, DD2, GP, PF, FT%]	7.217586e+15	0.667092	10
11	[FP, AGE, FG%, 3P%, +/-, PTS, DD2, GP, PF, FT%...	7.211987e+15	0.667350	11
12	[FP, AGE, FG%, 3P%, +/-, PTS, DD2, GP, PF, FT%...	7.193943e+15	0.668183	12
13	[FP, AGE, FG%, 3P%, +/-, PTS, DD2, GP, PF, FT%...	7.187313e+15	0.668488	13

Figure 4.3.2 The DataFrame object displayed is the product of the forward_stepwise method (Fig. 4.3.1), when given a selection of the NBA data set. (Source: Author's illustration)

In Figure 4.3.2, we can see that the output for the forward stepwise selection is remarkably similar to the best subset output, nevertheless a difference occurs for the 11th subset, as the RSS and R^2 values are different. Consequently, we must accept that the forward stepwise selection method cannot guarantee to find all subsets with the lowest training error.

4.4 Backward Stepwise Elimination

The implementation of the backward stepwise elimination method is very similar to the forward stepwise selection method, see Fig 4.4.1, however in each iteration of the inner for-loop, each remaining variable is removed and added again at the end. In this inner loop we overwrite the results if they have a lower *RSS* value

than the current lowest *RSS* value. Afterwards, we save our results into a DataFrame object, which is the output of this function.

```
def backwards_elimination(df,output='pred'):
    '''This function calculates the backward elimination subset selections of a given DataFrame-object'''
    y,df=df[output],df[df.columns.difference([output])]#defining the output and regressors
    k=df.shape[1]#saving the count of variables available in the Regressormatrix
    variables = list(df.columns.difference(['Intercept']).values)
    RSS_list, R_squared_list,variables_list,eliminated = [0], [0],dict(),['None']
    #calculating the output values for the full model
    for i in range(1,k):#first for loop to iterate over the amount of possible subset sizes
        temp_RSS_list,c,best_RSS=[],[],np.inf
        #Checking all possible subsets and saving the results
        for combo in itertools.combinations(variables,1):
            if len(variables)>1:#only remove when there are at least 2 variables left
                variables.remove(list(combo)[0])
                model = lrg(df[variables],y)
                temp_RSS_list.append(model[0])
                if (model[0] < best_RSS):
                #overwriting the 'best' values
                    best_RSS,best_R_squared,worst_feature = model[0],model[1],combo[0]
                variables=variables.copy()+list(combo)
        #adding the results of the iteration
        eliminated.append(worst_feature)
        RSS_list.append(best_RSS)
        R_squared_list.append(best_R_squared)
        variables.remove(worst_feature)
        variables_list[i] = variables.copy()
    #saving the reults in a DataFrame
    df_be = pd.concat([pd.DataFrame({'Variables':variables_list}),
                       pd.DataFrame({'RSS':RSS_list, 'R_squared': R_squared_list}),
                       pd.DataFrame({'Eliminated':eliminated})], axis=1, join='inner')
    df_be['Number_of_variables']=[len(i) for i in df_be['Variables']]
    return df_be
```

Figure 4.4.1 This method performs the backward stepwise elimination process on a given DataFrame object. (Source: Author's illustration)

Figure 4.4.2 shows that the backward stepwise elimination is also unable to select the best subset for each subset size. In this case the differences start with the 13th subset in Fig.4.4.2, as the subset selected contains *PTS* while the best subset includes *FP*. Additionally, this shows that the forward and backward stepwise methods do not select the same subsets.

	Variables	RSS	R_squared	Eliminated	Number_of_variables
1	[+/-, 3P%, AGE, AST, BLK, DD2, FG%, FT%, GP, M...	7.185583e+15	0.668568	FP	13
2	[+/-, 3P%, AGE, AST, BLK, DD2, FG%, GP, MIN, P...	7.194414e+15	0.668161	FT%	12
3	[+/-, 3P%, AGE, AST, BLK, DD2, FG%, GP, PF, PT...	7.207181e+15	0.667572	MIN	11
4	[+/-, 3P%, AGE, AST, BLK, DD2, FG%, GP, PTS, REB]	7.222090e+15	0.666884	PF	10
5	[+/-, 3P%, AGE, AST, BLK, DD2, FG%, GP, PTS]	7.253175e+15	0.665450	REB	9
6	[+/-, 3P%, AGE, AST, BLK, DD2, FG%, PTS]	7.280694e+15	0.664181	GP	8
7	[+/-, 3P%, AGE, AST, DD2, FG%, PTS]	7.427065e+15	0.657430	BLK	7
8	[+/-, 3P%, AGE, DD2, FG%, PTS]	7.550906e+15	0.651718	AST	6
9	[3P%, AGE, DD2, FG%, PTS]	7.771981e+15	0.641521	+/-	5
10	[3P%, AGE, DD2, PTS]	8.141361e+15	0.624483	FG%	4
11	[AGE, DD2, PTS]	8.406759e+15	0.612242	3P%	3
12	[AGE, PTS]	9.031645e+15	0.583419	DD2	2
13	[PTS]	1.190574e+16	0.450853	AGE	1

Figure 4.4.2 The depicted DataFrame object is the product of the backwards_elimination method (Fig. 4.4.1), when its given a selection of the NBA data set. (Source: Author's illustration)

4.5 Comparison of the subset selection methods

This chapter deals with methods that select the subset with the minimal estimated test error. Moreover, we use the method visualized in Figure 4.5.1 to receive the different test error estimates.

```
from scipy import stats
from sklearn.metrics import mean_squared_error as m
def ic(df_subset,df1,df_test,output='pred'):
    '''This method compares the subset-results with Information criteria and direct estimation of the test mse'''
    Xtr,ytr=df1[df1.columns.difference([output])],df1[output]
    Xtst,ytst=df_test[df_test.columns.difference([output])],df_test[output]#the test split from our data
    aicl,bicl,ricl,variables,mse_list=[],[],[],[],[]#lists to save the temporary results of the Information criteria
    for i in df_subset:
        x,y=Xtr[i],ytr#defining x as the selected subsets of the data
        model = sm.OLS(y, x).fit()#fitting the model
        aicl.append(model.aic)#saving the Information criteria and the related variables
        bicl.append(model.bic)
        ricl.append(model.rsquared_adj)
        pred_y=model.predict(Xtst[i])#predicting the target variable
        mse= m(ytst,pred_y)#calculating the test mse of our predictions and the actual values
        mse_list.append(mse)
        variables.append(i)
    #saving our results in a DataFrame
    df=pd.concat([pd.DataFrame({'aic':aicl}),pd.DataFrame({'bic':bicl}),pd.DataFrame({'r_squaredadj':ricl})
                  ,pd.DataFrame({'variables':variables}),pd.DataFrame({'mse':mse_list})], axis=1, join='inner')
    df['nf']=[len(i) for i in df['variables']]#saving the number of variables in a new column
    #output DataFrame for the optimal selection of each Information criterion (minimum and maximum)
    df_selection=pd.concat([df[df['aic']==min(df['aic'])],df[df['bic']==min(df['bic'])],
              df[df['r_squaredadj']==max(df['r_squaredadj'])],df[df['mse']==min(df['mse'])]],join='inner')
    df_selection['']=['MIN AIC','MIN BIC','MAX ADJ. R-sq.','MIN MSE TEST']
    df_selection=df_selection.set_index('')
    #an output dictionary for the one standard error rule selection
    df_selection_one_se=dict()
    df_selection_one_se['One SE AIC']=min(df[df['aic']-stats.sem(df.aic)<=min(df['aic'])]['nf'])
    df_selection_one_se['One SE BIC']=min(df[df['bic']-stats.sem(df.bic)<=min(df['bic'])]['nf'])
    df_selection_one_se['One SE adj R sq']=min(
        df[df['r_squaredadj']+stats.sem(df.r_squaredadj)>=max(df['r_squaredadj'])]['nf'])
    df_selection_one_se['One SE MSE']=min(df[df['mse']-stats.sem(df.mse)<=min(df['mse'])]['nf'])
    return df,df_selection,df_selection_one_se
```

Figure 4.5.1 This method computes different information criteria and the directly estimated test mean squared error. (Source: Author's illustration)

The function *ic* receives *df_subset* (the *variables* column of our different subset output DataFrames), *df1* (our training data split), *df_test* (our testing data split) and the name of the target variable. The following for-loop iterates over all entities of *df_subset*, so over all subsets and their included variables. After that, we fit our model with the selected subset and determine the *AIC*, *BIC*, $\bar{R}^2$ and estimated test *MSE* values through methods supplied by the *scipy* package. Our first output is a DataFrame, which contains all results. The second output is filtered to return the optimal values of each criterion. Our third and last output returns the best subset selection, when we apply the One Standard Error Rule. The different selections for our dataset from the previous three chapters are visualized in Fig. 4.5.2.

This figure shows four graphs for each different test error estimate plotted against the number of variables. First, we can see that the red and blue graphs follow a remarkably similar distribution, they also select the same subsets with the lowest testing error. However, the backward stepwise elimination seems to have a relatively good choice for its subset with six variables, as this subset is selected by three out of the four measurements. Additionally, this subset outperforms the best subset choice for six variables in some criteria. This shows that the best subset approach can have a lower test error than other subsets. Thus, we have another reason why the best subset selection is not necessarily worth the effort.

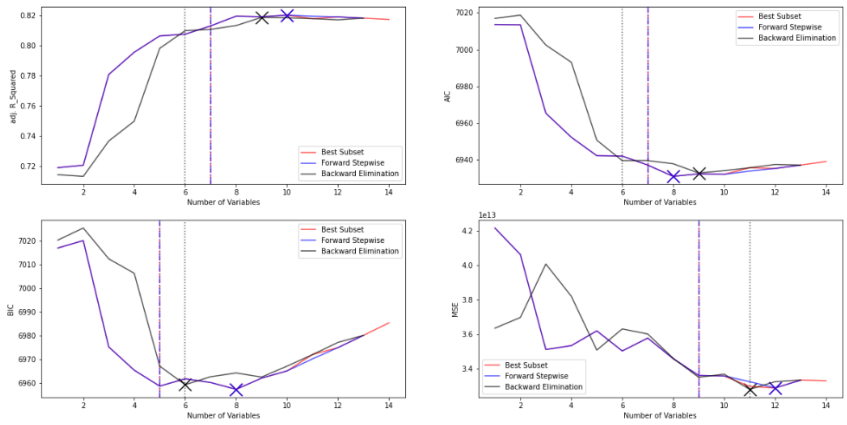

Figure 4.5.2 Each plot displays the values of different criteria applied to the selected subsets chosen by the three subset selection methods. The best subset selection is displayed in red, the forward stepwise selection in blue and the backward stepwise elimination in black. The crosses mark the optimal points of each criterion. The dashed lines represent the implementation of the One Standard error rule. (Source: Author's illustration)

When we compare the different criteria, we can see that the *BIC* penalizes the addition of unrelated regressors more than the *AIC*, as its values for nine or more variables have a steeper rise compared to the *AIC*. Therefore, the *BIC* usually selects smaller models. This is also the case in Fig. 4.5.1, as the *BIC* model chooses the smallest subsets.

In addition to the traditional Information criteria, a direct test error estimate is depicted on the bottom right, it is calculated by using Cross-Validation. This plot shows that the directly estimated test error seems to select larger subsets than the traditional criteria, which indicates that many variables in this dataset are relevant.

Figure 4.5.3 depicts the best subset selections on two created datasets with a mean of zero, a standard deviation of one and twelve unrelated regressors. We also split the data, so that 75% are used for the training dataset.

15 observations and 12 unrelated predictors

	number_of_variables	RSS	R_squared	variables
0	1	3.547795	0.370055	[8]
1	2	1.511968	0.731536	[8, 9]
2	3	0.763195	0.864487	[8, 9, 11]
3	4	0.377244	0.933017	[7, 8, 9, 11]
4	5	0.212006	0.962356	[1, 7, 8, 9, 11]
5	6	0.108424	0.980748	[0, 4, 5, 6, 7, 10]
6	7	0.029149	0.994824	[0, 2, 3, 8, 9, 10, 11]
7	8	0.002092	0.999629	[0, 3, 4, 5, 6, 7, 10, 11]
8	9	0.000029	0.999995	[0, 3, 4, 5, 6, 7, 9, 10, 11]
9	10	0.000000	1.000000	[0, 1, 3, 4, 5, 6, 7, 8, 9, 11]
10	11	0.000000	1.000000	[0, 1, 2, 3, 4, 6, 7, 8, 9, 10, 11]
11	12	0.000000	1.000000	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

150 observations and 12 unrelated predictors

	number_of_variables	RSS	R_squared	variables
0	1	93.852560	0.114785	[1]
1	2	92.307680	0.129356	[1, 9]
2	3	90.349466	0.147826	[1, 9, 10]
3	4	88.620445	0.164134	[1, 5, 9, 10]
4	5	87.754329	0.172303	[1, 5, 9, 10, 11]
5	6	86.720168	0.182057	[1, 5, 8, 9, 10, 11]
6	7	86.146696	0.187466	[1, 3, 5, 8, 9, 10, 11]
7	8	85.785405	0.190874	[1, 3, 5, 7, 8, 9, 10, 11]
8	9	85.552805	0.193068	[1, 2, 3, 5, 7, 8, 9, 10, 11]
9	10	85.383686	0.194663	[1, 2, 3, 5, 6, 7, 8, 9, 10, 11]
10	11	85.366069	0.194829	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
11	12	85.361460	0.194873	[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

Figure 4.5.3 Two dataframe outputs from the best_subset method. The input data is created with twelve unrelated variables and fifteen observations (Left) as well as twelve unrelated variables and 150 observations (Right). (Source: Author's illustration)

A problem we can see in Figure 4.5.3 is that the R^2 and *RSS* values seem to be exceptionally good for fifteen observations, therefore we could conclude to select one of the models and expect really good results. However, we know that none of these variables is in fact related to the response, therefore these measurements are not adequate. Moreover, in the right table we have more observations, resulting in lower dimensional data and more accurate predictions.

Figure 4.5.4 visualizes the estimated test error analogically to Figure 4.5.2, however, we use the left dataset of Figure 4.5.3. Here, we can see that all three of the traditional measurements indicate that large models have the best prediction power, while the actual direct test error estimate selects smaller subsets. Because we know that there is no related regressor, we can see that the directly estimated test error provides us with the most accurate result, as its overall lowest values are close to zero.

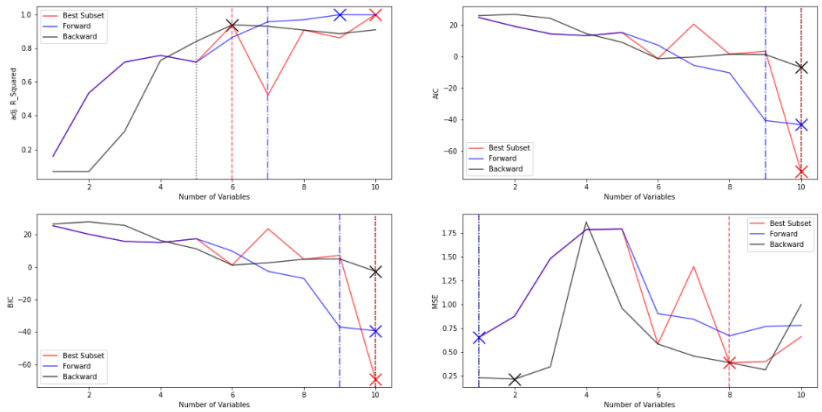

Figure 4.5.4 The plots are constructed equally to Figure 4.5.2, but in this case the underlying data is created to fit certain criteria. (Source: Author's illustration)

4.6 Shrinkage

The regularized shrinkage methods discussed in Chapter 3, are imported with the python module *sklearn.linear_model* as *Lasso* and *Ridge*. These are then implemented into the method *best_lamda*, which is shown in Fig. 4.6.1. This method receives a list, containing all values that are going to be set as λ in the *Ridge* or *Lasso* function. Additionally, we input the training and testing DataFrames and target variable(s) name(s). The method we want to use, either '*ridge*' or '*lasso*', also has to be declared. Then we calculate the shrunken coefficients for each given λ and determine the test *MSE* estimate. Finally, we create a DataFrame object that contains the coefficient values of each model, additionally, we also output a previously estimated minimal test *MSE* value and its corresponding λ value.

```python
def best_lambda(lambda_list,df_train,df_test,output='pred',methode='ridge'):
    '''Method to find the minimal lambda value for ridge or lasso'''
    #defining the regressormatrix and target variable for the training and testing set
    Xtr,ytr=df_train[df_train.columns.difference([output])],df_train[output]
    Xtst,ytst=df_test[df_test.columns.difference([output])],df_test[output]
    #creating empty lists, which are later used to save the results
    mse_list,r_squared_list,paramet=[],[],[]
    #iterating over all elements in the given lists (contains all values we want to check for the min lambda)
    for a in lambda_list:
        #performing the lasso or ridge regression (dependent on the choice at the method call)
        if (methode=='lasso'):
            model=Lasso(alpha=a,normalize=True).fit(Xtr,ytr)
        elif (methode=='ridge'):
            model=Ridge(alpha=a,normalize=True).fit(Xtr,ytr)
        #determining and saving the test error, rsquared and model coefficients
        r_squared_list.append(model.score(Xtr,ytr))
        mse_list.append(m(ytst,model.predict(Xtst)))
        paramet.append(model.coef_)
    df_p=pd.DataFrame(paramet)
    df_p.columns=Xtr.columns
    #calculating the minimal mse value and its asociated lambda position
    min_mse= min(mse_list)
    lambda_pos= lambda_list[mse_list.index(min_mse)]
    #returning the minimal mse value and its position,as well as the mse and r-squared lists
    #and the dataframe,which contains all coefficients and their values for each lambda-value-model
    return min_mse,lambda_pos,mse_list,r_squared_list,df_p
```

Figure 4.6.1 This method computes the lasso or ridge regression on a given set of λ values. It also calculates the estimated mean squared test error estimate for each λ value. (Source: Author's illustration)

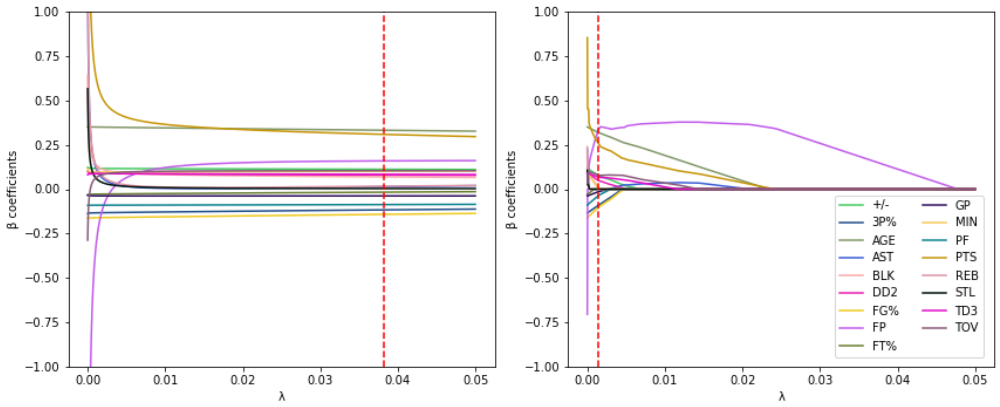

Figure 4.6.2 The two plots represent the β coefficient values of the ridge (Left) and lasso (right) regression for $\lambda \in [0.0,0.05)$. The underlying data used is the z-score standardized NBA data set. (Source: Author's illustration)

Figure 4.6.2 shows that both methods shrink the coefficients towards zero, but we can already see that the lasso model performs variable selection as many coefficients equal zero at a certain point. This is not the case for the ridge regression, where the coefficients never equal zero. The dashed red lines represent the selection of λ with the lowest test error estimate.

In Figure 4.6.3 we specifically perform fifteen different splits of a created data set, containing five related and twelve unrelated variables with fifteen observations and a mean of zero as well as a standard deviation of one. Each split uses 75% of the original data for the training data set. The reason for having several train and test splits is that different splits can create different results, for example different variable selections and different prediction accuracies. By having multiple different splits, we can get a more general performance outlook.

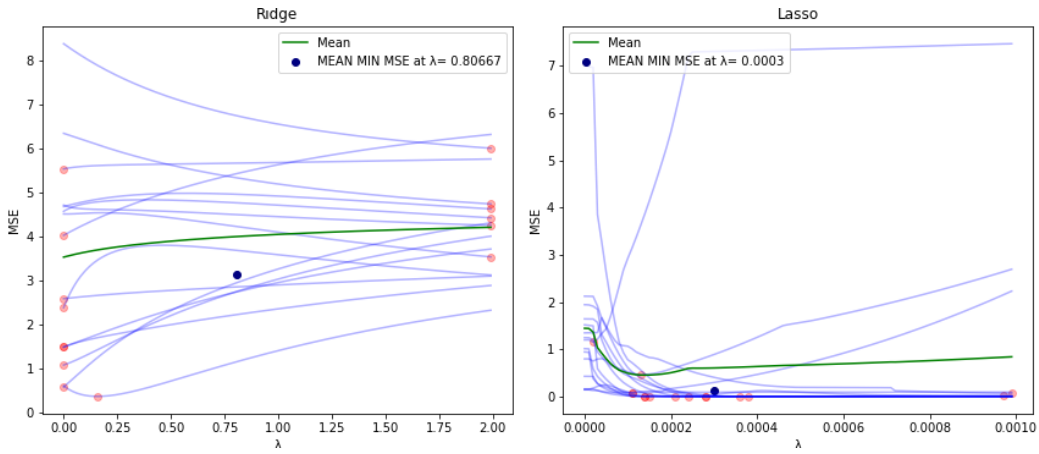

Figure 4.6.3 Both plots display the estimated mean squared test error values in blue for fifteen data splits. The red points mark the minimal estimated test mean squared error for each split. The mean estimated test mean squared error distribution is displayed in green. The blue point marks the mean minimal estimated test mean squared error. (Source: Author's illustration)

The first notable difference between these two methods is that the lasso regression has evidently smaller optimal λ values and smaller test *MSE* estimate values on average than the ridge regression. Moreover, the lasso regression is better than the ridge regression in cases, where we have few truly related and many truly unrelated variables. However, the ridge regression is better in cases where we have many truly related variables with coefficients roughly the same sizes. Consequently, we do not know the true relations beforehand, which means that none of these two methods is undeniably better than the other.

4.7 Comparison and results

We compare the four methods with the standardized NBA data set, the best subset selection is not included because of its computational unfeasibility. The comparison is made with fifteen different 70% training data splits, in order to get a more general outlook. Furthermore in Fig. 4.7.1 we can see that the lasso model selects many variables in all fifteen splits, while the backward elimination and forward selection both choose smaller models. It's also visible that all shown models have a similar mean test error estimate, moreover the largest estimated test error difference of about 3.33% is between the forward stepwise selection and the lasso regression. In addition, the regularization methods include more parameters than the stepwise methods on average. Obviously, this is always the case for the ridge regression, as it is unable to perform variable selection. However, the lasso regression also implements many non-zero variables, in this case leading to the worst test error estimate average of the four models. The regressors that are included in more than 50% of the splits across all models are: *+/-*, *3P%*, *TD3*, *PTS*, *PF*, *FG%*, *BLK* and *AGE*.

	AGE	FP	FG%	PF	PTS	3P%	+/-	TD3	BLK	TOV	REB	MIN	GP	STL	AST	DD2	FT%	mean min MSE	mean number variables
Backwards	15	14	11	10	9	9	8	8	8	6	5	4	4	4	4	1	1	0.372180	8.066670
Forwards	15	6	10	9	15	10	9	9	10	3	10	1	2	6	9	2	1	0.369060	8.466670
Ridge	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	0.369841	15.000000
Lasso	15	15	15	15	15	15	15	15	15	14	14	14	15	13	11	12	13	0.381770	14.176471

Figure 4.7.1 The first seventeen columns of this DataFrame object resemble the count of how often each variable is included in each model's minimal estimated test MSE for 15 different data splits. The last two columns show the overall mean of the minimal estimated test MSE and the mean of the included variables at the minimal estimated test MSE point. (Source: Author's illustration)

5 Conclusion

At first, this seminar thesis introduced the different subset selection methods in chapter two. Here, we have talked about the best subset and stepwise methods, additionally we have discovered that the best subset approach is not efficiently applicable when a lot of regressors are included. Therefore, the stepwise methods are generally implemented more often, here we have left out the hybrid methods, which can make forward and backward steps. In the next chapter we have discussed two shrinkage methods that try to decrease the prediction error by reducing the coefficient estimates. An outlook on this topic is the elastic net, which combines the lasso and ridge regression. Therefore, the elastic net is widely present in variable selection tasks, as we do not have to choose between the ridge or lasso regression. The last chapter has dealt with the implementation of the previously introduced methods into python applications. Moreover, we have used a real dataset from the NBA, which indicated the difference in behaviour of our implemented variable selection processes. In the end we have received variables that were selected in most models, additionally, we have also compared the performance of each different method. However, the field of dimension reduction methods has been left out, which includes for example the Principal Components Regression. These methods reduce the dimensionality by transforming the original regressor matrix X with linear combinations into a new regressor matrix Z, which has smaller dimensions.

6 Bibliography

Choueiry, G., 2019. *Quantifying Health.* [Online]
Available at: https://quantifyinghealth.com/stepwise-selection/
(Accessed: 12 January 2022).

Dinov, I. D., 2018. *Data Science and Predictive Analytics.* 1st edition. Cham,Switzerland: Springer International Publishing.

James G., W. D. H. T. T. R., 2021. *An Introduction to Statistical Learning.* 2nd edition. New York, NY: Springer.

Nair, A., 2019. *analyticsindiamag.* [Online]
Available at: https://analyticsindiamag.com/r-squared-vs-adjusted-r-squared/
(Accessed: 13 January 2022).

Oleszak, M., 2019. *towardsdatascience.* [Online]
Available at: https://towardsdatascience.com/a-comparison-of-shrinkage-and-selection-methods-for-linear-regression-ee4dd3a71f16
(Accessed: 14 January 2022).

Weisberg, S., 2014. *Applied Linear Regression.* 4th edition. Hoboken, New Jersey: Wiley.

Zach, 2020. *Statology.* [Online]
Available at: https://www.statology.org/stepwise-selection/
(Accessed: 14 January 2022).

Erklärung

Ich versichere wahrheitsgemäß, die Arbeit selbstständig verfasst, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde sowie die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Karlsruhe, den 15. Januar 2022