Assignment 3

Assignment 3 - Linear Regression and LASSO

1

Modeling fat content of meat using lasso In this assignment we are trying to use absorbance data to predict the Fat content in meat. We initially assume that Fat can be modeled as a linear regression where absorance on 100 different channels are used as features. The underlying probabilistic model can be expressed as:

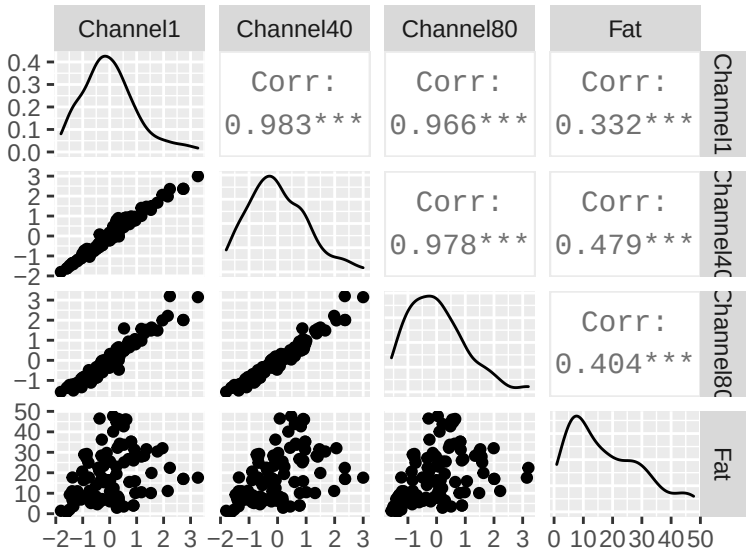
$$p(y|\boldsymbol{x}, w, \sigma^2) \sim N(w^T\boldsymbol{x}, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma}} \exp - \frac{(y - w^T\boldsymbol{x})^2}{2\sigma^2}$$

which then can be estimated using a maximum likelihood estimate. Here, $\boldsymbol{x}$ are the absorbance channels 1-100 and w their respective parameters to be estimated.

To report error rates throughout the report, the root mean squared error (RMSE) is used, defined as:

$$RMSE(Y, \hat{Y}) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2}$$

It is worth noting that the features are highly correlated, signalling multicollinearity might be a problem. As we now also have more features than observations, the linear regression model will probably not perform very well. Both features and target exhibit a skewness to the left.



Fitting linear regression The data has been split into test and training sets (50/50) and standarized and then a linear regression model has been fitted.

The RMSE of the training data:

```
## [1] 0.0755587
```

The RMSE of the test data:

```
## [1] 21.54814
```

Errors are extremely low for the training set and much higher the test set, indicating that the model is highly overfitted on the training data. The most evident cause of this is that there are more parameters than observations and that the features are highly correlated. *See appendix for code to plot training, test and target values.*

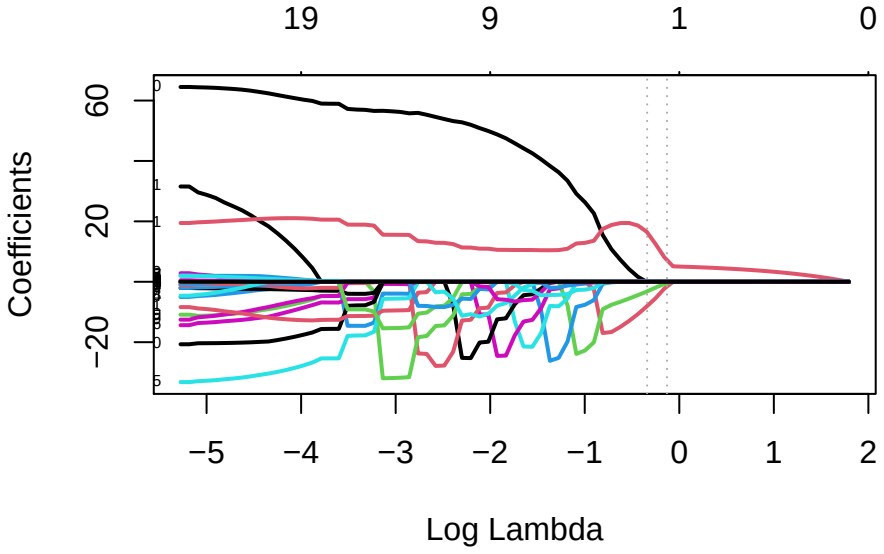
2

The objective function that should be optimized is (see Hastie et al.):

$$\hat{w}^{lasso} = argmin \left\{ \frac{1}{2} \sum_{i=1}^{N} (y_i - w_0 - \sum_{j=1}^{p} w_j x_{ij})^2 + \lambda \sum_{j=1}^{p} |w_j| \right\}$$

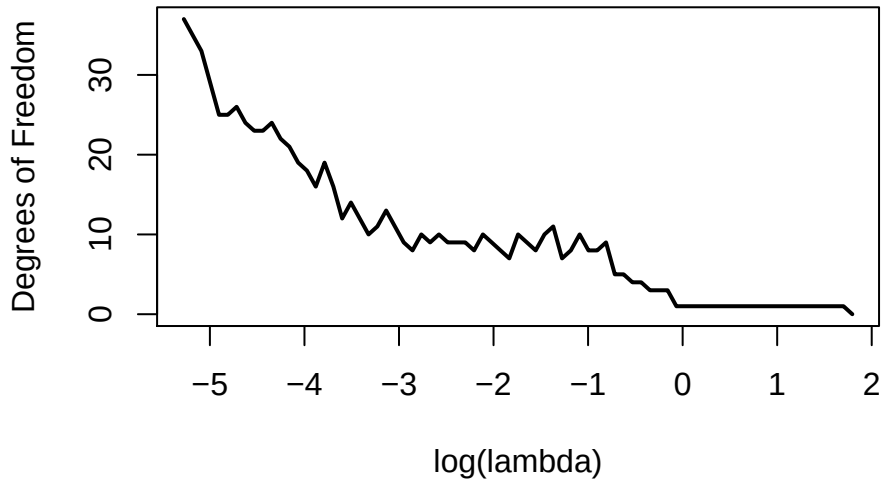
3

Fitting the lasso After fitting a lasso regression model using glmnet with an alpha-setting of 1, we can produce the following plot which shows how the coefficients vary with different levels of log lambda. The higher the penalty factor, the fewer coefficients that are non-zero. Should we want only three predictors, we can choose a log lambda corresponding to a region where there are only three non-zero coefficients (approx [-0.43,-0.13]). Interestingly, coefficients tend to go to zero and then reappear as another coefficient is eliminated. This supports our earlier finding that the multicollinearity is prominent. The variance inflation factor could be calculated to determine the severity of the problem.



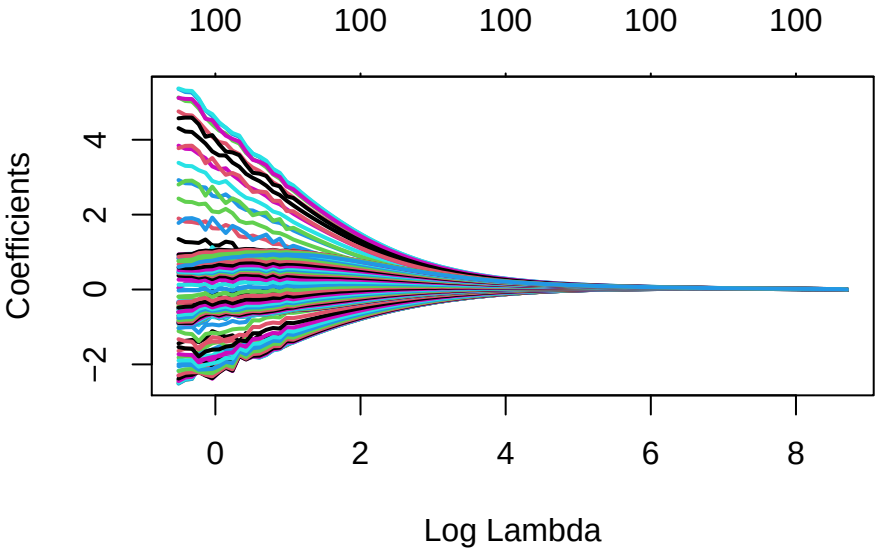
4

Degrees of Freedom The plot below shows how the degrees of freedom decreases with higher values of the regularization parameter lambda. This coincides with the number of non-zero coefficients being an exact unbiased estimate of the degrees of freedom, i.e. higher lambda means fewer non-zero components and therefore a lower degrees of freedom. The definition of the degrees of freedom should you wish to calculate it is: $df(\hat{y}) = \frac{1}{\sigma^2}\sum_{i=1}^{N} Cov(\hat{y}_i, y_i)$



5

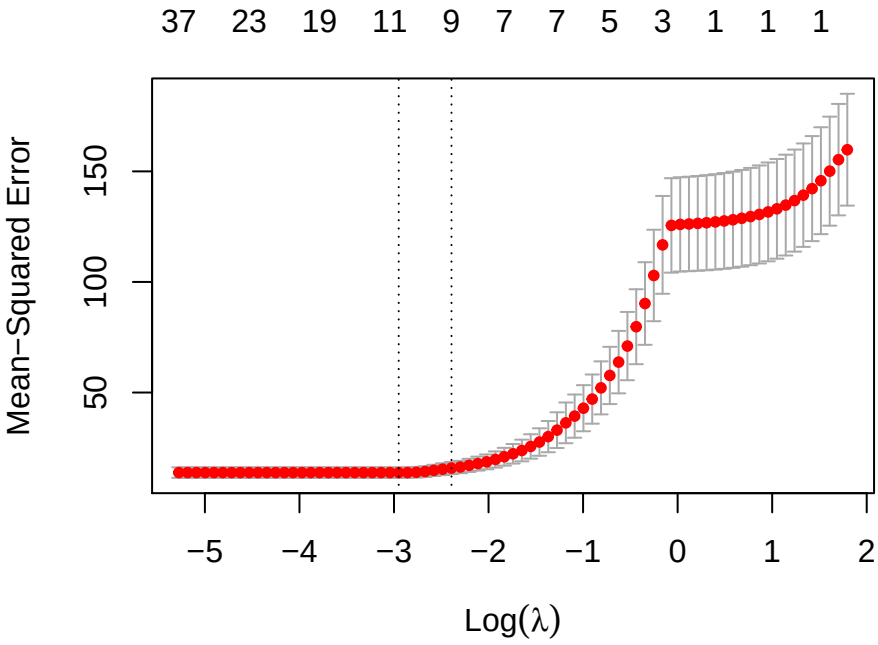
Fitting ridge model Repeating step 3 but with ridge regression gives a very different result:



It seems the ridge regression, unlike lasso, does not shrink any coefficients to zero. We are in other words left with the same number of features.

6

Cross-validation Here, cross-validation is used to find the optimal lasso model. See how the error increases when a too high log-lambda is used.



It can be determined from the plot above that the optimal value (left line) is not statistically significantly better than log lambda of -2 by looking as the error bars just so slightly overlaps. One could also extract the standard errors and intervals from the cross-validation object itself and make the calculations.

```
# 3.6 - number of non-zero coefficients in optimal model
crossval$nzero[match(crossval$lambda.min, crossval$lambda)]
```

```
## s51
##   9
```

```
# minimum log lambda
log(crossval$lambda.min)
```

```
## [1] -2.949955
```

```
# one standard error from the minimum log lambda
log(crossval$lambda.1se)
```

```
## [1] -2.391753
```

The optimal number of parameters are nine, plus the intercept. See the selected model below:

```
##                Coef.
## (Intercept)  17.899
## Channel13   -31.768
## Channel14    -3.921
## Channel15    -5.542
## Channel16    -0.743
## Channel40    56.224
## Channel41    15.547
## Channel50     0.000
## Channel51    -9.464
## Channel52   -15.280
```

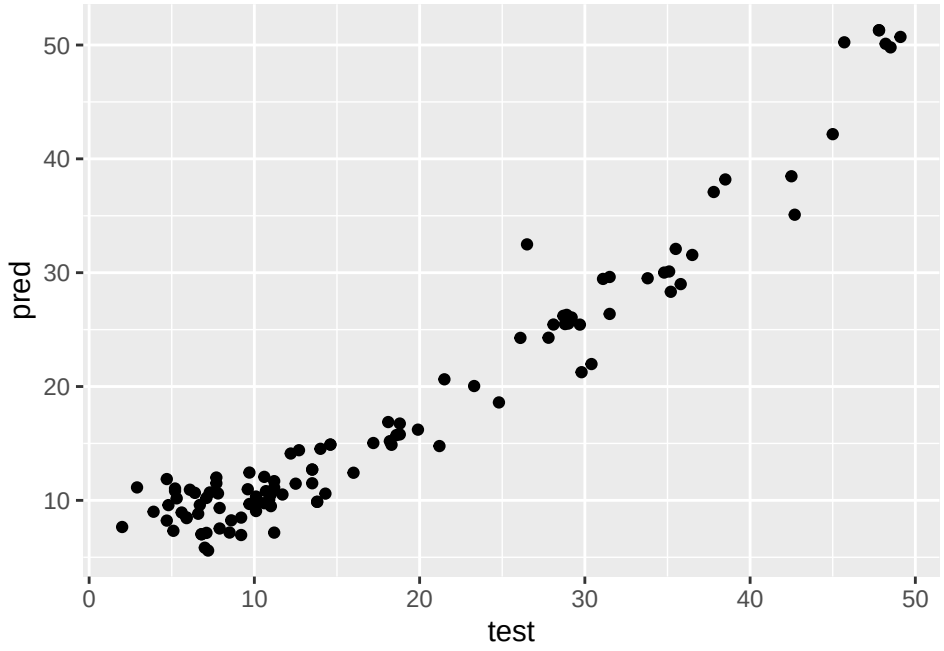
Note that the coefficient for Channel50 is very small, but non-zero.

It is possible to obtain the errors of fitted values for the training data:

```
# 3.6

# RMSE of the fitted values
sqrt(sum((y_train-as.vector(train_pred))^2) / length(y_train))
```

```
## [1] 3.330301
```

Optimal model The optimal model used with the test features shows a quite strong linear relationship. Here are the fitted values of the test data plotted against the original target of the test data:



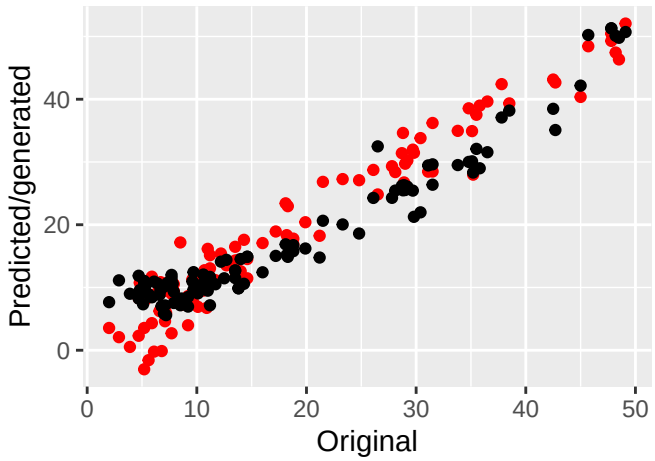
The error of the test set is given below. The difference from the error of the training data is marginal, which indicates that the model is not overfitted.

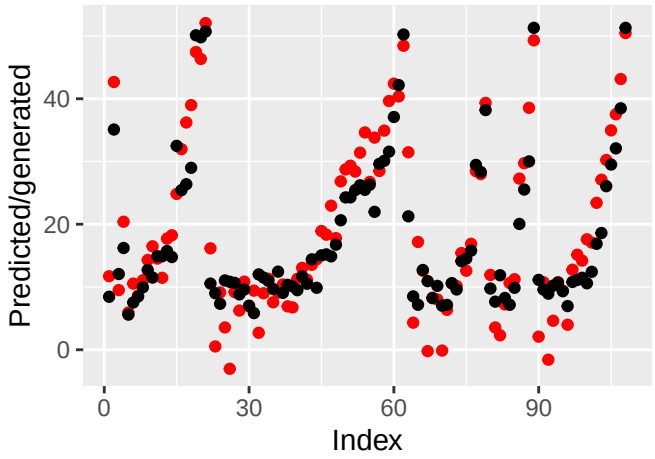
```
# 3.6

# Error of test
sqrt(sum((y_test-as.vector(lasso_test))^2) / length(y_test))
```

```
## [1] 3.537068
```

7

Generating new target values Here new data is generated from the feature values of the test data and plotted against the original target values in the test data.





Newly generated points are marked in red above. Black dots are the predicted values. The match seems decent, but is not terribly exciting since we have basically used the predicted values and added some noise.