Ankit Jadav, ankitjadav94@gmail.com
Lucas Lucian, lucaslucian471@gmail.com
Fredrik Svanström, fresva16@student.hh.se

Estimating the cetane number of diesel fuel

1. Introduction

The two dominating types of internal combustion engines today are the Otto[1] and Diesel[2] engines.

The Otto engine runs on petrol or alcohol and ignites the fuel-air mixture by means of a spark plug. The Diesel engine, as one might expect, runs on diesel fuel. Here the fuel-air mixture is ignited by the high temperature resulting from the piston compressing the air inside the cylinder.

Since we don't want any combustion in the cylinder of an Otto engine before the spark plug is activated, we need a fuel that has some resistance to unintended ignition caused by compression. This resistance of the petrol is specified by its octane number. The higher the octane number the more resistant it is to unintended ignition, sometimes known as knocking. Normal petrol for cars have an octane number of around 100.

In the Diesel engine, on the other hand, we strive for a fuel igniting from compression. The willingness of the diesel fuel to ignite and burn is specified by its cetane number. Normally the cetane number for diesel fuel is around 50 but can sometimes be as low as 40. Premium diesel has a cetane number closer to 60.

The task in this project is to estimate the cetane number of samples of diesel fuel using the near infrared spectrum (NIR-spectrum). We are given two sets of measured NIR-spectrum and for one of the sets we also have access the actual cetane number. Using some common supervised learning regression methods we will try to predict the cetane numbers of the sample set for which the cetane values are unknown to us. We also try to predict the correctness of our predictions i.e. the estimated generalization errors.

Doing this project we have used Matlab. Both the app called Regression learner from the Statistics and Machine learning toolbox and the Neural Net fitting-app from the Neural Net toolbox are employed. We also utilized the standard Matlab script editor for writing m-files.

Besides this report we also hand in our prediction of the unknown cetane numbers from a Principal Component Analysis (PCA) linear regression model, a Partial Least Squares (PLS) linear model and a Multi-layer Perceptron (MLP).

All Matlab-scripts are made available so that the analysis can be repeated.

[1] Nicolaus Otto, 1832-1891
[2] Rudolf Diesel, 1858-1913

Ankit Jadav, ankitjadav94@gmail.com
Lucas Lucian, lucaslucian471@gmail.com
Fredrik Svanström, fresva16@student.hh.se

2. Methodology

The techniques used in this projects are all supervised learning regression methods. We use both script based m-files and the matlab apps available to predict the cetane numbers of the NIR-spectrum samples given in the data set *cnTestX*.

To interpret the data it is firstly imported into the Matlab workspace and then a principal component analysis (PCA) is performed on the data set *cnTrainX* using both the matlab command `pca` in the script-environment and in the regression learning app. As we can see in Fig. 1 the app lets us directly chose either the explained variance level or the number of components that the linear regression model will use.

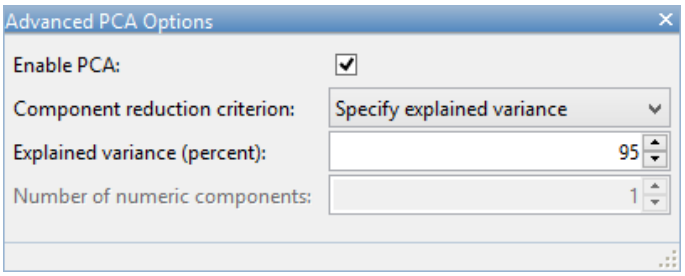

Fig. 1: PCA options in the Regression learner app

Using the `pca` command followed by `explained` we can also get information about the percentage of the total variance explained by each principal component. This information is also available in the app.

Next we train a linear model using the regression learner app. The app will output the resulting code so that we can use the trained model to make predictions.

We also implement the linear regression model directly in the Matlab script-environment. In both the implementations we use a 5-fold cross validation to estimate the generalization error of the predictions.

We also try to predict the cetane number of the *cnTestX* using another linear model called partial least squares (PLS). With this we don't have to do any principal component analysis before training the model. We note that in the project description there is a reference to the PLS toolbox. This is not something that we access to and neither is the PLS method available in the regression learner app, so we implement this part solely in the script-environment of Matlab.

A crucial distinction of the PLS is that it also takes advantage of the response of the training data, in this case the data set *cnTrainY*, to construct the set of linear combinations used in the linear regression model. An important part of our implementation is choosing the number of PLS-components to use.

Ankit Jadav, ankitjadav94@gmail.com
Lucas Lucian, lucaslucian471@gmail.com
Fredrik Svanström, fresva16@student.hh.se

Finally we implement a principal component multi-layer perceptron (MLP). Doing so we try to find optimal values for the number of units in the hidden layer and PCA-components to use. This done in both the script-environment and the neural net fitting-app from the neural net toolbox, however since the script-based solution runs the same algorithm as the app only the results from the script-based implementation is included in this report.

To get insights in the methods we have referred to the course literature i.e.

- Introduction to machine learning / Ethem Alpaydin. — 2nd ed. The MIT Press.
- The Elements of Statistical Learning. Data Mining, Inference, and Prediction / Trevor Hastie, Robert Tibshirani, Jerome Friedman. - 2nd ed. Springer.

Ankit Jadav, ankitjadav94@gmail.com
Lucas Lucian, lucaslucian471@gmail.com
Fredrik Svanström, fresva16@student.hh.se

3. Data

The data sets that we are given consists of a total of 245 observations of the NIR-spectrum where every observation has 401 channels. For 112 of these observations we have no access to the correct cetane number.

The data sets are the following:
cnTrainX - 133 observations of NIR-spectrum for which we have the correct cetane number
cnTrainY - The correct cetane numbers of the 133 samples *cnTrainX*
cnTestX - 112 observations for which we don't have the correct cetane number

Looking at the given cetane number in *cnTrainY* we see that it has a range between 40.7 and 60. The mean value is 48.9. This is consistent with what we expect the range to be. All the cetane numbers of *cnTrainY* is plotted in Fig. 2 below.

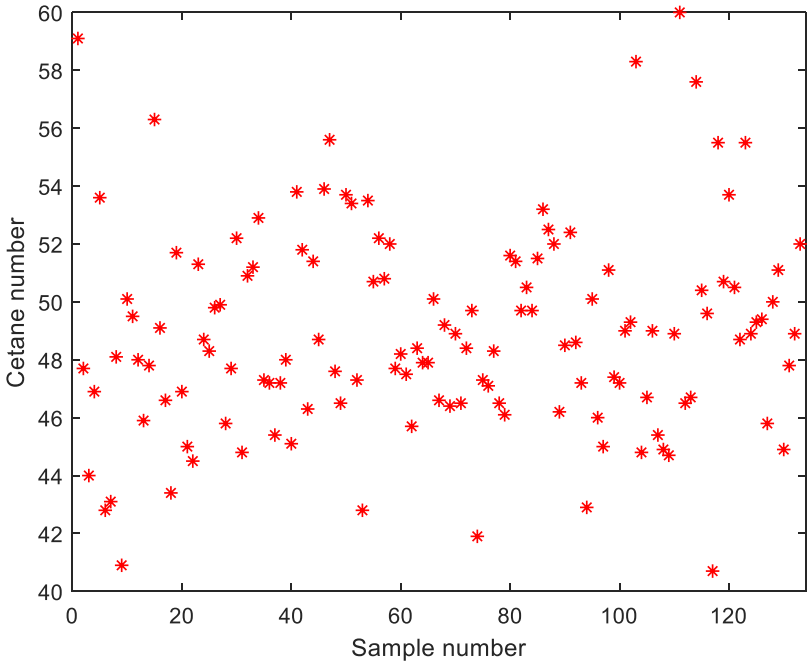

Fig. 2: Plot of the cetane numbers found in *cnTrainY*

Looking at the data found in *cnTrainX* i.e. the NIR-spectrum, we can point out that it is hard to do any intuitive analysis of the content since we have no specification of how the NIR-spectrum is measured and formed. The numerical values of the data in *cnTrainX* have a range of -0.0398 to 0.0618.

The range of the data in *cnTestX* is -0.0399 to 0.0601 and hence not significantly deviating from the range of *cnTrainX*.

Ankit Jadav, ankitjadav94@gmail.com
Lucas Lucian, lucaslucian471@gmail.com
Fredrik Svanström, fresva16@student.hh.se

4. Results

Looking at the result of the principal component analysis, we see that both the `pca` command and the app outputs the same values of variance explained per PCA-component. We show this in Fig. 3.

```
    86.2143    6.2911    4.7145    1.5258    0.4785    0.2105
```

PCA
PCA is keeping enough components to explain 95% variance.
After training, 3 components were kept.
Explained variance per component (in order): 86.2%, 6.3%, 4.7%, 1.5%, 0.5%, 0.2%

Fig. 3: The explained variance values of the 6 first components of the PCA-analysis from the script environment and from the regression learner app

As we can see in Fig. 3 we have chosen to set the PCA-options to keep enough components to explain 95% of the variance. This results in the three first components being kept. We also try using all values from just one and up to 15 components but can conclude that any improvement in estimated generalization error from using more than three is basically non-existing.

We also include the plot of the known cetane numbers of the test set and the first principal component in Fig. 4 below.

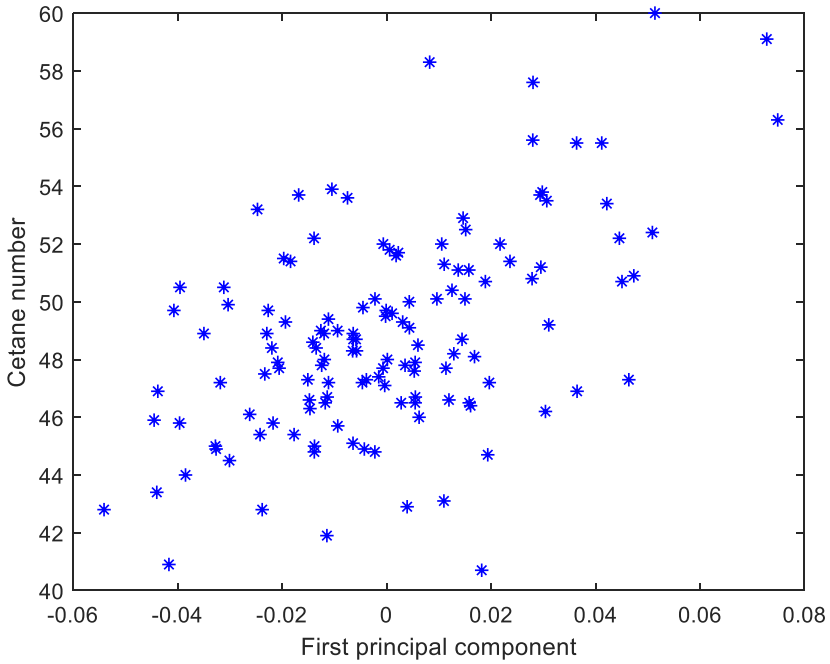

Fig. 4: Plot of the cetane numbers found in *cnTrainY* and the first principal component of the NIR-spectrum data set *cnTrainX*

As described we use the app to train a linear regression model to predict the unknown cetane numbers. The results from training the model is shown in Fig. 5. In this figure we get information about the mean squared error (MSE) and the root mean squared error (RMSE).

Ankit Jadav, ankitjadav94@gmail.com
Lucas Lucian, lucaslucian471@gmail.com
Fredrik Svanström, fresva16@student.hh.se

This will serve as our estimates of the generalization errors when we predict the unknown cetane numbers for the samples of the data set *cnTestX*.

Model 1: Trained

Results

RMSE	2.1539
R-Squared	0.63
MSE	4.6393
MAE	1.7479
Prediction speed	~400 obs/sec
Training time	4.5524 sec

Model Type
Preset: Linear
Terms: Linear
Robust option: Off

Feature Selection
All features used in the model, before PCA

PCA
PCA is keeping enough components to explain 95% variance.
After training, 3 components were kept.
Explained variance per component (in order): 86.2%, 6.3%, 4.7%, 1.5%, 0.5%, 0.2%

Fig. 5: The results from the linear regression model using the 3 most significant principal components.

In Fig. 6 we plot the true value of *cnTrainY* and the values predicted by the trained model. As we can see, for low values the prediction is a little high and for high values the prediction is a bit low. This is also confirmed if we look at the residuals as shown in Fig. 7.

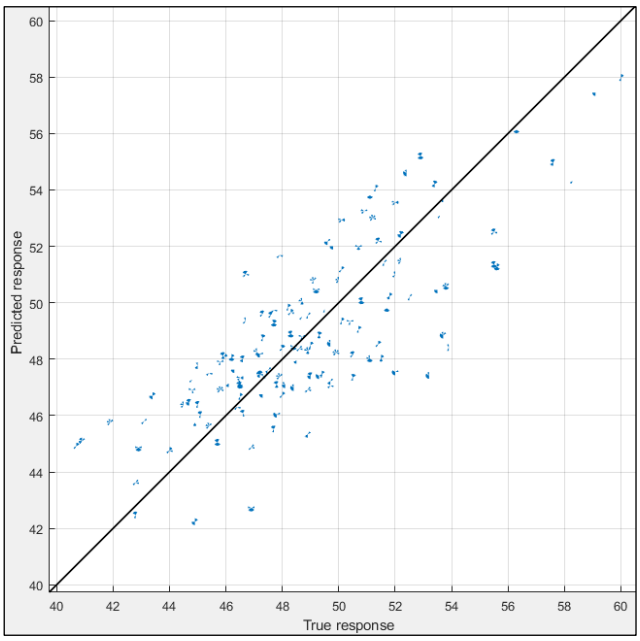

Fig. 6: Predicted and true cetane values of *cnTrainY using the regression learner app*

Ankit Jadav, ankitjadav94@gmail.com
Lucas Lucian, lucaslucian471@gmail.com
Fredrik Svanström, fresva16@student.hh.se

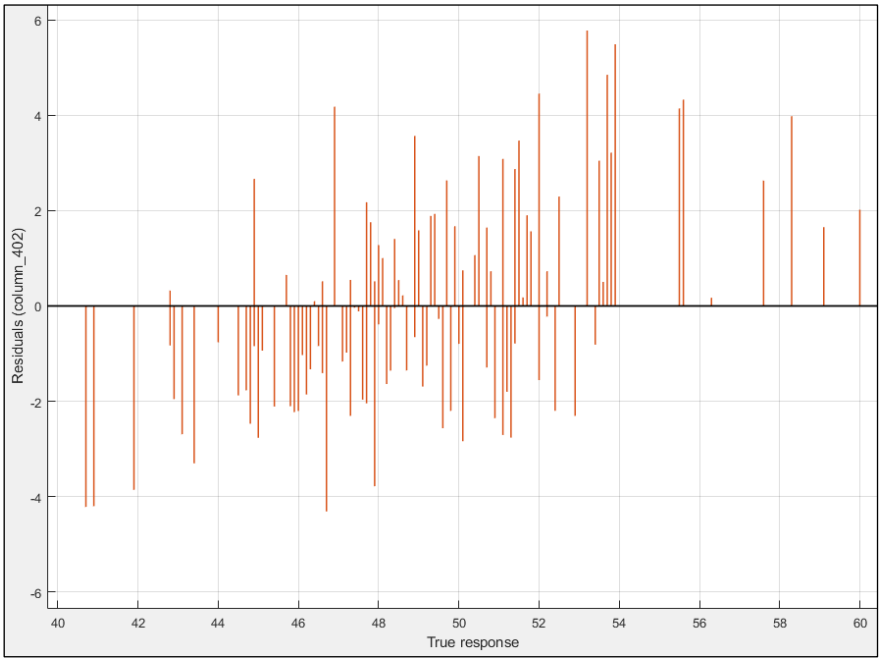

Fig. 7: Residuals of the predicted *cnTrainY* using the regression learner app

Since we didn't just use the app but also implemented the principal component analysis and the linear regression model in the Matlab script-environment, we can now compare the performance of the two methods.

Here we can also do a more profound analysis of how many PCA-components to use. In Fig. 8 we show the estimated generalization error using 5-fold cross validation and the training error for different numbers of components. Just as for the linear regression learner app we decide to use 3 PCA-components in the manual implementation.

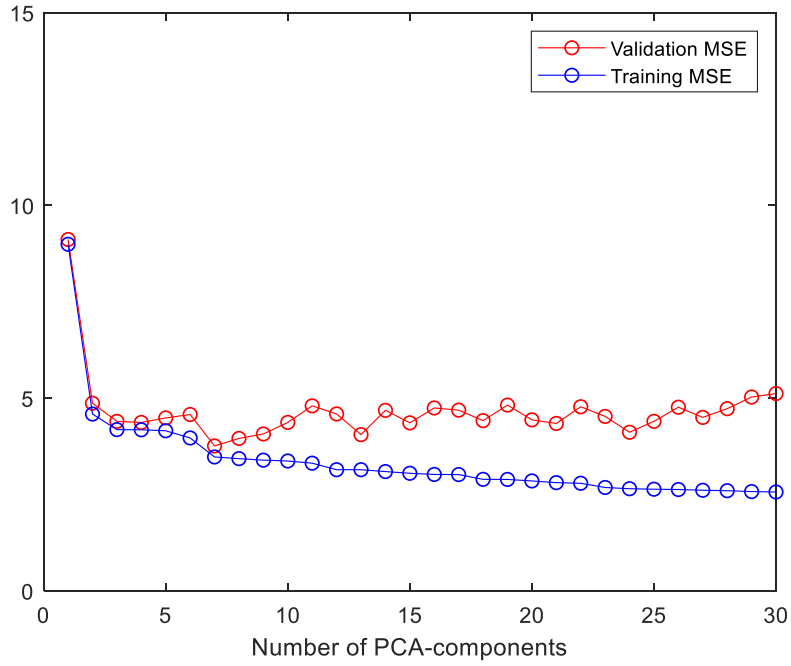

Fig. 8: Cross validation and training errors using different number of PCA-components

Ankit Jadav, ankitjadav94@gmail.com
Lucas Lucian, lucaslucian471@gmail.com
Fredrik Svanström, fresva16@student.hh.se

As for the app above we can compare the true and predicted values of the script based implementation. In Fig. 9 we also plot the residuals. By inspection we can see that Fig. 9 agree with Fig. 6 and Fig. 7 above.

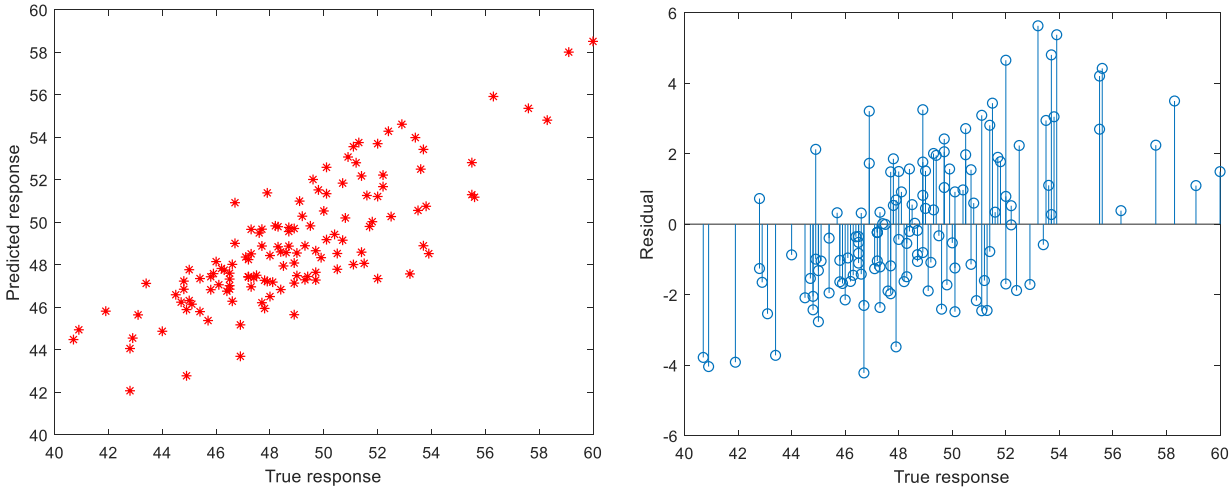

Fig. 9: True and predicted cetane numbers of *cnTrainY* using the 3 most significant principal components implemented in the Matlab script-environment

In the script-based solution we can also find good use of the `plotResiduals` command. In Fig. 10 we show the results of this.

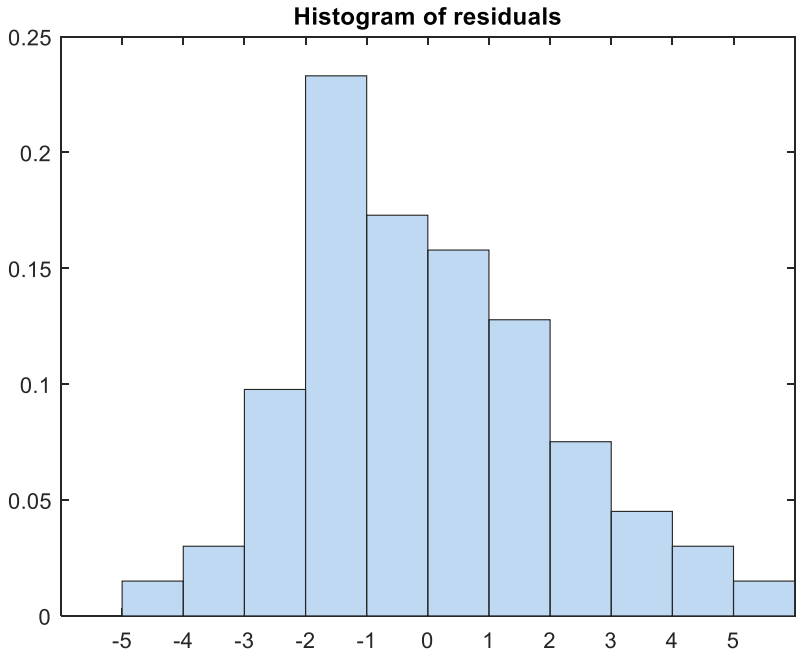

Fig. 10: Histogram plot of the residuals

Wrapping up the work with the PCA linear regression model, in Fig. 11 we plot the cetane number predictions for the 112 NIR-spectrum measurements of *cnTestX* (the data set for which the cetane number are unknown to us) using the model in the app (red stars) and from the Matlab script-environment (blue circles). Indeed we can see that they agree. We estimate that the generalization error of this prediction will have a MSE of 4.64 and a RMSE of 2.15.

Ankit Jadav, ankitjadav94@gmail.com
Lucas Lucian, lucaslucian471@gmail.com
Fredrik Svanström, fresva16@student.hh.se

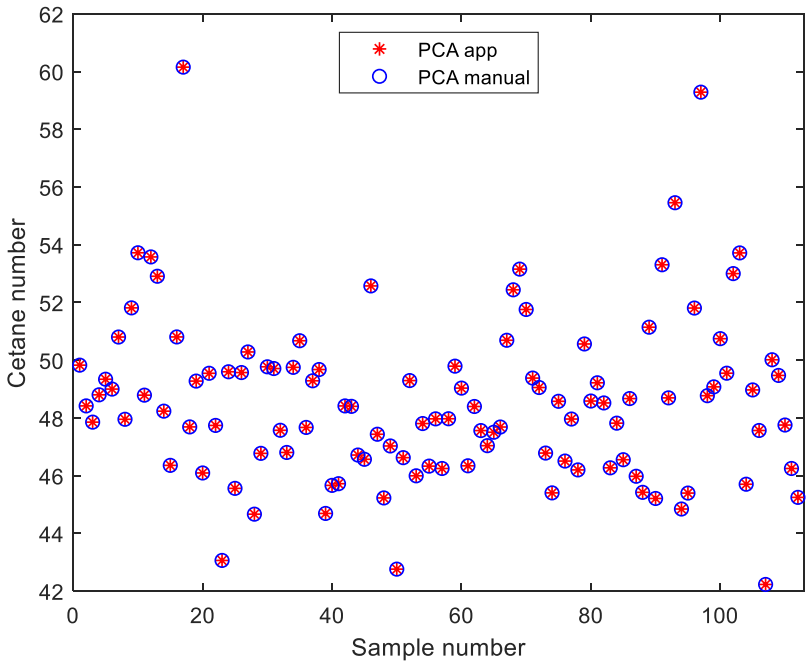

Fig. 11: Predictions of the cetane number for the 112 NIR-spectrum measurements of *cnTestX* using both the regression learner app and a script-based implementation

Continuing with the results from the partial least squares (PLS) implementation we firstly look at the training and validation errors as a function of the number of PLS-components used. This is shown in Fig. 12. Here we use 5-fold cross validation.

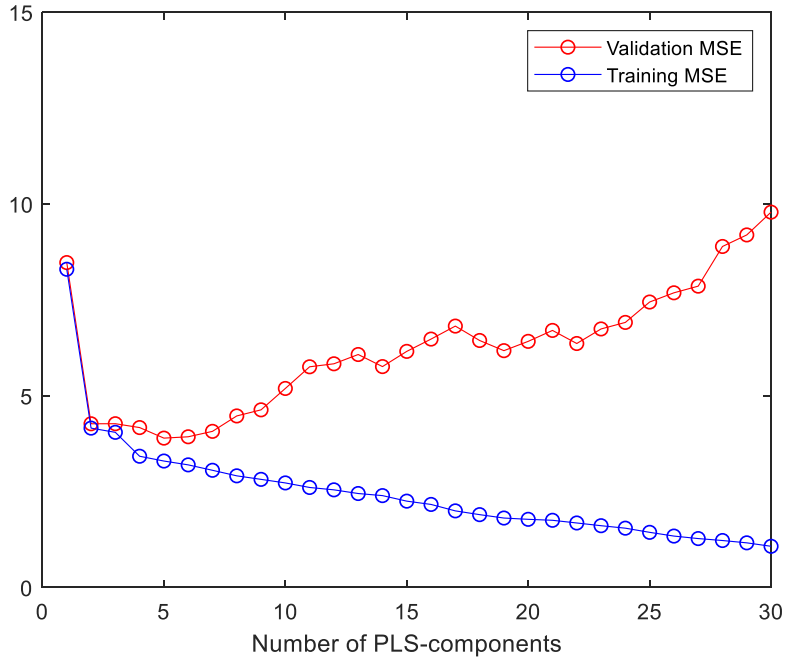

Fig. 12: Cross validation and training errors using different number of PLS-components

We can see that using 5 components seems to be a good choice to avoid overfitting.

Ankit Jadav, ankitjadav94@gmail.com
Lucas Lucian, lucaslucian471@gmail.com
Fredrik Svanström, fresva16@student.hh.se

Just as with the linear regression model using PCA we look at the true and predicted cetane values of *cnTrainY* in Fig. 13 below. As motivated above we use 5 PLS-components to do our predictions.

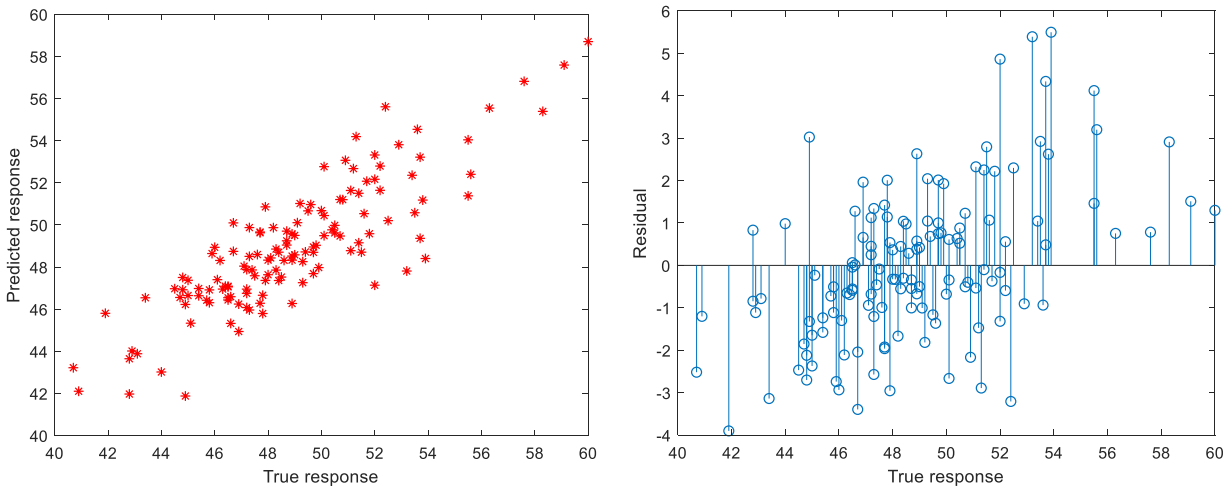

Fig. 13: True and predicted cetane numbers of *cnTrainY* using 5 PLS-components

Now we use this model to predict the cetane numbers for the 112 samples in *cnTestX*. This is shown in Fig. 14 as black crosses. To compare the results to the PCA linear model we have included the predictions from that part shown as before using red stars and blue circles. As we can see the predictions do not coincide for every sample but the difference is neither significantly large.

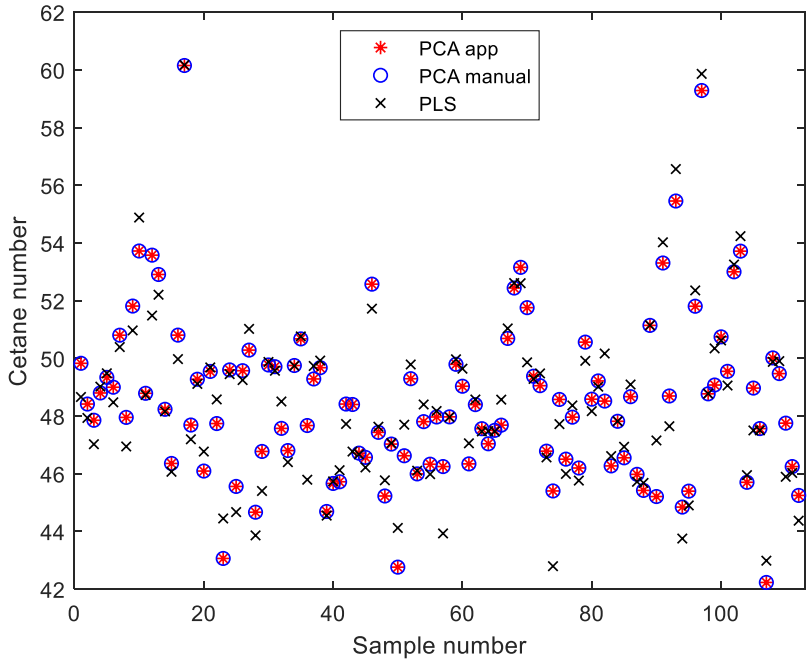

Fig. 14: Predictions made using 5 PLS-components for the cetane numbers of the 112 measurements of *cnTestX* marked as black crosses.

We estimate that the generalization error of this prediction will have a MSE of 4.02 and a RMSE of 2.01.

Ankit Jadav, ankitjadav94@gmail.com
Lucas Lucian, lucaslucian471@gmail.com
Fredrik Svanström, fresva16@student.hh.se

Finally we implement a principal component Multi-layer perceptron (MLP) to predict the cetane numbers of the data set *cnTestX*. This algorithm is also available in an app called the neural net fitting-app included in the neural net toolbox, however since the script-based solution will run the same algorithm as the app only the results from the script-based implementation is included in this report.

While running the script-implementation, Matlab will also show the progress of the training in the same way as in the app. We include an example of this in Fig. 15.

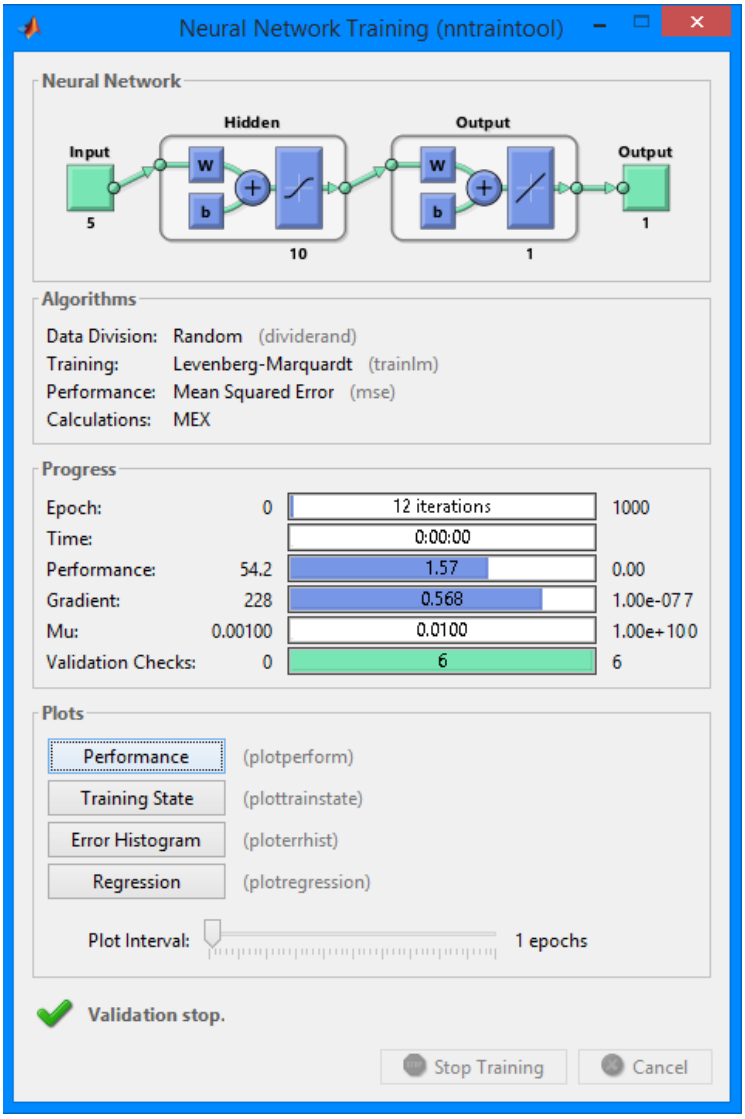

Fig. 15: Training the MLP in Matlab.

We try to optimize the choice of units in the hidden layer and the number of PCA-components to use. To do this we try all combinations of values of hidden units from 1 to 20 and PCA-components from 1 to 10. When doing this we see that the optimal number giving the lowest validation error is not the same every time. However running the code several time gives that a choice of 10 hidden units and 5 PCA-components appears to work well.

Ankit Jadav, ankitjadav94@gmail.com
Lucas Lucian, lucaslucian471@gmail.com
Fredrik Svanström, fresva16@student.hh.se

Just as for the other algorithms described in this report we look at the difference between the true and the predicted values of *cnTrainY*. This is shown in Fig. 16.

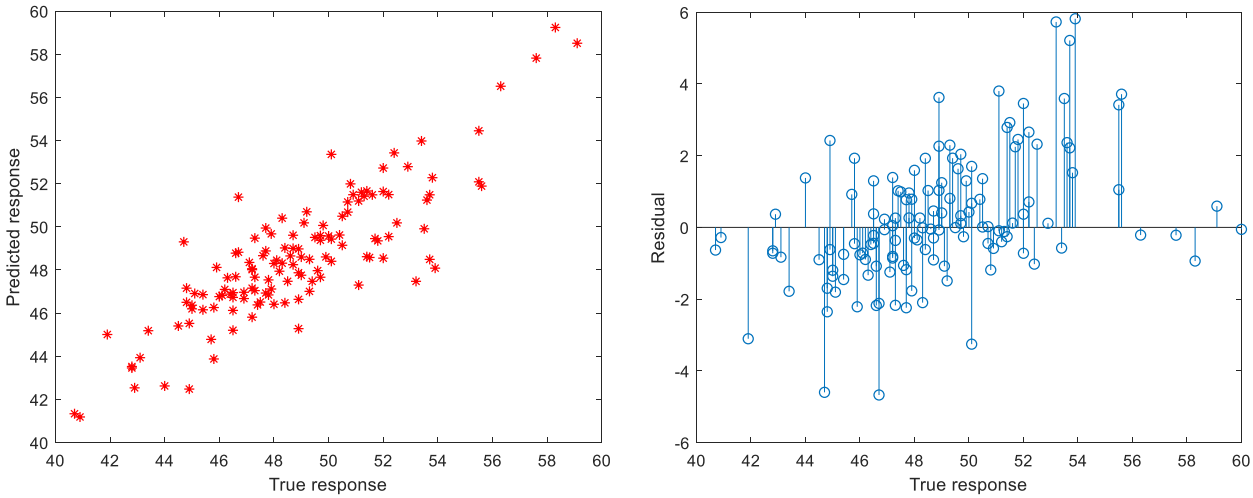

Fig. 16: True and predicted cetane numbers of *cnTrainY* using the MLP

We use this trained model to predict the cetane numbers of *cnTestX* and estimate that the generalization error should be MSE = 4.04 and RMSE = 2.01.

To summarize the predictions made by all the models we have included Fig. 17 below.

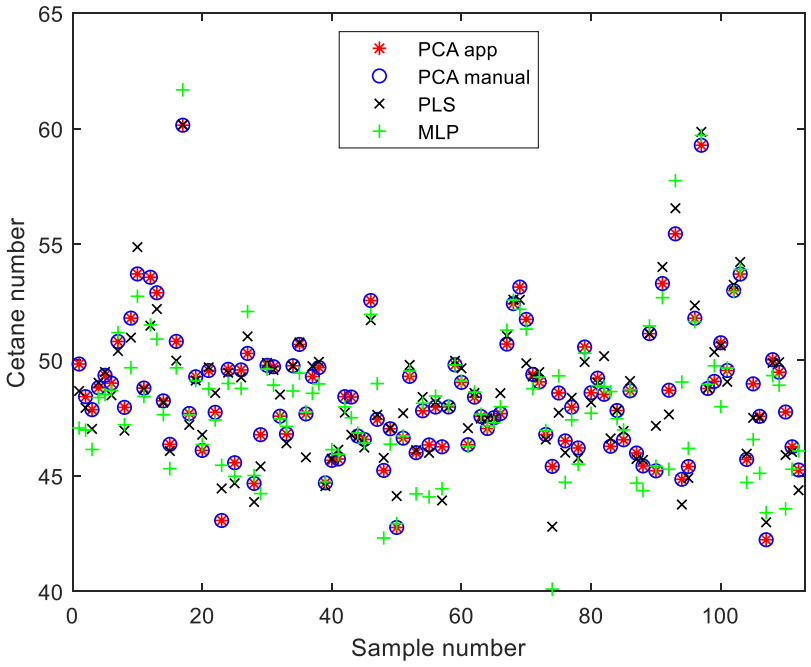

Fig. 17: Predictions of the cetane numbers of *cnTestX* made by all the implemented algorithms.

Ankit Jadav, ankitjadav94@gmail.com
Lucas Lucian, lucaslucian471@gmail.com
Fredrik Svanström, fresva16@student.hh.se

5. Discussion

In solving this task we have noticed that even though different algorithms are used to do the predictions, the results are very similar. This seen not only in the estimations of the generalization errors but also the actual predictions of the cetane numbers of the data set *cnTestX* as shown in Fig. 17 above.

Another thing that caught our attention is the difference in behaviour of the validation error curves of the PCA- and PLS-implementations. When we compare Fig. 8 and Fig. 12 we can clearly see that there is much more of "sweet spot" for the optimal value of components to keep when using the PLS-model. The number of components to keep in the PCA-case is not that obvious.

We also note that a traditionally script based software such as Matlab nowadays also have quite a lot of easy-to-use apps making machine learning more available. It can even easily be used to compare different models just by simply choosing to implement all available models at the same time. This is shown in Fig. 18 below.

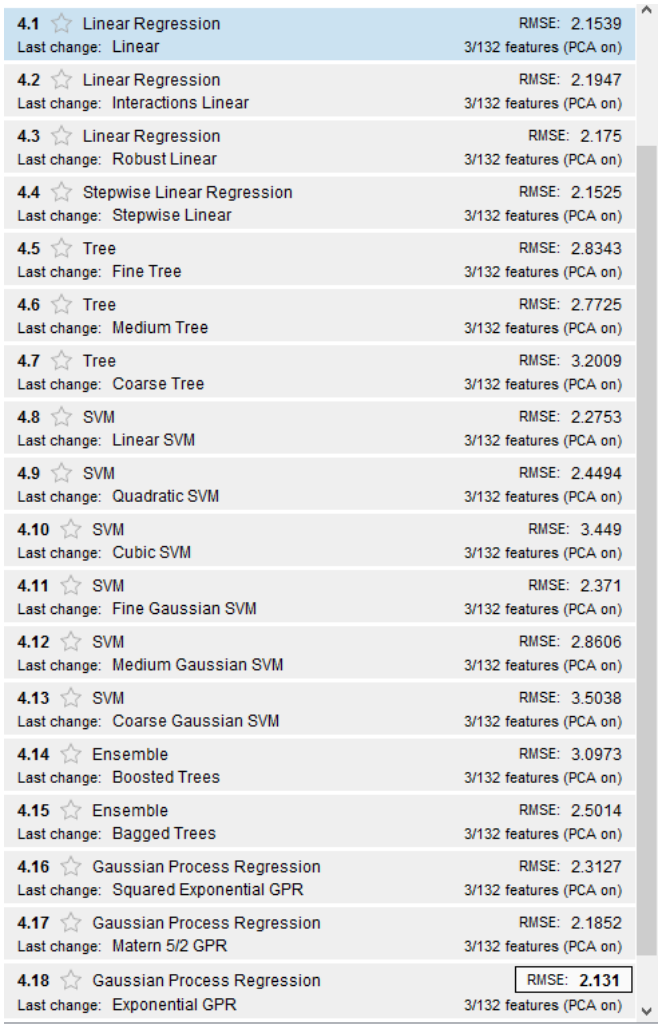

Fig. 18: Comparing different regression models using Matlab