HarvardX Data Science Capstone - CYO Project - Car testing

Manuel Diaz

December 19th, 2019

I. Introduction

This document describes my work on the projet of building a system to predict the time that a car manufacturer will spend in the quality control bench. The dataset contains an anonymized set of variables describing custom features of a Mercedes car and the time spent in testing them. It is available to the public in www.kaggle.com.

The final R code used to build the model and the prediction is provided in a separated file and it also available to the public in the GitHub repository manueldiaz50/CarTesting.git.

II. Method and Analysis

2.1 The Dataset

The dataset contains 4,209 observations each one corresponding to the time spent in the test of a car with different features. Present in the dataset are a unique identifier of the test, categorical and numeric fields indicating the car characteristics, and the time spent in the test.

Fields in the dataset

Field	Definition
ID	Unique identifier of the row
y	time spent in the test
X0 to X8	9 fields with categorical values
X10 to X377	368 numerical fields with values 0 or 1

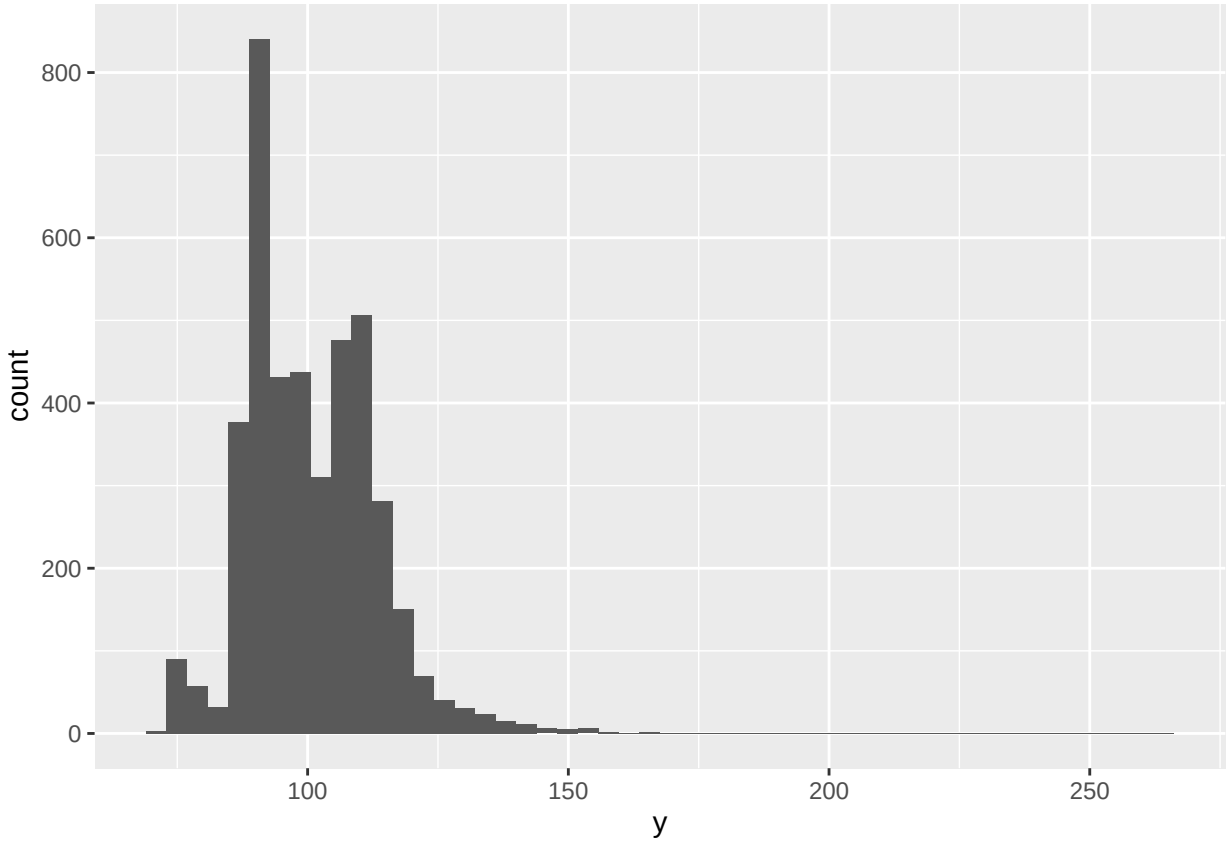
Below a few records as an example:

ID	y	X0	X1	X2	X3	X4	X5	X6	X8	X10	X11	X12	X377
0	130.81	k	v	at	a	d	u	j	o	0	0	0	1
6	88.53	k	t	av	e	d	y	l	o	0	0	0	0
7	76.26	az	w	n	c	d	x	j	x	0	0	0	0
9	80.62	az	t	n	f	d	x	l	e	0	0	0	0
13	78.02	az	v	n	f	d	h	d	n	0	0	0	0
18	92.93	t	b	e	c	d	g	h	s	0	0	0	1

To start, we split the dataset into two separated sets, *carTesting* and *validation*, with 90% and 10% of the data respectively sampled randomly from the original dataset. We will use *carTesting* to build and train the model and *validation* to measure the final performance.

2.2 Exploratory analysis

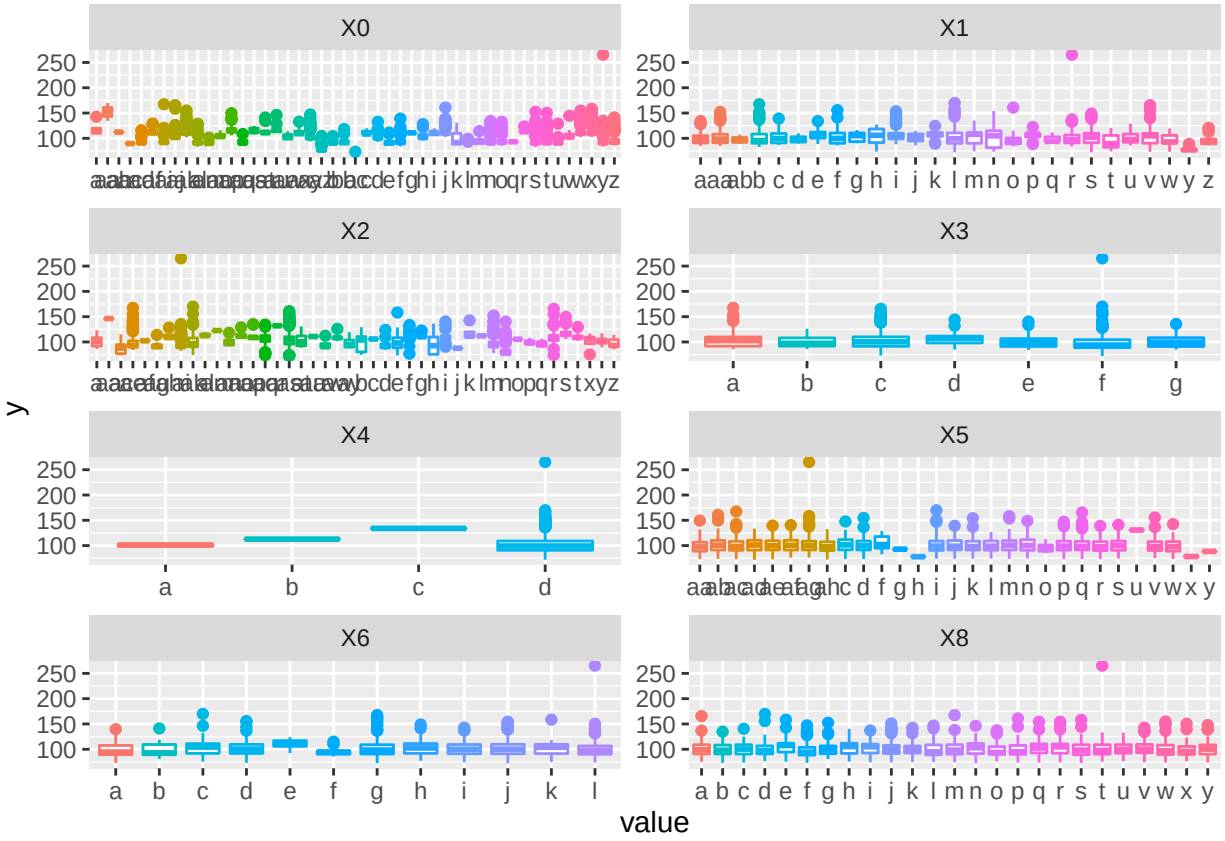
We start examining the outcome variable Y:

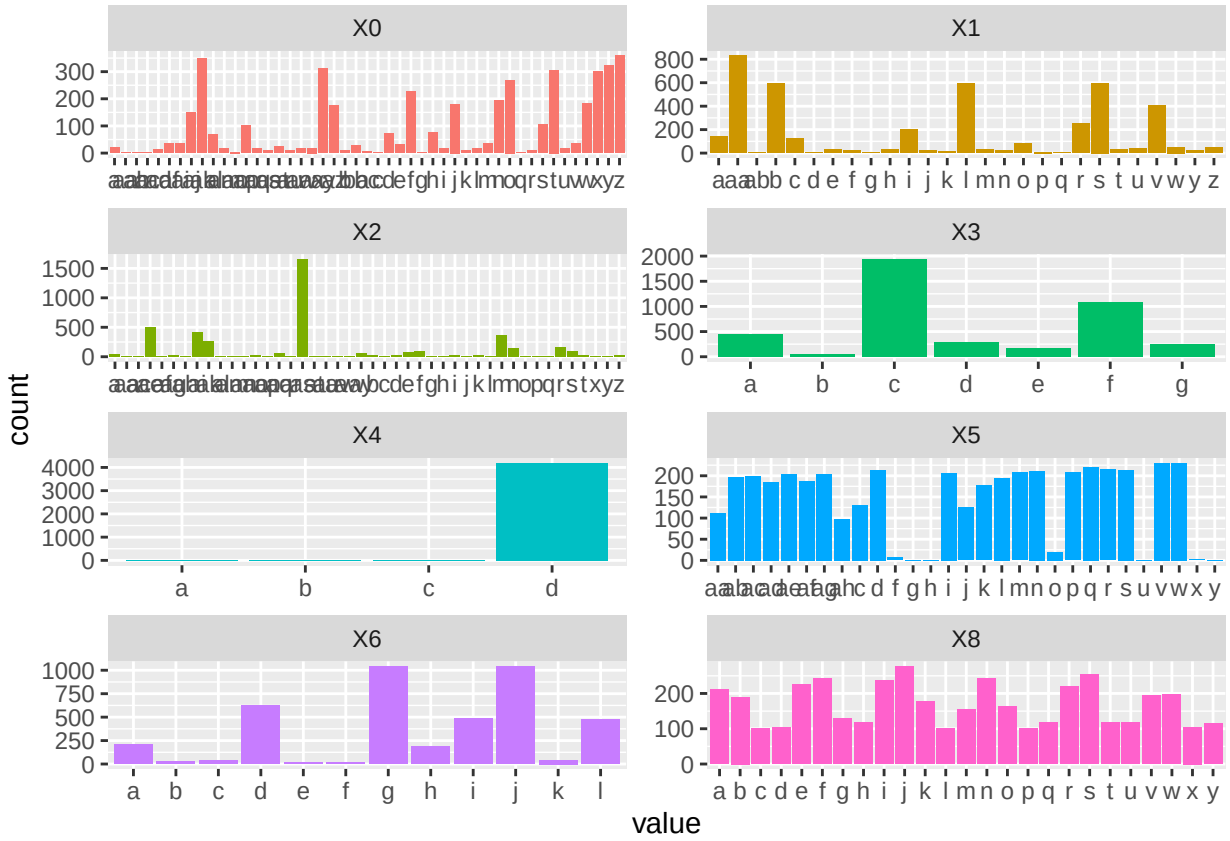


We can see 3 different modes in the values distribution which suggests the presence of clusters in the data. Most observations have Y values between 72 and 175 except for one observation that lays on 265.32.

y statistics	value
Min	72.11
1st Qu.	90.82
Median	99.15
Mean	100.67
3rd Qu.	109.01
Max.	265.32
Sd.	12.68

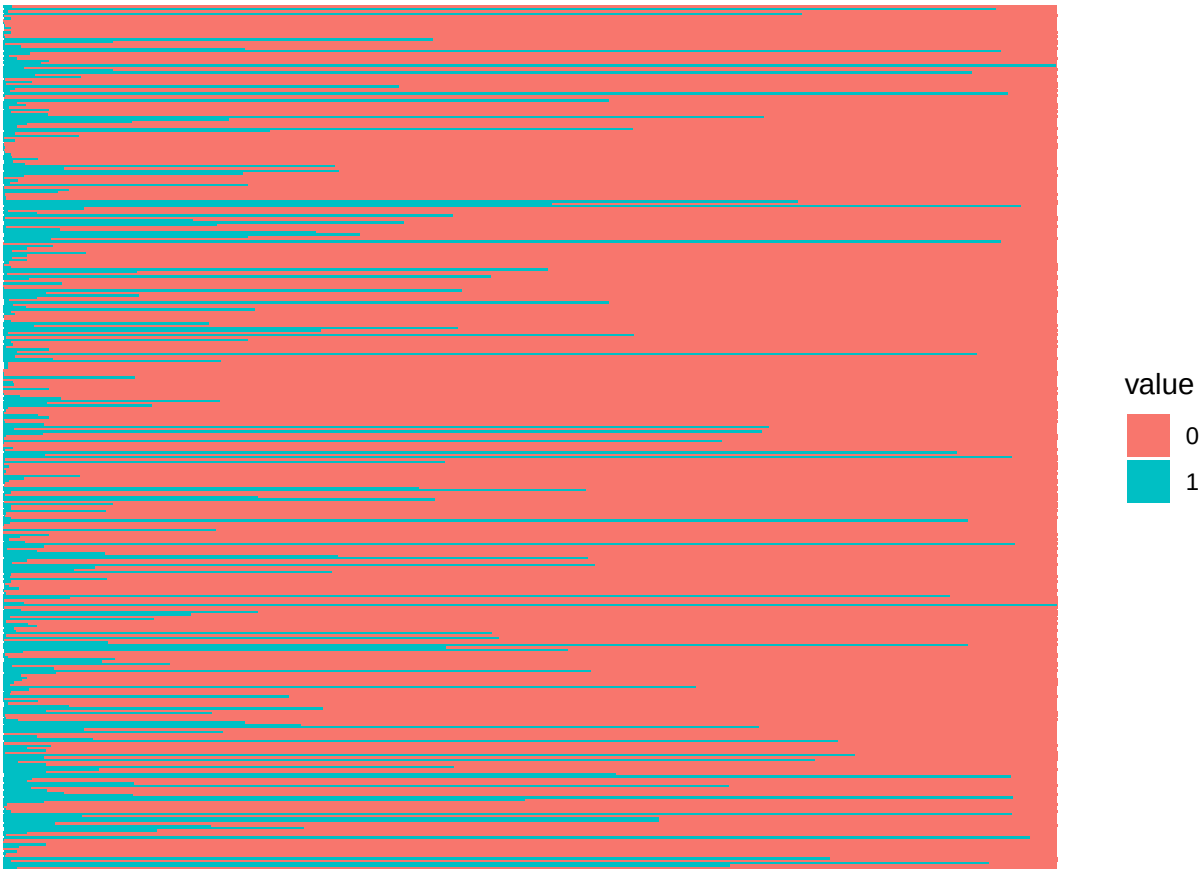
Next we inspect the categorical features. The charts below show their distribution.





Most observations have the value d in the variable X4 so this feature will not be informative.

Finally, the next chart shows the value counts of the binary features.



We can see high variability in many of them while a few are almost always zero or one.

2.3 General model

After the analysis we are now ready to formulate a model for the prediction where the time spent in the test of a car i can be calculated as the total average testing time plus a variability effect for each of the features present in the car.

$$\underline{\overline{y_i = \mu + \sum E_j + \epsilon_i}}$$

Where μ is the average testing rate, E_j is the varability for the feature j and ϵ_i is an independent random error.

We call μ and E_j the baseline predictors, so the time estimate for the car i will be computed as:

$$\underline{\overline{\hat{y}_i = \hat{\mu} + \sum \hat{E}_j}}$$

Where $\hat{\mu}$ and $\hat{E}_j$ are estimates of the baseline predictors.

We can easily compute the estimates of the binary features as:

$$\underline{\overline{\hat{E}_b = \hat{e}_b X_b}}$$

Where $\hat{e}_b$ is the avearage time variability of testing the feature b and X_b is 1 when the feature is present in the car and 0 otherwise.

We can extend this model to the categorical variables too and compute the predictor estimate for a feature c as:

$$\hat{E}_c = \sum_1^n \hat{e}(v) X_{cv}$$

Where n is the number of different values of the feature c, $\hat{e}(v)$ is the average time variability of testing the feature c when the value is v, and X_{cv} is 1 when the value of X_c is v and zero otherwise.

In practice, this is equivalent to convert each categorical feature X_c into n binary features X_{cv} so our final model is left only with binary features.

The table below shows the example of converting the categorical feature $X1$:

ID	y	X1	*	X1_v	X1_t	X1_w	X1_b
0	130.81	v	*	1	0	0	0
6	88.53	t	*	0	1	0	0
7	76.26	w	*	0	0	1	0
9	80.62	t	*	0	1	0	0
13	78.02	v	*	1	0	0	0
18	92.93	b	*	0	0	0	1

We will add this pre-processing step in our dataset and work only with binary features in the following sections.

2.4 Loss function

The loss function measures the perfomance of the prediction in a given test dataset. Here we want to establish how far our prediction $\hat{y}_i$ is from the true time y_i. For this, we will use the *least squares estimates* as the sum of the squares of the differences between the predicted and the true times and our best prediction will be the winner of the minimum residual mean squared error (RMSE) defined as:

$$\overline{RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}}$$

Where $\hat{y}_i$ is our prediction for the testing time of the car i, y_i is the true time in the test dataset for the same car and N is the total number of observations in the test dataset.

2.5 The challenge of dimensionality

After converting the categorical features our *train* dataset has now 563 variables for only 3,786 observations. The number of variables is already a handicap for some good learning algorithms like KNN or Random Forests but most important is the fact that can make our model overtrained.

Overtraining will occur when the model learns the train data including their noise component and produces low errors in the train data but does not perform well in any other dataset.

Our concern is whether we have the number of observations enough for our model to generalise sufficiently well. The algorithm will try to separate data into classes based on the features present in the train dataset, so when the number of them increases we need more observations to be sure that our sample data represents well the reality.

As we cannot increase the number of observations in this case, we will focus on selecting only the features that make our model the most general possible. In order to know if our model is generalising well at each step, we split the *carTesting* dataset in two separated *train* and *test* datasets with the 70% and 30% of the data respectively randomly sampled from *carTesting*.

In the following sections we will explore a few methods of dimension reduction, train different algorithms in the *train* dataset with the reduced dimension spaces, and produce predictions in the *test* dataset to measure their performance. We will call *expected RMSE* the peformance of the predictions in the *train* dataset to separate it from the performance obtained in the *test* dataset.

2.6 Naive Bayes

In this first model, we assume the same time for all cars independently of the features to test with the differences explained by a random variation:

$$\overline{\underline{y_i = \mu + \epsilon_i}}$$

Where μ the true testing time and ϵ_i are independent errors sampled from the same distribution and centered at zero. Our prediction for a car i will then be:

$$\overline{\underline{\hat{y}_i = \hat{\mu}}}$$

Where $\hat{\mu}$ is the estimate of the true testing time that minimises the loss function:

$$\overline{\underline{RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{\mu})^2}}}$$

Where N is the total number of observations in the *train* dataset. By mathematics, we know that the estimation that minimises the RMSE is the least squares estimation of μ which in this case is the average of all times in the *train* dataset, which is $\hat{y}_i = \frac{1}{N}\sum_{i=1}^{N} y_i$ where y_i are the true times in the *train* dataset:

[1] 100.67

If we compute the prediction with $\hat{\mu}$ in the *train* dataset the expected RMSE obtained is:

[1] 12.674

Which is actually the standard deviation in the dataset.

To confirm our assumption, we can see that we get higher *RMSE* when computing the prediction with any other number than $\hat{\mu}$ in the *train* set:

mu_hat_1	rmse
75	28.626
85	20.152
100	12.692
130	31.954
65	37.852

Finally, the *RMSE* value using $\hat{\mu}$ in the *test* dataset is:

[1] 12.687

Which is actually not too far from the *expected RMSE*.

2.7 Regression model

After the exploratory analysis and the conversion into binary of the categorical features, our model looks like:

$$\overline{y_i = \mu + \sum_{j=1}^{F} e_j X_j + \epsilon_i}$$

Where y_i is the total testing time for the car i, μ is the average testing time, F is the total number of features, e_j is the variability of testing the feature j, X_j is 1 when the feature j is present in the car and 0 otherwise and ϵ_i is and independent error.

We call μ and e_j the baseline predictors for which the algorithm has to compute estimates $\hat{\mu}$ and $\hat{e}_j$ that we will use to calculate predictions in the *test* dataset as:

$$\overline{\hat{y}_i = \hat{\mu} + \sum_{j=1}^{F} \hat{e}_j X_j}$$

By mathematics we know that we can use regression techniques to estimate the intecept μ and the coefficients e_j. Here we will apply two algorithms *Linear Regression* and *Regression Trees* with all the features available in the *train* dataset.

More information and the mathematical foundations of the algorithms can be found in the professor Irizarri's book sections Linear Models and Classification and Regression Trees.

Below the outcome using regression with all the features:

method	expectedRMSE	RMSE
mu_hat	12.6762	12.6867
linear regression	9.5017	8.9767
regression trees	9.4547	9.3000

The performance improves in both cases where linear regression offers better *RMSE* values in *train* and *test* meanwhile regression trees seems to be more stable with similar *RMSE* values in each dataset.

2.8 Dimension reduction - Variable importance

There are three different ways to do dimensionality reduction. The first is just by feature selection that means looking through the available features and decide which ones are actually useful, i.e., correlated to the ouput variable.

The second method is feature derivation, which means deriving new features from the old ones by applying transformations in the original dataset. Considering the features space as a n-dimensions matrix where n is the number of the available features, the transformations consist of changing the axes of coordinates by moving or rotating them and achieving the dimensionality reduction by combining the existing features in a smaller set of axes.

The third is just using clustering to group similar datapoints and to see if this allows fewer features to be used.

We start the exercise of reducing the number of features with the first method of feature selection, and for this we are leveraging the regression tree model built in the previous section using recurrent partitioning.

The algorithm creates partitions in the features space recursively starting with one partition R with the whole features space, then split it into two partitions *R1* and *R2* that will then be split resulting in three partitions, then four, then five, etc. In order to create two new partitions, the algorithm finds a feature X_j and a value s and split the observations in the current partition asking if the value of X_j is *0* or *1*.

The criteria to pick X_j and s are the pair that minimises the residual sum of squares in the new partitions: $\sum(y_i - \hat{y}_{R_1})^2 + \sum(y_i - \hat{y}_{R_2})^2$.

To do the feature selection, we can look at the features that have been used at each partition step in the regression tree model and use only those to train new models:

x
X314
X261
X127
X313
X250
X178
X315
X221
X0_af
X0_at
X264
X62

With the reduced set of dimensions we can afford more expensive algorithms like k-Nearest Neighbors. You can follow the link to find more information about *knn* in Professor Irizarri's book.

method	expectedRMSE	RMSE
mu_hat	12.6762	12.6867
linear regression	9.5017	8.9767
regression trees	9.4547	9.3000
variable importance - linear regression	8.6458	8.6992
variable importance - k-nearest neighbors	9.8573	9.5165

We can see a clear improvement using linear regression with the reduced set of features.

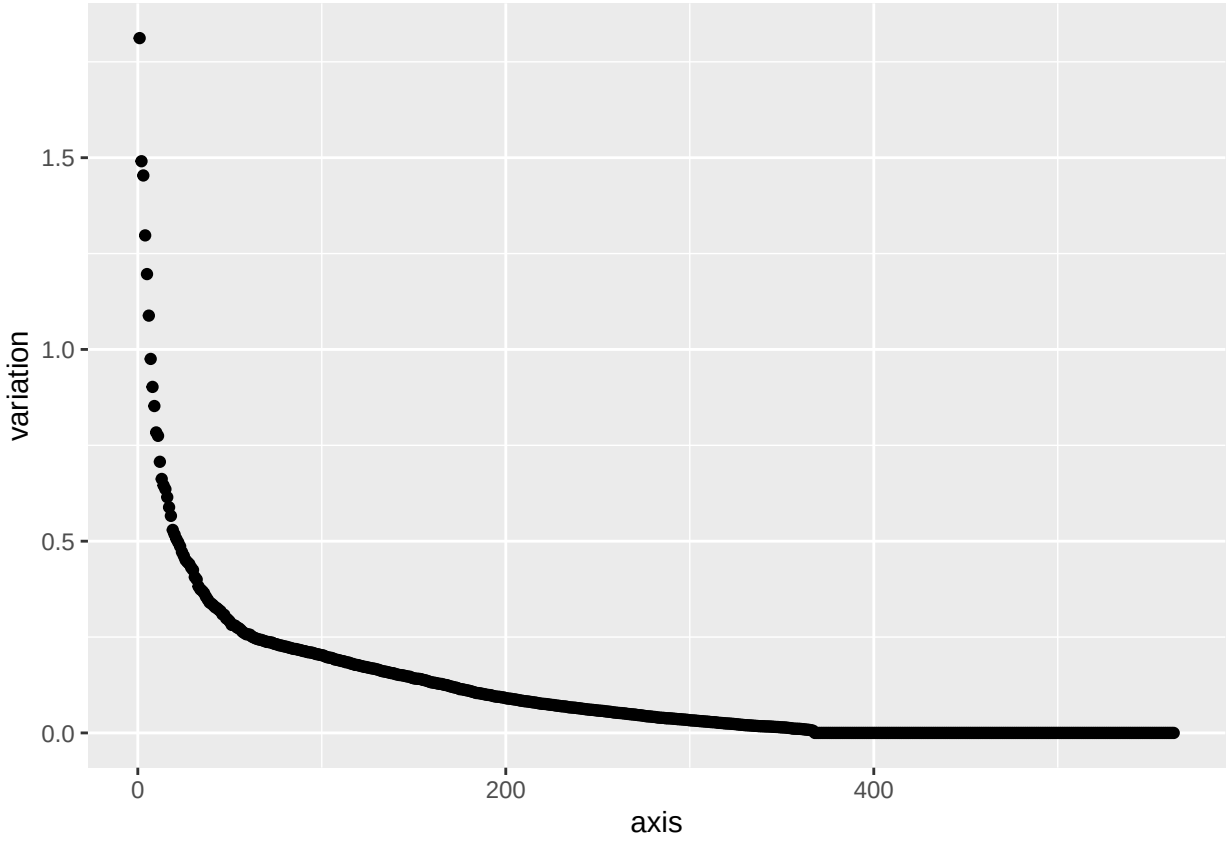
2.9 Dimension reduction - Principal components analysis

PCA is the first of the two dimensionality reduction methods based in feature derivation we will use, that is computing transformation in the data to find a lower dimensional set of axes.

A principal component is a direction of the data with the largest variation. The algorithm starts centering the data removing off the mean, then finds the direction in the data points with the largest variation and puts an axis in that direction. Then looks at the remaining variation and finds an orthogonal (perpendicular but in a n-dimension space) axis to the first one that conveys the largest remaining variation. The process repeats through all the possible axes. The final result is that the higher variation is in the first axes meanwhile the last ones have little variation and can be removed without significantly affecting the varability in the data.

You can find the mathematics and more information in the section Dimension reduction of Professor Irizarri's book.

The below chart shows the variation of the transformed axes in the *train* dataset:



Only 216 dimensions hold the 99% of the variation. We can now train some models using only the 216 first dimensions out of the total 563.

The table below shows the results of training two models, Linear regression and KNN with the reduced features space:

method	expectedRMSE	RMSE
mu_hat	12.6762	12.6867
linear regression	9.5017	8.9767
regression trees	9.4547	9.3000
variable importance - linear regression	8.6458	8.6992
variable importance - k-nearest neighbors	9.8573	9.5165
PCA - linear regression	8.8482	8.4708
PCA - k-nearest neighbors	12.2882	9.7591

We get an improvement using linear regression with the PCA reduced features space meanwhile KNN is still under performing.

2.10 Dimension reduction - Independent Component Analysis

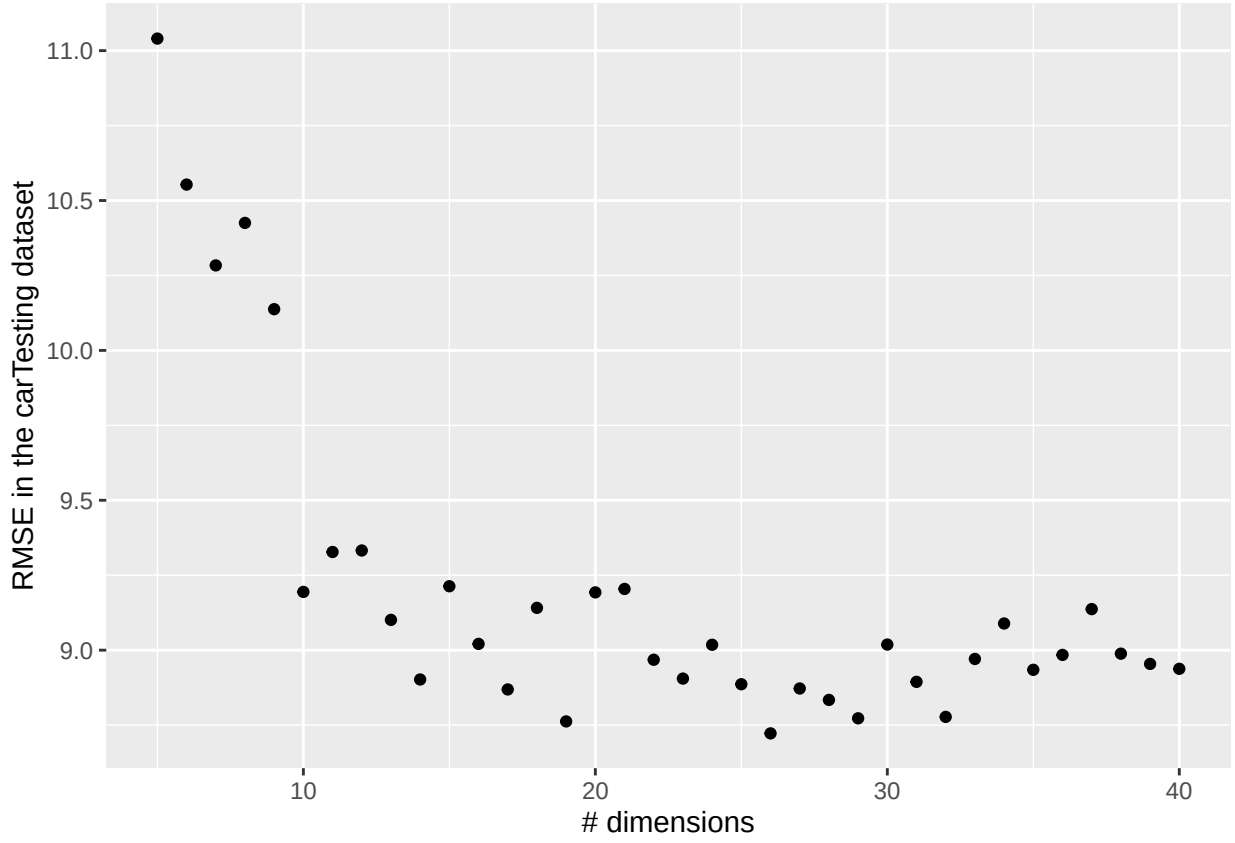
While PCA is about correlation by maximising the variance, our last method of dimension reduction is focused on independence. ICA tries to transform our original set of features $X_1, X_2, ..., X_n$ into a new set of features $I_1, I_2, ..., I_n$ that are mutually independent in the statistical sense while the correlation between each of the new features and the original ones is as high as possible.

Though ICA was initially developed to separate a multivariate signal into additive subcomponents or blind source separation also known as the cocktail party problem, it has another useful application on feature extraction. The idea is to use linear transformations to find suitable but smaller representations from the data, which is what we want to achieve with dimension reduction.

You can find the mathematics and more information about ICA in the book Machine Learning (An Algorithmic Perspective) by Stephen Marsland.

To perform the Independent component analysis we have to provide the number of components in the final predictors space (number of sources in the original notation) to the algorithm. As we do not know which number will be the optimal, we will try to work it out iterating the algorithm and picking up the number of components that gives the best RMSE in the *carTesting* dataset.

The chart below shows the results:



And the best number of components is:

[1] 26

We can now compute the reduced ICA predictors space and train some models:

method	expectedRMSE	RMSE
mu_hat	12.6762	12.6867
linear regression	9.5017	8.9767
regression trees	9.4547	9.3000
variable importance - linear regression	8.6458	8.6992
variable importance - k-nearest neighbors	9.8573	9.5165
PCA - linear regression	8.8482	8.4708
PCA - k-nearest neighbors	12.2882	9.7591
ICA - linear regression	8.9636	9.1790
ICA - k-nearest neighbors	11.7698	9.4887

Linear regression using the ICA features does not give any improvement in the *test* dataset though is the second best performer in the *train* dataset meanwhile KNN is again the worst performer in both datasets.

2.11 Summary of the analysis

From the previous analysis we can see that regression are the best performing methods in all the cases and that we can also achieve improvements by dimension reduction.

method	expectedRMSE	RMSE
linear regression	9.5017	8.9767
regression trees	9.4547	9.3000
variable importance - linear regression	8.6458	8.6992
PCA - linear regression	8.8482	8.4708
ICA - linear regression	8.9636	9.1790

It seems safe to discard the kernel algorithm in the final model and stick to regression using one of the features reduction methods, variable importance, principal component analysis or independent component analysis. PCA is the one who performs better in the *test* set but not in the *train* set, besides variable importance seems more coherent as it 'learns' better the *train* set including some of the noise and therefore gets slightly worse results in the *test* set. In summary also all these results can be just due to luck, the way both datasets are built.

Our decision is to use a blend of the three methods in the final model and compute the predictions as the average of the predictions obtained with the three of them.

III. Results

In the final model we will put together an ensemble of regression algorithms and the three feature reduction methods explained in the previous chapter:

Dimension reduction	Algorithm
Principal Component Analysis	Linear regression
Independent Component Analysis	Linear regression
Independent Component Analysis	Random forests
Variable Importance	Linear regresssion
Variable Importance	Random forests

Here we have included the Random forest algorithm for ICA and variable importance due to the resulting low number of features in both methods. You can find more information about the algorithm in the section Random forests of Professor Irisarri's book.

We will train five models with the *carTesting* set and make a final prediction in the *validation* set using a weighted average of the predictions of the five models. So our prediction for the testing time of a car i is:

$$\hat{y}_i = \sum_{m=1}^{5} w_m \hat{y}_{im} / \sum_{m=1}^{5} w_m$$

Where w_m is the weight of the model m in the final prediction and $\hat{y}_{im}$ is the prediction of the model m for the car i.

We assume that some models are better learners than others and therefore we want to assing different weights to each one in the final prediction. In the following section we explain the steps performed to calculate these weights using cross-validation techniques.

3.1 Cross-validation

We have to estimate the weights to be assigned to each of the algorithms and to do that we want to compare the outcome of the predictions with different weights combinations. We cannot do that just training the algorithms and testing the predictions in the *carTesting* at the risk of overtraining our model. Therefore our first step is to split *carTesting* in two new random training and testing sets, *train_cv* and *test_cv* with a 80% and 20% of the data respectively.

Next, we will train the five models with the *tran_cv* dataset and run predictions in *test_cv* with different weights combinations to find the best perfomer. To save time, we will test the weights of one model at a time and use the weights obtained in the previous steps to test the next model.

The below table shows the best performers after running predictions on *test_cv* using weights from 1 to 50 for each model:

Algorithm	Weight	Expected RMSE
PCA-Linear regression	9	7.8839
VarImp-Linear regresssion	5	7.8517
ICA-Linear regresssion	1	7.8517
VarImp-Random forests	2	7.8510
ICA-Random forests	1	7.8510

3.2 Prediction in the validation dataset

To finalise, we train a model in the *train_cv* set and build a prediction in the *validation* set using the weights obtained in the previos section. Below the final *RMSE* value:

RMSE	*7.911*

IV. Conclusions

The performance obtained 7.911 is below the standard deviation of the *validation* dataset 12.448 by just a 37% which is not a spectacular result. Using or including in the blend better learners like Neural Networks algorithms could lead to further improvement but would increase the risk of overtraining so more attention should be put on this point.

The limitations of the dataset related to the low number of observations have to be also mentioned. Increasing the observations and adding more information like the dates of the tests, the team or machines involved or other details of the proccess, will undoubtely help to train more accurate models.

Murcia, 3^{rd} January 2020