Auto MPG

Machine Learning

Martin Saarman

Introduction

I will use the Car mpg dataset from the Machine Learning Repository[1] to predict the mpg for a car depending on its features. First I will analyze the data to look for outliers, skewed data, correlations etc., after I will preprocess the data to fix everything I found in the previous step. Next I will construct a test harness where I can feed different machine learning algorithms to my data to see which algorithms has best performance, the best performing models will be taken into the next step for tuning and then compared again. I will then take the best performing algorithms and use ensemble tuning to get even better performance.

Data Understanding

First I print the top 20 rows to have a look on what kind of features and values I have to work with. The “car name” feature seems to have a lot of unique values which can be unnecessary noise for the dataset.

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year	origin	car name
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	chevrolet chevelle malibu
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	buick skylark 320
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	plymouth satellite
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	amc rebel sst
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	ford torino
5	15.0	8	429.0	198.0	4341.0	10.0	70	1	ford galaxie 500
6	14.0	8	454.0	220.0	4354.0	9.0	70	1	chevrolet impala
7	14.0	8	440.0	215.0	4312.0	8.5	70	1	plymouth fury iii
8	14.0	8	455.0	225.0	4425.0	10.0	70	1	pontiac catalina
9	15.0	8	390.0	190.0	3850.0	8.5	70	1	amc ambassador dpl
10	15.0	8	383.0	170.0	3563.0	10.0	70	1	dodge challenger se
11	14.0	8	340.0	160.0	3609.0	8.0	70	1	plymouth 'cuda 340
12	15.0	8	400.0	150.0	3761.0	9.5	70	1	chevrolet monte carlo
13	14.0	8	455.0	225.0	3086.0	10.0	70	1	buick estate wagon (sw)
14	24.0	4	113.0	95.00	2372.0	15.0	70	3	toyota corona mark ii
15	22.0	6	198.0	95.00	2833.0	15.5	70	1	plymouth duster
16	18.0	6	199.0	97.00	2774.0	15.5	70	1	amc hornet
17	21.0	6	200.0	85.00	2587.0	16.0	70	1	ford maverick
18	27.0	4	97.0	88.00	2130.0	14.5	70	3	datsun pl510
19	26.0	4	97.0	46.00	1835.0	20.5	70	2	volkswagen 1131 deluxe sedan

There are 305 unique car name values of 398, I will come back to this in the preprocessing.

```
There are 305 unique values in car name.

Dataset shape: (398, 9)
```

Looking at the different types of the dataset I found something interesting, the “horsepower” feature seems to be an object.

```
mpg             float64
cylinders         int64
displacement    float64
horsepower       object
weight          float64
acceleration    float64
model year        int64
origin            int64
car name         object
dtype: object
```

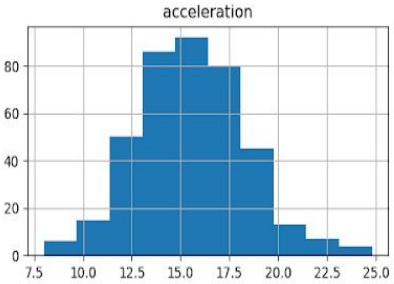
[1] "UCI Machine Learning Repository: Auto MPG Data Set." https://archive.ics.uci.edu/ml/datasets/auto+mpg.

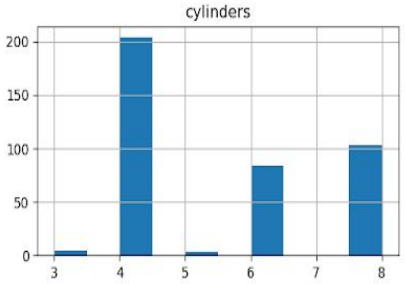
Looking at the 20 first rows, we can see that “horsepower” seems to be containing floating values. I created a loop that would transform the values into floating points and if an error would appear I appended it to a list that I printed to see if I had any non numerical values.

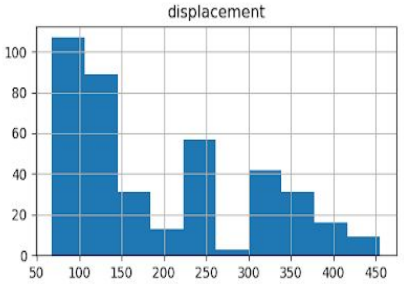
After running the loop it seems like I had 6 string values in myfeature.

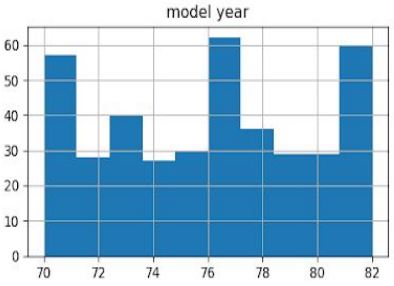
```
There are 6 <class 'str'> values in horsepower: ['?', '?', '?', '?', '?', '?']
```

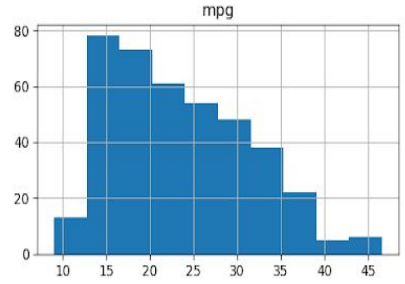
I have some skew in the data that has to be dealt with in the preprocessing stage.

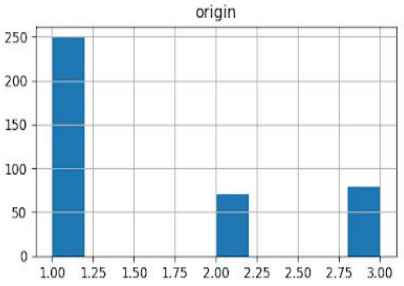


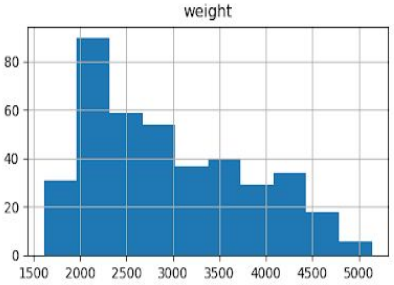













```
Skew in data:
 mpg             0.457066
cylinders        0.526922
displacement     0.719645
weight           0.531063
acceleration     0.278777
model year       0.011535
origin           0.923776
dtype: float64
```

Looking at the correlation between the data features we can see that the correlation of “cylinders” and “displacement” are similarly correlated to the target value “mpg” , they are also highly related to each other in the real world. I will comeback to this in the preprocessing.

```
DataCorrelation:
                   mpg  cylinders  displacement    weight  acceleration  model year    origin
mpg           1.000000  -0.775396     -0.804203 -0.831741      0.420289    0.579267  0.563450
cylinders    -0.775396   1.000000      0.950721  0.896017     -0.505419   -0.348746 -0.562543
displacement -0.804203   0.950721      1.000000  0.932824     -0.543684   -0.370164 -0.609409
weight       -0.831741   0.896017      0.932824  1.000000     -0.417457   -0.306564 -0.581024
acceleration  0.420289  -0.505419     -0.543684 -0.417457      1.000000    0.288137  0.205873
model year    0.579267  -0.348746     -0.370164 -0.306564      0.288137    1.000000  0.180662
origin        0.563450  -0.562543     -0.609409 -0.581024      0.205873    0.180662  1.000000
```

Data preprocessing

Let's take a look at the “car name” feature, we could extract the car companies and see if we get fewer unique values.

```
Unique values in company column: 37
```

Creating this new feature “company” gives us 37 unique values instead of 305. I’m sure we can bring down the number of unique values even more.

```
Frequency of values:
 ford            48
chevrolet        43
plymouth         31
dodge            28
amc              27
toyota           25
datsun           23
buick            17
pontiac          16
volkswagen       15
honda            13
mercury          11
oldsmobile       10
mazda            10
fiat              8
peugeot           8
audi              7
volvo             6
chrysler          6
vw                6
subaru            4
opel              4
saab              4
chevy             3
renault           3
bmw               2
mercedes-benz     2
maxda             2
cadillac          2
nissan            1
chevroelt         1
hi                1
mercedes          1
toyouta           1
triumph           1
vokswagen         1
capri             1
```

Looking at the numbers to the left we can see that there are 12 values that appears only 1-2 times, we can also see that some company names are misspelled and treated as different values.

A good start is to change the name of the misspelled values to the correct ones to decrease the amount of unique values.

After correcting the misspelled company names we decreased the amount of unique values by 7, as shown below.

```
Unique values in company column: 30
```

But we still have some values that only appears 1-2 times, so I am going to generalize these values and give them a special value.

Now we have 25 unique values instead of 305, much better!

```
Unique values in company column: 25
```

Before moving on I want to extract the categorical values from the numeric, but my categorical feature ‘origin’ is in integers, which we’ll have to change before we can extract it from the dataset. After some research I found out that 1 represents “America”, 2 represents “Europe” and 3 represents “Asia”[2].

Now when I have my categorical dataset I am gonna binarize the values so that the data works with more algorithms, but it leaves us with a higher dimension in the data which could cause overfitting.

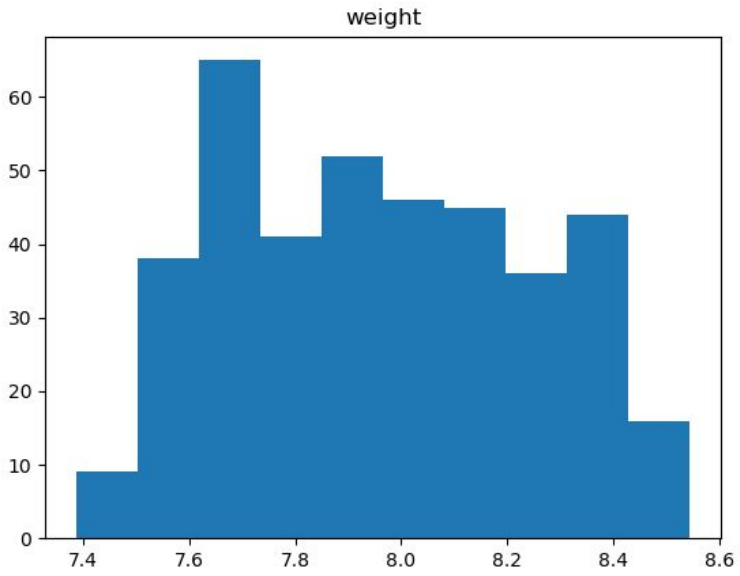
```
Shape of categorical data: (392, 28)
```

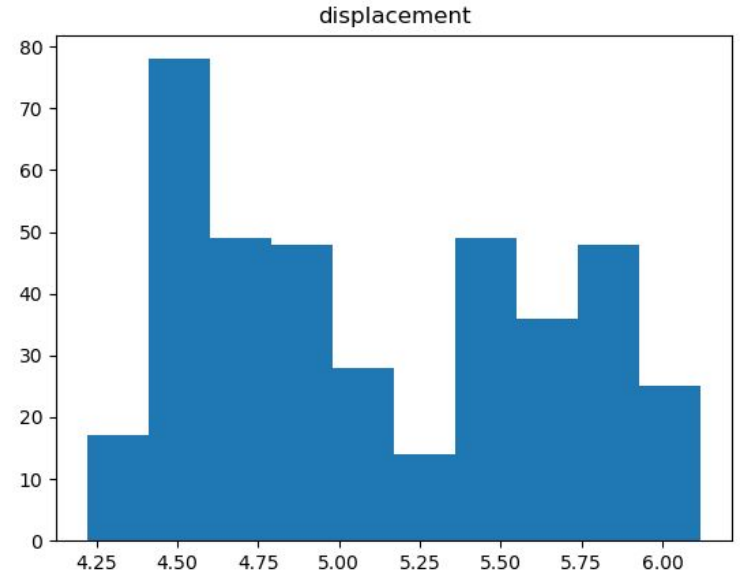
[2] "autompg: The Auto MPG dataset in dprep: Data Pre ... - Rdrr.io." https://rdrr.io/cran/dprep/man/autompg.html

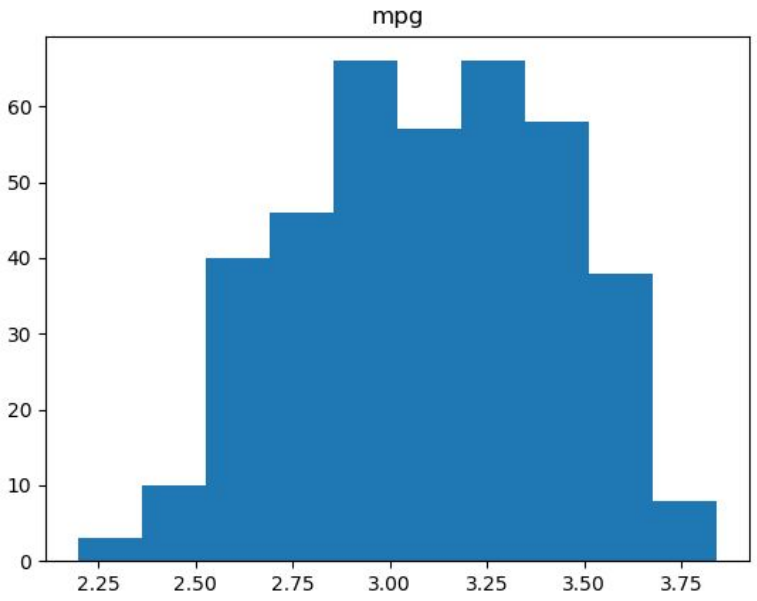
Moving onto our numerical dataset I will start off by fixing the string values in the “horsepower”. I locate the string values, and because there are only 6 of them, I drop the entire rows where were the values are located.

Now let’s take care of the features “model year” and “cylinders” by rescaling them. The reason I rescale is because they are representing something rather than being a value. And because I am gonna standardize and rescale the other data features I will remove “cylinders” and ”model year” from the dataset.

Now let’s fix the skew in the continuous data, I will do this by taking the log value of every feature with a skew above 0.4. Looking at the graphs below the skew seems to be much better than before.







mpg	-0.131382
displacement	0.211368
horsepower	0.370148
weight	0.148536
acceleration	0.291587

As I mentioned in the previous section I noticed a strong correlation between “cylinders” and “displacement”, my idea is to combine them to get the feature “cylinder_displacement” and remove the other two features. Because I am not entirely sure on this, I will do this in a copy of the original dataset, so when spot checking my algorithms I will use one dataset with “cylinder” and “displacement” features and one dataset with ‘cylinder_displacement” feature instead.

Now all the major skews are fixed, the categorical data binarized and the bounded continuous values are rescaled, it is now time to rescale the continuous values.
I start off by removing the “mpg” because it is our target value we do not have to rescale it.

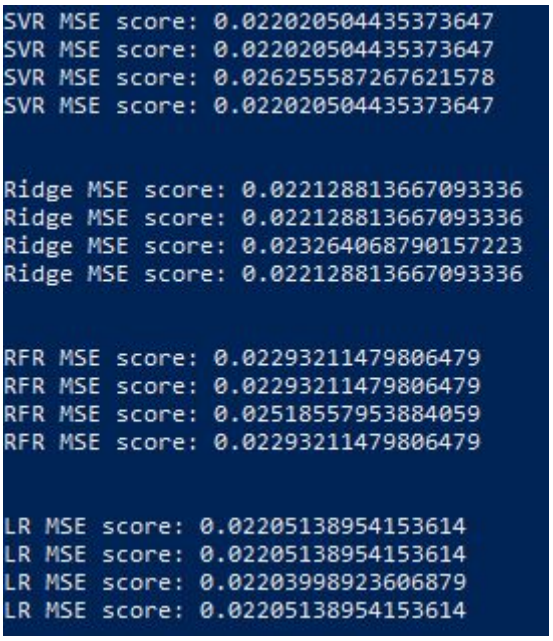
I will use rescaling and standardisation to rescale the data, my plan is to have 6 datasets, the dataset with the “cylinder_displacement” feature original, standardized and rescaled, and then the same for the original dataset.

Spot checking
Now when we have our data cleaned and optimized it is time to start looking at different algorithms. Because I do not know which algorithm will be a best fit for my data I will construct a test harness to test different algorithms and rank them by negative MSE (Mean Squared Error).

```
LinearRegression with Data original result: -0.025850113252024347
LinearRegression with Data original [cd] result: -0.02595771637399934
LinearRegression with Data Standardized result: -4.420607161551232e+18
LinearRegression with Data Standardized [cd] result: -1.1614418220205505e+20
LinearRegression with Data Rescaled result: -0.026206476987754485
LinearRegression with Data Rescaled [cd] result: -0.025957716373999386


DecisionTreeRegressor with Data original result: -0.03888415490406729
DecisionTreeRegressor with Data original [cd] result: -0.040617890359398
DecisionTreeRegressor with Data Standardized result: -0.0404444194776736
DecisionTreeRegressor with Data Standardized [cd] result: -0.03896834518299086
DecisionTreeRegressor with Data Rescaled result: -0.040848686031953185
DecisionTreeRegressor with Data Rescaled [cd] result: -0.03825672772512997


K-Nearest Neighbor with Data original result: -0.0429740931527496
K-Nearest Neighbor with Data original [cd] result: -0.03508103678321614
K-Nearest Neighbor with Data Standardized result: -0.0342225835704174
K-Nearest Neighbor with Data Standardized [cd] result: -0.034165135899770736
K-Nearest Neighbor with Data Rescaled result: -0.03269586180797037
K-Nearest Neighbor with Data Rescaled [cd] result: -0.03240860834554023


Support Vector Regressor with Data original result: -0.025288933628841475
Support Vector Regressor with Data original [cd] result: -0.027443585570567884
Support Vector Regressor with Data Standardized result: -0.027356818638160453
Support Vector Regressor with Data Standardized [cd] result: -0.02776865752461215
Support Vector Regressor with Data Rescaled result: -0.02979967446398507
Support Vector Regressor with Data Rescaled [cd] result: -0.028885646858026408


Ridge with Data original result: -0.02533669517558802
Ridge with Data original [cd] result: -0.025455407261251457
Ridge with Data Standardized result: -0.025865474130337924
Ridge with Data Standardized [cd] result: -0.02585953571574326
Ridge with Data Rescaled result: -0.026370161275561344
Ridge with Data Rescaled [cd] result: -0.026227475081545402


RandomForestRegressor with Data original result: -0.02654966179603239
RandomForestRegressor with Data original [cd] result: -0.027459123379384782
RandomForestRegressor with Data Standardized result: -0.0262926015560146
RandomForestRegressor with Data Standardized [cd] result: -0.02674006371686813
RandomForestRegressor with Data Rescaled result: -0.027677011145556187
RandomForestRegressor with Data Rescaled [cd] result: -0.028131881942840924


AdaBoostRegressor with Data original result: -0.028070045221153594
AdaBoostRegressor with Data original [cd] result: -0.027847599188704053
AdaBoostRegressor with Data Standardized result: -0.02848433074737349
AdaBoostRegressor with Data Standardized [cd] result: -0.02772817174417578
AdaBoostRegressor with Data Rescaled result: -0.026545742215795236
AdaBoostRegressor with Data Rescaled [cd] result: -0.02801329808879676
```

After running the algorithms through my test harness I decided to continue on with 4 models that performed the best. The models are Support Vector Regressor(negative MSE: -0.2529), Ridge(negative MSE: -0.2534), RandomForestRegressor(negative MSE: -0.2571) and Linear Regression(negative MSE: -0.2585). All the best performing models performed best on the original data set

Tuning the 4 chosen models we got some new scores, this time calculated in MSE



As we can see above SVR was the best performing(MSE: 0.02202), followed by LR(MSE: 0.02205), Ridge(MSE: 0.02213) and RFR(MSE: 0.02293).

I will take SVR alone into hyperparameter tuning and leave the others for a possible ensembles in the end for better performance.

Model tuning

I start the tuning by testing the parameters one by one, giving it very large values and the very low values and everytime I change a value I run it through a model evaluation function, then I work my way into the middle. When I found a set of good values I run different values close to the once I found through Scikits “GridSearchCV”[3] to find the best values down to the decimal.

[3] "sklearn.model_selection.GridSearchCV — scikit-learn 0.20.0 ...." http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html. Öppnades 16 nov.. 2018.

But before I start the tuning I run the algorithm through the mode evaluation to set threshold.

```
Test Result: 0.027997793288915065

Train Result: 0.024054948239860884
```

In the picture above I can see that the training set performs better then the test result, which is a sign of overfitting. Hopefully we can fix this with tuning, otherwise I have to look over the data again.

Going over the process previously mentioned I ended up with these parameters**:**

C=1.4, kernel='linear', max_iter=14, epsilon=0.132

And running the model evaluation we can see that the test set not only performs better then the training set but it also performs better then before.

```
Test Result: 0.022255486329956664

Train Result: 0.023654909468622753
```

Ensembles Tuning

For ensembles tuning I tried to combine the four best performing algorithms (Support Vector Regression, Random Forest Regressor, Linear Regression and Ridge), I got the best performance when I combined SVR, LR and Ridge, so I created a new model called “MR” with these three models combined.

Because of the changes made I tested my model on all of the datasets again and found that the rescaled original Dataset had the best performance. The scores below is a comparison against Support Vector Regression and MR on rescaled data, where we can see that MR performance much better.

```
SVR train result: 6.705367681487639
MR train result: 2.2459020763372184

SVR Result: 0.22033325894230907
MR Result: 0.02414392713583498
```

Summary
In this report you have followed my progression through my first Machine Learning project. We have gone through:

Data Understanding, where we looked at different information and graphs of the data to see what kind of data we we are working with and what we had to fix in the preprocessing stage.

Data Preprocessing, here we took the knowledge we gained in the previous stage and used different methods i.e logarithm to prevent skews, binarizing categorical data to avoid hierarchy between the variables, etc. This stage is where you can improve the performance the most.

Spot checking, this is really important because you can not know which algorithm will fit best on your data set, taking a few different algorithms (linear, non-linear, ensemble) and run test harness on them with your data, you can see which algorithms performs the best on your data.

Model tuning, in this stage we are trying to get the most of the algorithms that performed best in spot checking. This stage is all about getting the most out of the algorithm/algorithms you have chosen.

Ensembles tuning, if you had several well performed algorithms on your data, why not join them together to increase performance even more. I used ensembles tuning by taking the best performing algorithms and then taking the mean of the models predictions as the final prediction.