Autopreisprognosse_machine_learning_dataquest

May 31, 2020

1 Machine Learning Projekt von dataquest.io

Das folgende Projekt stammt von dataquest.io, einer Lernplattform für Data Science mit Pyhton und R. Nach jedem Lernkapitel wird eine Projektaufgabe ohne Lösung gestellt, in der die "Studenten" das gelernte anwenden können. Diese Projekt stammt aus dem 6. Kapitel des Data Science Tracks.

Das Datenset wird von dataquest gestellt und kann von den "Studenten" heruntergeladen werden.

1.1 Datenanalyse und Bereinigung

In einem ersten Schritt werde ich die Daten kurz untersuchen, schauen welche Werte bereinigt werden müssen und diese dann bereinigen

Zur schnelleren Analyse greife ich auf Bibliotheken wie pandas und numpy zurück.

[6]:
```
import pandas as pd

cols = ["symboling", "normalized_losses", "make", "fuel_type", "aspiration",
       "num_doors", "body_style", "drive_wheels", "engine_location",
       "wheel_base", "length", "width", "height", "curb_weight", "engine_type",
       "num_cylinders", "engine_size", "fuel_system", "bore", "stroke",
       "compression_ratio", "horsepower", "peak_rpm", "city_mpg", "highway_mpg",
       "price"]
cars = pd.read_csv("imports-85.data", names=cols)
cars.dtypes
```

[6]:
```
symboling              int64
normalized_losses     object
make                  object
fuel_type             object
aspiration            object
num_doors             object
body_style            object
drive_wheels          object
engine_location       object
wheel_base           float64
length               float64
width                float64
```

```
height               float64
curb_weight            int64
engine_type           object
num_cylinders         object
engine_size            int64
fuel_system           object
bore                  object
stroke                object
compression_ratio    float64
horsepower            object
peak_rpm              object
city_mpg               int64
highway_mpg            int64
price                 object
dtype: object
```

Wir haben 25 Spalte mit Daten, um einen Autpreis zu prognostizieren.

[7]:
```
# Anschauen der Daten
cars.head()
```

[7]:

	symboling	normalized_losses	make	fuel_type	aspiration	num_doors	\
0	3	?	alfa-romero	gas	std	two	
1	3	?	alfa-romero	gas	std	two	
2	1	?	alfa-romero	gas	std	two	
3	2	164	audi	gas	std	four	
4	2	164	audi	gas	std	four	

	body_style	drive_wheels	engine_location	wheel_base	…	engine_size	\
0	convertible	rwd	front	88.6	…	130	
1	convertible	rwd	front	88.6	…	130	
2	hatchback	rwd	front	94.5	…	152	
3	sedan	fwd	front	99.8	…	109	
4	sedan	4wd	front	99.4	…	136	

	fuel_system	bore	stroke	compression_ratio	horsepower	peak_rpm	city_mpg	\
0	mpfi	3.47	2.68	9.0	111	5000	21	
1	mpfi	3.47	2.68	9.0	111	5000	21	
2	mpfi	2.68	3.47	9.0	154	5000	19	
3	mpfi	3.19	3.40	10.0	102	5500	24	
4	mpfi	3.19	3.40	8.0	115	5500	18	

	highway_mpg	price
0	27	13495
1	27	16500
2	26	16500
3	30	13950

```
4          22  17450

[5 rows x 26 columns]
```

Wir wollen in einem ersten Schritt die Fragezeichen zu NaN verwandeln, um mit Ihnen arbeiten zu können. Weiter werden wir uns auf die SPalten mit numerischen Werten konzentrieren und versuchen so viele Werte wie möglich in Floating Werte umzuwandeln.

[9]:
```
import numpy as np

cars = cars.replace('?', np.nan)

# Now lets make things numeric
num_vars = ['normalized_losses', "bore", "stroke", "horsepower", "peak_rpm",
            "price"]

for i in num_vars:
    cars[i] = cars[i].astype('float64')

cars.head()
```

[9]:

	symboling	normalized_losses	make	fuel_type	aspiration	num_doors	\
0	3	NaN	alfa-romero	gas	std	two	
1	3	NaN	alfa-romero	gas	std	two	
2	1	NaN	alfa-romero	gas	std	two	
3	2	164.0	audi	gas	std	four	
4	2	164.0	audi	gas	std	four	

	body_style	drive_wheels	engine_location	wheel_base	…	engine_size	\
0	convertible	rwd	front	88.6	…	130	
1	convertible	rwd	front	88.6	…	130	
2	hatchback	rwd	front	94.5	…	152	
3	sedan	fwd	front	99.8	…	109	
4	sedan	4wd	front	99.4	…	136	

	fuel_system	bore	stroke	compression_ratio	horsepower	peak_rpm	city_mpg	\
0	mpfi	3.47	2.68	9.0	111.0	5000.0	21	
1	mpfi	3.47	2.68	9.0	111.0	5000.0	21	
2	mpfi	2.68	3.47	9.0	154.0	5000.0	19	
3	mpfi	3.19	3.40	10.0	102.0	5500.0	24	
4	mpfi	3.19	3.40	8.0	115.0	5500.0	18	

	highway_mpg	price
0	27	13495.0
1	27	16500.0
2	26	16500.0
3	30	13950.0

```
4           22  17450.0

[5 rows x 26 columns]
```

[10]:
```
# Anzahl der NaN zählen
print("normalized losses: ", cars['normalized_losses'].isnull().sum())
```

```
normalized losses:  41
```

In der Spalte normalized losses fehlen 41 Werte, was bei 205 Zeilen eine große Menge ausmacht. Nun wollen wir die fehlenden Werte in den anderen Spalten checken.

[11]:
```
cars.isnull().sum()
```

[11]:
```
symboling             0
normalized_losses    41
make                  0
fuel_type             0
aspiration            0
num_doors             2
body_style            0
drive_wheels          0
engine_location       0
wheel_base            0
length                0
width                 0
height                0
curb_weight           0
engine_type           0
num_cylinders         0
engine_size           0
fuel_system           0
bore                  4
stroke                4
compression_ratio     0
horsepower            2
peak_rpm              2
city_mpg              0
highway_mpg           0
price                 4
dtype: int64
```

Unser Datenset sieht eigentlich ziemlich sauber aus, bis auf "normalized-losses". Nun werde ich nach einem logischen Vorgehen Zeilen mit fehlenden Werten löschen.

[12]:
```
# Wir starten beim Preis
cars = cars.dropna(subset = ['price'])
```

[13]:
```
cars.isnull().sum()
```

[13]:
```
symboling            0
normalized_losses    37
make                 0
fuel_type            0
aspiration           0
num_doors            2
body_style           0
drive_wheels         0
engine_location      0
wheel_base           0
length               0
width                0
height               0
curb_weight          0
engine_type          0
num_cylinders        0
engine_size          0
fuel_system          0
bore                 4
stroke               4
compression_ratio    0
horsepower           2
peak_rpm             2
city_mpg             0
highway_mpg          0
price                0
dtype: int64
```

Das hat bereits geholfen. Nun werde ich die anderen Zeilen löschen.

[16]:
```
cars = cars.dropna(subset = ['bore', 'stroke', 'horsepower', 'peak_rpm',␣
↪'num_doors'])
```

[17]:
```
cars.isnull().sum()
```

[17]:
```
symboling            0
normalized_losses    34
make                 0
fuel_type            0
aspiration           0
num_doors            0
body_style           0
drive_wheels         0
engine_location      0
wheel_base           0
```

```
length               0
width                0
height               0
curb_weight          0
engine_type          0
num_cylinders        0
engine_size          0
fuel_system          0
bore                 0
stroke               0
compression_ratio    0
horsepower           0
peak_rpm             0
city_mpg             0
highway_mpg          0
price                0
dtype: int64
```

Bis auf normalized_losses sind alle fehlenden Werte entfernt.

In einem letzten Schritt vor der Modellierung will ich die numerischen Daten noch in ein neues Datenset übertragen bevor ich anfange zu modellieren.

[18]:
```
cols = ['wheel_base', 'length', 'width', 'height',
        'curb_weight', 'engine_size', 'bore', 'stroke', 'horsepower',
        'peak_rpm', 'city_mpg', 'highway_mpg', 'price']
cars = cars[cols]

normalized_cars = (cars - cars.mean()) / (cars.std())
```

1.2 Modellierung

In einem nächsten Schritt werde ich eine KNN Funktion aufstellen und die einzelnen Spalten mit KNN und RMSE darauf überprüfen, ob sie als sinnvolle "Predictor" für meine spätere Preisprognose dienen können.

Hierfür greife ich auf die scikit-learn Bibliotheken zurück, die ich während dem dataquest Kurs kennengelernt habe.

[19]:
```
# Funktion, die eindimensionale Modelle trainiert und testet
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error

def knn_train_test(train_col, target_col, df):
    knn = KNeighborsRegressor()
    np.random.seed(1)

    # willkürliche Anordnung der Daten
```

```
    shuffled_index = np.random.permutation(df.index)
    rand_df = df.reindex(shuffled_index)

    # Halbieren der Anzahl Zellen
    last_train_row = int(len(rand_df) / 2)

    # Die erste Hälfte dient als Trainingset
    # Die zweite Hälfte dient als Testset
    train_df = rand_df.iloc[0:last_train_row]
    test_df = rand_df.iloc[last_train_row:]

    # KNN Klassifizierung
    knn.fit(train_df[[train_col]], train_df[target_col])
    predicted_labels = knn.predict(test_df[[train_col]])

    # Berechnen der Fehlerwerte RMSE.
    mse = mean_squared_error(test_df[target_col], predicted_labels)
    rmse = np.sqrt(mse)
    return rmse
```

[20]:
```
# Testen der Spalten als Prediktor mit der KNN Klassifizierung
print('city mpg: ', knn_train_test('city_mpg', 'price', normalized_cars))
print('width: ', knn_train_test('width', 'price', normalized_cars))
print('highway mpg: ', knn_train_test('highway_mpg', 'price', normalized_cars))
print('engine size: ', knn_train_test('engine_size', 'price', normalized_cars))
print('horsepower: ', knn_train_test('horsepower', 'price', normalized_cars))
```

```
city mpg:  0.5977207173562914
width:  0.5771034906881971
highway mpg:  0.5608495944049304
engine size:  0.3971745521927362
horsepower:  0.6098874396874532
```

Die Auswertung zeigt, dass überraschenderweise horsepower ein schlechter Indikikator für den Autopreis ist und Hubraum ein sehr guter. Weiter scheint die Autolänge und Miles per Gallon in der Stadt ebenfalls eher schlecht Indikatoren für den Autopreis zu sein. Jedoch sind diese Fehlerwerte alle isoliert betrachtet ohne Kombination mit anderen Spalten.

Um meine Ergebnisse nochmals zu verifizieren, lasse ich das ganze nochmals mit unterschiedlichen Werte k trainieren und testen mit einem array [1,3,5,7,9].

[21]:
```
def knn_train_test_new(train_col, target_col, df):
    np.random.seed(1)

    # willkürliche Anordnung der Daten
    shuffled_index = np.random.permutation(df.index)
    rand_df = df.reindex(shuffled_index)
```

```
    # Halbieren der Anzahl Zellen
    last_train_row = int(len(rand_df) / 2)

    # Die erste Hälfte dient als Trainingset
    # Die zweite Hälfte dient als Testset
    train_df = rand_df.iloc[0:last_train_row]
    test_df = rand_df.iloc[last_train_row:]

    k_values = [1,3,5,7,9]
    k_rmses = {}

    for k in k_values:
        # KNN Klassifizierung
        knn = KNeighborsRegressor(n_neighbors=k)
        knn.fit(train_df[[train_col]], train_df[target_col])
        predicted_labels = knn.predict(test_df[[train_col]])

        # Berechnen der Fehlerwerte RMSE.
        mse = mean_squared_error(test_df[target_col], predicted_labels)
        rmse = np.sqrt(mse)

        k_rmses[k] = rmse
    return k_rmses

k_rmse_results = {}

# Auswertung wie oben
variables = ['wheel_base', 'length', 'width', 'height',
        'curb_weight', 'engine_size', 'bore', 'stroke', 'horsepower',
        'peak_rpm', 'city_mpg', 'highway_mpg']

for var in variables:
    rmse_val = knn_train_test_new(var, 'price', normalized_cars)
    k_rmse_results[var] = rmse_val

k_rmse_results
```

[21]:
```
{'wheel_base': {1: 0.7828031239770414,
  3: 0.7221718127658258,
  5: 0.7268602466331098,
  7: 0.7318249142495196,
  9: 0.7426709021404414},
 'length': {1: 0.7736226245682798,
  3: 0.7125607916591421,
  5: 0.7199542942094256,
  7: 0.7239918146856693,
  9: 0.7290176209706889},
```

```
'width': {1: 0.6176931867173693,
 3: 0.5690380013712371,
 5: 0.5771034906881971,
 7: 0.5504416097314524,
 9: 0.5848656944796756},
'height': {1: 1.3400340333646827,
 3: 1.0532562209407335,
 5: 1.0459430601353807,
 7: 1.0011920801837217,
 9: 1.010731315203313},
'curb_weight': {1: 0.6129716888197899,
 3: 0.5121654437092118,
 5: 0.5286404643259163,
 7: 0.5380609933418921,
 9: 0.590251744141821},
'engine_size': {1: 0.4313523026109327,
 3: 0.38540202322422074,
 5: 0.3971745521927362,
 7: 0.4407233560752905,
 9: 0.4816944688371549},
'bore': {1: 0.9011379460150852,
 3: 0.8965186318344839,
 5: 0.8919639155079472,
 7: 0.9026843866754162,
 9: 0.8924841963753289},
'stroke': {1: 0.9934459676306456,
 3: 0.935288095039312,
 5: 0.990090799676581,
 7: 1.0535201165048573,
 9: 1.0239432945294218},
'horsepower': {1: 0.5551792460213537,
 3: 0.5485689216336084,
 5: 0.6098874396874532,
 7: 0.5983131237792242,
 9: 0.6378913552365378},
'peak_rpm': {1: 0.9395997391524226,
 3: 0.918587698914419,
 5: 1.0176300243629757,
 7: 0.9766038214074376,
 9: 0.9832953915745098},
'city_mpg': {1: 0.6983005498860759,
 3: 0.5546692840207508,
 5: 0.5977207173562914,
 7: 0.5364764399465124,
 9: 0.5806425028118403},
'highway_mpg': {1: 0.5867136650883984,
 3: 0.5202565771379636,
```

```
5: 0.5608495944049304,
7: 0.5819525133241601,
9: 0.6140223087476621}}
```

Die Auswertung zeigt, dass k = 9 für manche Spalten besser passt als für andere. Die Auswertung zeigt zudem lediglich die isolierte Betrachtung einer Spalte. Diese Auswertung gibt uns noch kein zufriedenstellendes Ergebnis. In einem nächsten Schritt werde ich die 5 besten Prediktor nutzen und sie verlinkt nach dem KNN Algorithmus trainieren und testen.

Ich konzentriere mich im Anschluss auf “engine-size”, “highway_mpg”, “curb_weight”, “horsepower”, “city_mpg”.

[35]:
```python
def knn_train_test_mult(train_cols, target_col, df):
    np.random.seed(1)

    # willkürliche Anordnung der Daten
    shuffled_index = np.random.permutation(df.index)
    rand_df = df.reindex(shuffled_index)

    # Halbieren der Anzahl Zellen
    last_train_row = int(len(rand_df) / 2)

    # Die erste Hälfte dient als Trainingset
    # Die zweite Hälfte dient als Testset
    train_df = rand_df.iloc[0:last_train_row]
    test_df = rand_df.iloc[last_train_row:]

    k_values = [5]
    k_rmses = {}

    for k in k_values:
        # KNN Klassifizierung
        knn = KNeighborsRegressor(n_neighbors=k)
        knn.fit(train_df[train_cols], train_df[target_col])

        predicted_labels = knn.predict(test_df[train_cols])

        # Berechnen der Fehlerwerte RMSE.
        mse = mean_squared_error(test_df[target_col], predicted_labels)
        rmse = np.sqrt(mse)

        k_rmses[k] = rmse
    return k_rmses

train_cols_2 = ['engine_size', 'highway_mpg']
train_cols_3 = ['engine_size', 'highway_mpg', 'curb_weight']
train_cols_4 = ['engine_size', 'highway_mpg', 'curb_weight',
                'horsepower']
```

```
train_cols_5 = ['engine_size', 'highway_mpg', 'curb_weight',
                'horsepower', 'city_mpg']

k_rmse_results = {}

rmse_val = knn_train_test_mult(train_cols_2, 'price', normalized_cars)
k_rmse_results["2 Indikatoren"] = rmse_val
rmse_val = knn_train_test_mult(train_cols_3, 'price', normalized_cars)
k_rmse_results["3 Indikatoren"] = rmse_val
rmse_val = knn_train_test_mult(train_cols_4, 'price', normalized_cars)
k_rmse_results["4 Indikatoren"] = rmse_val
rmse_val = knn_train_test_mult(train_cols_5, 'price', normalized_cars)
k_rmse_results["5 Indikatoren"] = rmse_val

k_rmse_results
```

[35]:
```
{'2 Indikatoren': {5: 0.4191209082213026},
 '3 Indikatoren': {5: 0.4636929631531235},
 '4 Indikatoren': {5: 0.47673590549429157},
 '5 Indikatoren': {5: 0.48473636731175224}}
```

Die Auswertung zeigt, dass der Autopreis am besten mit engine-size und highway_mpg prognostiziert werden kann. Dies scheint wesentlich besser als bis zu 5 Indikatoren zu berücksichtigen. Es ist zu erkennen, dass beim Einbezug von mehr Indikatoren der Fehlerwert größter wird.

Aufgrund der Auswertung werde ich mit den zwei Indikatoren engine-size und highway-mpg weiterfahren. Ich bin noch nicht ganz glücklich mit dem k-Wert und werde deshalb eine weitere Analyse fahren, die k-Werte von 1-12 trainieren und testen.

[39]:
```
def knn_train_test_mult(train_cols, target_col, df):
    np.random.seed(1)

    # Randomize order of rows in data frame.
    shuffled_index = np.random.permutation(df.index)
    rand_df = df.reindex(shuffled_index)

    # Divide number of rows in half and round.
    last_train_row = int(len(rand_df) / 2)

    # Select the first half and set as training set.
    # Select the second half and set as test set.
    train_df = rand_df.iloc[0:last_train_row]
    test_df = rand_df.iloc[last_train_row:]

    k_values = list(range(1,12))
    k_rmses = {}

    for k in k_values:
```

```
        # Fit model using k nearest neighbors.
        knn = KNeighborsRegressor(n_neighbors=k)
        knn.fit(train_df[train_cols], train_df[target_col])

        # Make predictions using model.
        predicted_labels = knn.predict(test_df[train_cols])

        # Calculate and return RMSE.
        mse = mean_squared_error(test_df[target_col], predicted_labels)
        rmse = np.sqrt(mse)

        k_rmses[k] = rmse
    return k_rmses

k_rmse_results_2 = {}

rmse_val = knn_train_test_mult(train_cols_2, 'price', normalized_cars)
k_rmse_results_2["2 Indikatoren"] = rmse_val
rmse_val = knn_train_test_mult(train_cols_3, 'price', normalized_cars)
k_rmse_results_2["3 Indikatoren"] = rmse_val
rmse_val = knn_train_test_mult(train_cols_4, 'price', normalized_cars)
k_rmse_results_2["4 Indikatoren"] = rmse_val
rmse_val = knn_train_test_mult(train_cols_5, 'price', normalized_cars)
k_rmse_results_2["5 Indikatoren"] = rmse_val

k_rmse_results_2
```

[39]:
```
{'2 Indikatoren': {1: 0.4023119204496319,
  2: 0.34775590666938055,
  3: 0.3507184342389931,
  4: 0.4067514295115171,
  5: 0.4191209082213026,
  6: 0.4188435188707149,
  7: 0.4413089734275166,
  8: 0.4591034248021089,
  9: 0.48264287270600864,
  10: 0.5093119092683336,
  11: 0.5349839561048475},
 '3 Indikatoren': {1: 0.35218286564751927,
  2: 0.32979442084968225,
  3: 0.3754954388813044,
  4: 0.4218338552059866,
  5: 0.4636929631531235,
  6: 0.4914370282915947,
  7: 0.5215248079192926,
  8: 0.5379645832617136,
  9: 0.5497995664635731,
```

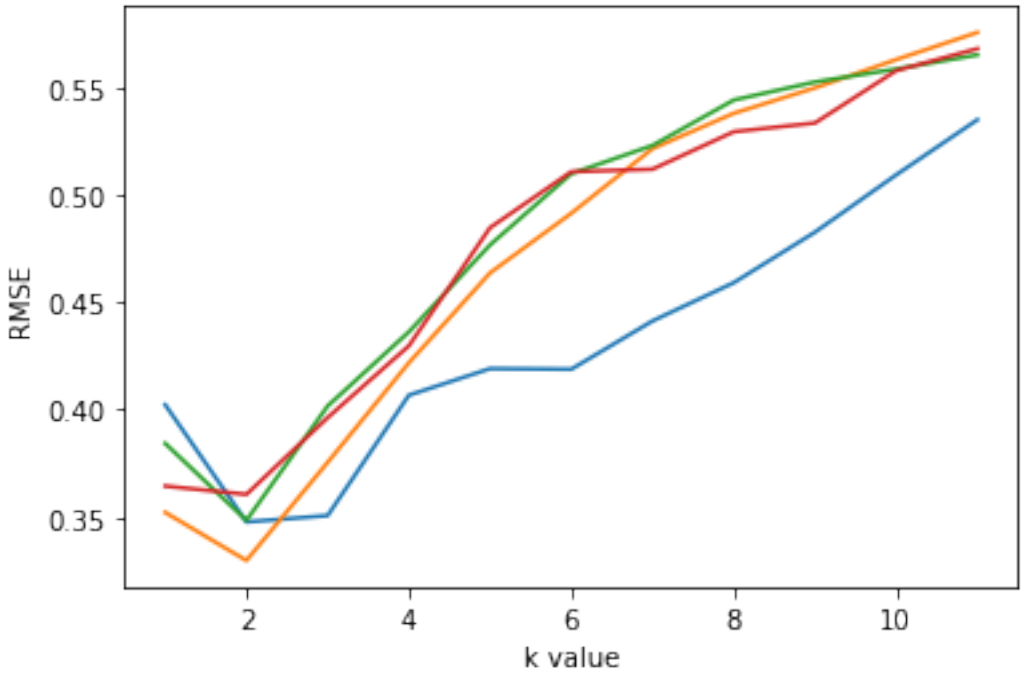
```
 10: 0.5629664343247851,
 11: 0.5756343394184102},
'4 Indikatoren': {1: 0.3843138526538685,
 2: 0.3484779243821034,
 3: 0.40172110525771537,
 4: 0.43623039986312573,
 5: 0.47673590549429157,
 6: 0.5098674809262911,
 7: 0.522998321668861,
 8: 0.5441444346389539,
 9: 0.5525564226736684,
 10: 0.5585701021067181,
 11: 0.5651536243365779},
'5 Indikatoren': {1: 0.36445475858367077,
 2: 0.3605878322838429,
 3: 0.39630768202414457,
 4: 0.42972052753199136,
 5: 0.48473636731175224,
 6: 0.5107596280883786,
 7: 0.5119494193806042,
 8: 0.529391227283452,
 9: 0.5334546850227097,
 10: 0.5578241006856575,
 11: 0.5680440476599966}}
```

[40]:
```
import matplotlib.pyplot as plt

for k,v in k_rmse_results_2.items():
    x = list(v.keys())
    y = list(v.values())

    plt.plot(x,y)
    plt.xlabel('k value')
    plt.ylabel('RMSE')

plt.show()
```



Die Auswertung zeigt, dass egal für welche Anzahl Prediktor eine Anzahl k zwischen 2 und 3 am besten ist.

Zusammenfassung: Wir haben herausgefunden, dass für die Prognose vom Autopreis die Indikatoren engine-size und highway mpg völlig ausreichen und k entweder 2 oder 3 sein sollte.