1 Predicting house prices (part 2)

We try to improve prediction of house prices based on Erdogan Seref’s German housing dataset from www.immobilienscout24.de published at www.kaggle.com under a Attribution-NonCommercial-ShareAlike 4.0 International License.

We load preprocessed data and adjust data types.

```python
[1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import sklearn.linear_model as linear_model
import sklearn.metrics as metrics
import sklearn.model_selection as model_selection
import sklearn.preprocessing as preprocessing
import sklearn.pipeline as pipeline

data_path = 'house_prices2/'

[2]: data = pd.read_csv(data_path + 'german_housing_preprocessed.csv', index_col=0)

data['Type'] = data['Type'].astype('category')
data['Condition'] = data['Condition'].astype('category')
data['Garage_type'] = data['Garage_type'].astype('category')

data['Condition'].cat.reorder_categories(['first occupation', 'first occupation after refurbishment', 'maintained', 'renovated', 'modernized', 'refurbished', 'refurbishment'],)
```
Results obtained from linear regression showed that input variables do not suffice to explain the targets. Thus, we should add more input variables. When preprocessing the data we dropped several columns. Keeping them could increase prediction quality slightly, but there were several good reasons to drop those columns. The main reason were lots of missing values in those columns.

A far better idea is to collect additional data. What features of a house influence the selling price? Of course its location! Up to now we did not use location information at all, but we have location information available. There are columns State, City, Place. But city names do not help. We need something like proximity to big cities or nice landscape. Adding a layer of abstraction we might ask for the demand for houses and the wealth of potential buyers. So we should head out for statistical information about local real estate markets and about economic power of different regions in Germany.

Everything we need is publicly available at www.regionalstatistik.de provided by Statistische Ämter des Bundes und der Länder under the license Datenlizenz Deutschland – Namensnennung – Version 2.0. Clicking here and there we find two interesting tables: * annual income per inhabitant * prices for construction ground

From those tables we may compile a table with 4 columns (region id, region name, income, ground prices).
The difficult part is matching region names in German housing data set with region names in the region table. Here is some code doing the job:

```python
# remove rows with missing location information
data = data.dropna(subset=('State', 'City'))

# reindex to remove gaps in the index
#data.index = data.index.reindex(range(0, len(data)))[0]
data.index = pd.RangeIndex(0, len(data))

# read CSV file containing region names
regions = pd.read_csv(data_path + 'regions.csv')
regions.head(5)

data['city_short'] = data['City'].str.replace(' (Kreis)', '', regex=False)
data['region_idx'] = 0

for (idx, city_short) in enumerate(data['city_short']):
    find_results = regions['region'].str.find(str(city_short))
    if not (find_results > -1).any():
        #print(city_short)
        if data.loc[idx, 'State'] == 'Hamburg':
            find_results = regions['region'].str.find('Hamburg')
            data.loc[idx, 'region_idx'] = regions.index[find_results > -1][-1]
        elif data.loc[idx, 'State'] == 'Bremen':
            find_results = regions['region'].str.find('Bremen')
            data.loc[idx, 'region_idx'] = regions.index[find_results > -1][-1]
        elif data.loc[idx, 'State'] == 'Berlin':
            district = city_short.split('(','')[-1][0:-1]
            if district == 'Weißensee':
                district = 'Pankow'
            elif district == 'Prenzlauer Berg':
                district = 'Pankow'
            elif district == 'Hohenschönhausen':
                district = 'Lichtenberg'
            elif district == 'Wedding':
                district = 'Berlin-Mitte'
            find_results = regions['region'].str.find(district)
            #print('***', city_short, ':', district, '-->', regions['Region'][find_results > -1])
            data.loc[idx, 'region_idx'] = regions.index[find_results > -1][-1]
```

---

3
elif city_short == 'Neuss (Rhein-Kreis)':
    find_results = regions['region'].str.find('Neuss')
data.loc[idx, 'region_idx'] = regions.index[find_results > -1][[-1]
elif city_short == 'Sankt Wendel':
    find_results = regions['region'].str.find('Wendel')
data.loc[idx, 'region_idx'] = regions.index[find_results > -1][[-1]
elif city_short == 'Stadtverband Saarbrücken':
    find_results = regions['region'].str.find('Saarbrücken,Regionalverband')
data.loc[idx, 'region_idx'] = regions.index[find_results > -1][[-1]
elif city_short.split()[1] in ('in', 'im', 'an', 'am'):
    if city_short.split()[0] == 'Neustadt':
        find_results = regions['region'].str.find(city_short.split()[-1])
    else:
        find_results = regions['region'].str.find(city_short.split()[0])
        data.loc[idx, 'region_idx'] = regions.index[find_results > -1][[-1]
    data.loc[idx, 'region_idx'] = regions.index[find_results > -1][[-1]
else:
    print(‘NOT FOUND:’, city_short)
else:
    # take last match (smallest region)
data.loc[idx, 'region_idx'] = regions.index[find_results > -1][[-1]

[7]: data['Region_id'] = regions.loc[data['region_idx'], 'id'].values
data['Income'] = regions.loc[data['region_idx'], 'income'].values
data['Land_prices'] = regions.loc[data['region_idx'], 'prices'].values

[8]: data = data.drop(columns=['city_short', 'region_idx'])

[9]: data.head(5)

<table>
<thead>
<tr>
<th>Price</th>
<th>Type</th>
<th>Living_space</th>
<th>Lot</th>
<th>Rooms</th>
<th>Bathrooms</th>
<th>Floors</th>
<th>Year_built</th>
<th>Year_renovated</th>
<th>Condition</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.118355</td>
<td>Multiple dwelling</td>
<td>4.663439</td>
<td>5.433722</td>
<td>5.5</td>
<td>1.0</td>
<td>2.0</td>
<td>2.772589</td>
<td>2.772589</td>
<td>modernized</td>
<td>Baden-Württemberg</td>
</tr>
<tr>
<td>13.526494</td>
<td>Farmhouse</td>
<td>5.093075</td>
<td>4.406719</td>
<td>5.0</td>
<td>2.0</td>
<td>4.0</td>
<td>2.079442</td>
<td>2.079442</td>
<td>dilapidated</td>
<td>Baden-Württemberg</td>
</tr>
<tr>
<td>12.464583</td>
<td>Farmhouse</td>
<td>4.941642</td>
<td>6.701960</td>
<td>4.0</td>
<td>2.0</td>
<td>2.0</td>
<td>4.795791</td>
<td>3.044522</td>
<td>fixer-upper</td>
<td>Baden-Württemberg</td>
</tr>
<tr>
<td>12.804909</td>
<td>Duplex</td>
<td>5.424950</td>
<td>6.880384</td>
<td>10.0</td>
<td>4.0</td>
<td>2.0</td>
<td>4.406719</td>
<td>1.945910</td>
<td>modernized</td>
<td>Baden-Württemberg</td>
</tr>
<tr>
<td>14.375126</td>
<td>Mid-terrace house</td>
<td>5.347108</td>
<td>7.286192</td>
<td>6.0</td>
<td>2.0</td>
<td>3.0</td>
<td>4.406719</td>
<td>1.945910</td>
<td>modernized</td>
<td>Baden-Württemberg</td>
</tr>
</tbody>
</table>
### 1.2 Linear regression

Now we do linear regression as before, but with two additional columns.

```python
[11]:
data['Condition_codes'] = data['Condition'].cat.codes
data = pd.get_dummies(data, columns=['Type', 'Garage_type'], drop_first=True)

[12]:
y = data['Price'].to_numpy()
X = data.drop(columns=['Price', 'Condition', 'State', 'City', 'Place',
                      'Region_id']).to_numpy()

print(X.shape, y.shape)

(4763, 23) (4763,)

[13]:
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y,
                            test_size=0.2)

print(y_train.size, y_test.size)

3810 953

[14]:
steps = [('poly', preprocessing.PolynomialFeatures()), ('ridge', linear_model.Ridge())]

pipe = pipeline.Pipeline(steps)

param_grid = {'poly__degree': [1, 2, 3], 'ridge__alpha': [0] + [2 ** k for k in range(5, 15)]}

gs = model_selection.GridSearchCV(pipe, param_grid=param_grid,
```
```python
gs = gs = GridSearchCV(pipe, param_grid, scoring='neg_mean_squared_error', n_jobs=-1, cv=5)
gs.fit(X_train, y_train)
best_params = gs.best_params_
```

1.3 Evaluation

Now the interesting part. Do we see an increase in prediction quality?

```python
[15]:
print(best_params)

pipe.set_params(**best_params)
pipe.fit(X_train, y_train)
```

```plaintext
{'poly__degree': 2, 'ridge__alpha': 512}
```

```python
y_test_pred = pipe.predict(X_test)
```

```python
[16]:
rmse = np.sqrt(metrics.mean_squared_error(y_test, y_test_pred))
sigma = np.std(y_test)
print('RMSE:', rmse)
print('standard deviation:', sigma)
print('ratio:', rmse / sigma)
```

RMSE: 0.4248066771021125
standard deviation: 0.801578549021144
ratio: 0.5299626313863683

```python
[17]:
fig, ax = plt.subplots()
ax.plot(y_test, y_test_pred, 'or', markersize=3)
ax.plot([9, 17], [9, 17], '-b')
ax.set_xlabel('true targets')
ax.set_ylabel('predictions')
ax.set_aspect('equal')
plt.show()
```
Looks much better!

1.4 Feature importance

With a trained model we may look at feature importances to see which features have high influence on the selling price.

```python
[18]: import sklearn.inspection as inspection

[19]: result = inspection.permutation_importance(pipe, X, y, n_jobs=-1)

[20]: cols = data.drop(columns=['Price', 'Condition', 'State', 'City', 'Place', 'Region_id']).columns
    imp = pd.Series(result.importances_mean, index=cols)
    imp = imp.sort_values(ascending=False)
    imp

[20]: Land_prices    0.372617
    Living_space    0.362034
    Year_built      0.174309
    Lot             0.068924
    Income          0.057952
    Rooms           0.031420
    Type_Villa      0.015153
    Bathrooms       0.014141
```
<table>
<thead>
<tr>
<th>Feature</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type_Duplex</td>
<td>0.013529</td>
</tr>
<tr>
<td>Year_renovated</td>
<td>0.010670</td>
</tr>
<tr>
<td>Garage_type_Outside parking lot</td>
<td>0.009923</td>
</tr>
<tr>
<td>Type_Corner house</td>
<td>0.008855</td>
</tr>
<tr>
<td>Floors</td>
<td>0.007275</td>
</tr>
<tr>
<td>Condition_codes</td>
<td>0.007134</td>
</tr>
<tr>
<td>Type_Mid-terrace house</td>
<td>0.005969</td>
</tr>
<tr>
<td>Garages</td>
<td>0.004720</td>
</tr>
<tr>
<td>Garage_type_Garage</td>
<td>0.003324</td>
</tr>
<tr>
<td>Type_Multiple dwelling</td>
<td>0.002968</td>
</tr>
<tr>
<td>Type_Single dwelling</td>
<td>0.002767</td>
</tr>
<tr>
<td>Type_Farmhouse</td>
<td>0.001868</td>
</tr>
<tr>
<td>Type_Special property</td>
<td>0.001690</td>
</tr>
<tr>
<td>Garage_type_Underground parking lot</td>
<td>0.001336</td>
</tr>
<tr>
<td>Type_Residential property</td>
<td>0.000284</td>
</tr>
<tr>
<td>dtype: float64</td>
<td></td>
</tr>
</tbody>
</table>