Predicting Apartment rental offers in Germany with Polynomial and Random Forest Regression

1. Introduction

The chosen application for this project is quite a common business case as well as an important aspect for consumers. Germany is the major hub for the tech industry in Europe and is home to many Multinational corporations. Each years, thousands of students and workers come to Germany aiming to get a good spot in some of the most prestigious universities and companies in the world. On planning to go to Germany, finding a suitable house to live is an essential task that must be focused on. This is such a daunting and time-consuming task as there are so many aspects that make up the house rent that unless you have plenty of experience already, you would end up living in a house that does not fit you best. For this reason, I made this project aiming to provide future prediction on house rent and help people to better understand about the real estate market situation in Germany so to make their planning process become easier. This is a supervised project.

Section 1 of this report is an introduction to the topic, while section 2 ("Problem formulation") discusses how to formulate the above-mentioned problem to make it suitable for applying different machine learning problems. Section 3 discusses the details of the dataset, data processing, appliance of polynomial regression, and Random Forest regression models. Sections 4 and 5 ("Results" and "Conclusions") are about the results of the project as well as the conclusions that can be drawn from these results, while Section 6 ("References") lists the references. Section 7 ("Code Appendix")

2. Problem formulation

Due to the high volume and active real estate market in Germany, there is a lot of publicly available dataset regarding housing information in general in this country and they are updated constantly. However, those sets of data are too complicated to analyze and without proper knowledge can lead to unreliable prediction on price. Moreover, as we suffered from the Covid pandemic, the real estate markets in many countries are unstable and soon when everything comes back to normal, the prediction on these sets of data is longer accurate [1][2]. As stated above, this project aims to provide information for future students and workers, and because of this, I chose the dataset on the pre-pandemic period. The dataset used in the project is from Kaggle, created by CorrieBar [3], whose data was scraped from Immoscout24, the biggest real estate platform in Germany. Immoscout24 has listings for both rental properties and homes for sale.

Features	Data type	Explanations
ServiceCharge	Numerical (float)	auxiliary costs such as electricity or internet in €
livingSpace	Numerical (float)	Size of the rental object (in square meter)

baseRent	Numerical (float)	Rent excluding heating, service charges, etc. (Kaltmiete, in €)
pictureCount	Numerical (int)	how many pictures were uploaded to the listing
priceTrend	Numerical(float)	price trend as calculated by Immoscout
Date	Categorical	Time of data scraping (month, year)
Label	Data type	Explanations
totalRent	Numerical (float)	Total rent including heating, service charges, etc. (Warmmiete, in €

3. Methods

3. Methods

3.1. Dataset

Originally, the dataset contains over 200000 data points with 49 different variables which cover every aspect that can affect the house rent. The real estate market is complicated containing many properties and this is the main reason leading to the high number of variables. However, too many features may lead to computational complexity, which was a probable case for this. The dataset also contains both numeric and non-numeric data which can make it even harder to compute the result. Hence, the need for feature selection was obvious. Most data used for this project is numeric ones and some non-numeric if they can easily be turned in to categorical data. After filtering, this project’s dataset contains just 3000 datapoints with 6 different variables. Based on what I have read [10][11], these features are fundamental parts of a house rent which make up a large part of the final rent. That is the reason why just 6 of them are sufficient for this project.

When it comes to comparatively small datasets K-fold cross-validation is especially useful as the risk of the unlucky single split is high. (Jung, p.156) [4]. In this project, there are 3000 data points, and this number is high enough to use the single split method. The project TA suggest me to choose the training set to be bigger than the validation and test sets, and all sets should be large enough to represent reality. For that reason, the data set was split into three as the training set, validation set, and test set with ratios 0.6, 0.2, and 0.2, respectively.

3.2. Linear regression model

We start off with a simple linear regression model. This model was chosen because it is generally a good starting point for describing continuous quantities such as prices and there appears to be a linear relationship between the label and the feature based on visualizations, as can be seen above. This hypothesis space takes the form:

$$Y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \ldots + \beta_p X_{ip} + \epsilon_i$$

for each observation $i = 1, \ldots, n$.

I used mean squared error as my loss function because that is what is typically used with linear regression.

3.3 Polynomial regression model

However, after analyzing the dataset and calculate error of the above model, I saw that there is not a simple linear relationship between the feature values and the label value. Therefore, it was reasonable to use polynomial regression to make an accurate prediction about the label value. The mathematical ground of polynomial regression with degree k is as follows:

$$\hat{y} = \beta_0 + \beta_2 * x_1^1 + \beta_2 * x_1^2 + \ldots + \beta_n * x_n^k$$ [5]

For this method, mean squared error was used, as it is customary to use it with polynomial regression problems, which is a combination of polynomial features with linear mapping. (Jung, p. 81) The main characteristic of mean squared error is its rate of punishment increases exponentially which might cause problem in case where the dataset contains outliers. However, that is not the case for the dataset used in this project. Therefore, mean square error was a reasonable choice. The formula of mean square error is as follows with ŷ representing prediction and y representing actual value:

$$\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$ [5]

To implement this problem, scikit-learn’s LinearRegression [6], PolynomialFeatures [7], and mean_squared_error [8] methods were used in addition to basic Python commands.

3.3. Random Forest regression model

Considering the non-linear relationship between features and the label in this project, it was needed to find a non-linear model. The Random Forest was selected due to its effectiveness in analyzing non-linear relationships, suitability to multiple feature values, and requiring less data cleaning. It is also a popular method used in many other similar projects. Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operate by constructing many Random Forests at training time. For classification tasks, the output of the random forest is the class selected by most trees. For regression tasks, the mean or average prediction of the individual trees is returned. Random decision forests correct for Random Forests' habit of overfitting to their training set (Wikipedia). Like the polynomial regression, the mean squared error was used for the Random Forest regression. The reasons for this choice were the same as the previous one. It is approriate to use mean squared error with Random Forest regressors as it is the default loss function and since the dataset does not contain so many outliers, the mean squared error could give a reasonable value. To implement this problem, scikit-learn’s RandomForestRegressor [9] and mean_squared_error [8] methods were used in addition to basic Python commands.

4. Results

4.1. Polynomial regression model results

One of the most critical parts of using polynomial regression is deciding on the degree. Since it directly affects the predicted relationship between features and label value, correct degree choice is important. To avoid both underfitting and overfitting, a common model validation method was used. By using mean squared error to calculate training and validation errors, polynomial regressions with the degree of range from 1 to 7 were compared:

Degree	1	2	3	4	5	6	7
Training Error	0.0011293	0.000524	0.000375	0.00035	0.000175	0.00080	0.00086
Validation Error	0.00084	0.00068	0.003654	0.302793	113.274	185932	6468236

Table 1: Training and Validation Errors for Polynomial Regression from 1 to 7

Considering these training and validation errors, the best degree choice for polynomial regression is degree 2 since it yields the smallest validation error of 0.00068 with an acceptable training error. According to the values above, it is reasonable to claim that degree 4 is the degree just before the model starts to overfit.

4.2. Random Forest regression model results

One of the most important design choices related to deciding on trees is the depth of the tree. An increase in the depth means an increase in the model complexity, and it comes with the risk of overfitting. To avoid that, a similar model validation technique used for the polynomial regression was used for the Random Forest regressor. By using mean squared error to calculate training and validation errors, Random Forest regressions with the degree of range from 10 to 16 were compared.

Degree	10	11	12	13	14	15	16
Training Error	0.0001578	0.000145	0.000125	0.000116	0.000124	0.000129	0.000112
Validation Error	0.0010541	0.0010265	0.001027	0.001037	0.001079	0.001045	0.0010558

Table 2: Training and Validation Errors for Random Forest Regression from 10 to 16

Considering these training and validation errors, the best degree choice for polynomial regression is degree 11 since it yields the smallest validation error of 0.0010265 with an acceptable training error.

4.3. Comparing different models

Comparing the validation errors of polynomial regression with degree 2 (0. 00068) and validation error of Random Forest regression with a maximum depth of 11 (0. 0010265) shows that on the given validation set, polynomial regression is performing better (the training of the two models is 0.0005246 and 0.000145 respectively). Therefore, polynomial regression with degree 2 was chosen as the final model. However, to understand the accuracy of the final model, it is needed to evaluate its performance with the test set. The test set consists of elements that are completely new to the model and gives a valuable metric to measure the model's success. In this project, the test set is 20% of the whole dataset and contains 3000 data points. The test set accuracy was calculated by using mean squared error, the same as before and it is 0.00099298 for the final chosen model, polynomial regression with degree 2.

5. Conclusion

In a sense, it was expected to have better predictions by polynomial regression, mostly because this specific project may fall into cases where random forests will struggle [12]. However, its performance exceeded expectations and yielded remarkably close errors with the polynomial regression. For example, the difference between the validation errors of the best Random Forest regression model with the best polynomial regression model is only 0.00052121, a small number. The final model chosen, polynomial regression with degree 2, yielded a test error of 0.000992. Although this might not prove that the model is super accurate, its performance is within the acceptable tolerance limits. Choosing features based on domain knowledge was reasonable in general but doing so might lead to overlooking the mathematical relationship between the label and some feature values. For this reason, it might be worth trying regularization with polynomial regression and increasing the number of feature values. However, it should be considered that increasing the number of features also means an increase in computational complexity. Therefore, a more detailed analysis is needed to find the optimal point.

6. References

[1] Germany home prices to fall in 2023 but rents to keep rising By Indradip Ghosh. Access date October 10, 2023, from
https://www.reuters.com/markets/europe/germany-home-prices-fall-2023-rents-keep-rising-2023-08-31/

[2] Lasting Effects of COVID-19 on the National Housing Market April 24, 2023, MJEDomestic Economics. Access date October 10, 2023, from
https://sites.lsa.umich.edu/mje/2023/04/24/lasting-effects-of-covid-19-on-the-national-housing-market/

[3] Apartment rental offers in Germany. Rental offers scraped from Germany biggest real estate online platform by CORRIEBAR. Retrieved October 10, 2023, from
https://www.kaggle.com/datasets/corrieaar/apartment-rental-offers-in-germany

[4] Jung, A. "Machine Learning: The Basics," Springer, Singapore, 2022. Access date October 10, 2023.

[5] Abhigyan. (2020, August 2). Understanding polynomial regression. Medium. Retrieved October 10, 2023, from https://medium.com/analytics-vidhya/understanding-polynomial-regression-5ac25b970e18
[6] Sklearn.linear_model.linearregression. scikit-learn. Retrieved October 10, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html
[7] Sklearn.preprocessing.PolynomialFeatures. scikit-learn. Retrieved October 10, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html
[8] Sklearn.metrics.mean_squared_error. scikit-learn. Retrieved October 10, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html
[9] sklearn.ensemble.RandomForestRegressor. scikit-learn. Retrieved October 10, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
[10] 12 Rental Pricing Factors 2022: How to Price Your Rental Property. Retrieved October 10, 2023, from
https://belonghome.com/blog/rental-pricing-factors
[11] Determining How Much You Should Charge for Rent. Retrieved October 10, 2023, from https://smartasset.com/mortgage/how-much-you-should-charge-for-rent
[12] When to avoid Random Forest? Retrieved October 10, 2023, from https://stats.stackexchange.com/questions/112148/when-to-avoid-random-forest

7. Code Appendix

HouseRentProject

October 11, 2023

[1]:
```
import pandas as pd
import numpy as np
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
df = pd.read_csv('C.csv', delimiter = '\t')
```

[2]:
```
mask = df.loc[:,'baseRent'] > df.loc[:,'totalRent']
df = df.drop(df.loc[mask,:].index)
```

[3]:
```
df['date']
```

[3]:
```
0       May-19
1       May-19
2       Oct-19
3       May-19
4       Feb-20
         …
2995    Feb-20
2996    May-19
2997    Sep-18
2998    Feb-20
2999    May-19
Name: date, Length: 2993, dtype: object
```

[4]:
```
for col in df.columns:
    if df[col].dtype == 'float':
        df[col] = df[col].fillna(df[col].mean())
```

[5]:
```
for col in df.columns:
    if df[col].dtype == 'object':
        df[col] = df[col].fillna(df[col].mode()[0])
```

[6]:
```
dat = df
dat=dat.join(pd.get_dummies(dat.date, dtype=float)).drop(['date'], axis =1)
```

```
[7]: for col in ['pictureCount']:
           dat[col]=dat[col].astype('float')
```

```
[8]: min_value = dat['totalRent'].min()
     max_value = dat['totalRent'].max()
     dat['normalized_value'] = ((0.99 - 0.01) * (dat['totalRent'] - min_value) /␣
      ↪(max_value - min_value)) + 0.01
     X = dat.drop(['normalized_value','totalRent'], axis =1).to_numpy().reshape(-1,9)
     y = dat['normalized_value'].to_numpy()
```

```
[9]: X_train, X_val_and_test, y_train, y_val_and_test = train_test_split(X,␣
      ↪y,test_size=0.4, random_state=55)
     X_val, X_test, y_val, y_test = train_test_split(X_val_and_test,␣
      ↪y_val_and_test,test_size=0.5, random_state=55)

     linear=LinearRegression()
     linear.fit(X_train,y_train)
     y_pred_lin = linear.predict(X_val)
     mean_squared_error(y_val, y_pred_lin)
```

```
[9]: 0.000847166912977633
```

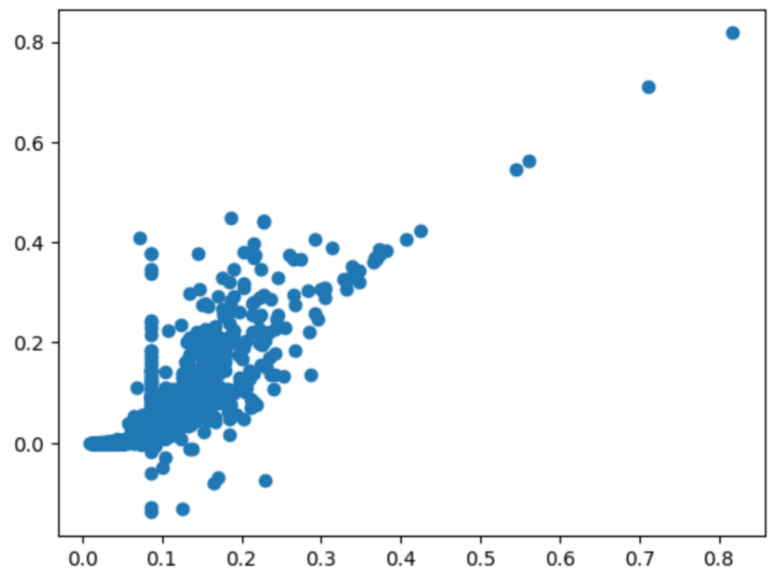
```
[13]: poly_tr_errors = []
      poly_val_errors = []
      poly_degrees = []
      lin_regr = LinearRegression(fit_intercept=False)
      for degree in range(1, 10):
          poly = PolynomialFeatures(degree=degree)
          X_train_poly = poly.fit_transform(X_train)
          lin_regr.fit(X_train_poly, y_train)
          y_pred_poly_train = lin_regr.predict(X_train_poly)
          poly_tr_error = mean_squared_error(y_train, y_pred_poly_train)
          X_val_poly = poly.fit_transform(X_val)
          y_pred_poly__val = lin_regr.predict(X_val_poly)
          poly_val_error = mean_squared_error(y_val, y_pred_poly__val)
          poly_tr_errors.append(poly_tr_error)
          poly_val_errors.append(poly_val_error)
          poly_degrees.append(degree)
      print(poly_tr_errors)
      print(poly_val_errors)
```

```
[0.0011293337508034103, 0.0005246424627486828, 0.000375317567786922,
0.0003528063562241336, 0.00017505217624280616, 0.0008040842774417295,
0.0008669343578439873, 0.0021565604946663184, 0.0041896432877787096]
[0.0008471669129776314, 0.0006823684127609282, 0.0036540394352645514,
0.3027930580693332, 113.27413852758623, 185932.02477131062, 6468236.4014236815,
140331382.51368305, 3024617050.690414]
```

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

      plt.plot(y_train, y_pred_poly_train, 'o')
```

```
[14]: [<matplotlib.lines.Line2D at 0x7f80f7064f10>]
```



```
[15]: lin_regr = LinearRegression(fit_intercept=False)
      poly = PolynomialFeatures(degree=2)
      X_train_poly = poly.fit_transform(X_train)
      lin_regr.fit(X_train_poly, y_train)
      y_pred_poly_train = lin_regr.predict(X_train_poly)
      poly_tr_error = mean_squared_error(y_train, y_pred_poly_train)
      X_val_poly = poly.fit_transform(X_val)
      y_pred_poly_val = lin_regr.predict(X_val_poly)
      poly_val_error = mean_squared_error(y_val, y_pred_poly_val)
      print(poly_tr_error)
      print(poly_val_error)
```

```
0.0005246424627486828
0.0006823684127609282
```

```
[22]: forest_tr_errors = []
      forest_val_errors = []
      m_depths = []
      for depth in range(10, 17):
          forest_regr = RandomForestRegressor(max_depth=depth)
          forest_regr.fit(X_train, y_train)
          y_pred_forest_train = forest_regr.predict(X_train)
          forest_tr_error = mean_squared_error(y_train, y_pred_forest_train)
          y_pred_forest_val = forest_regr.predict(X_val)
          forest_val_error = mean_squared_error(y_val, y_pred_forest_val)
          forest_tr_errors.append(forest_tr_error)
          forest_val_errors.append(forest_val_error)
          m_depths.append(degree)
      print(forest_tr_errors)
      print(forest_val_errors)
```

```
[0.00015780078227448031, 0.00014505069099183043, 0.00012569928938157624,
0.00011640012538904386, 0.00012416586265612804, 0.00012928822325607864,
0.0001122110117085392]
[0.0010541837220832948, 0.0010265204026362565, 0.0010276694970518937,
0.0010374618422396343, 0.0010793666345742037, 0.0010459245408130215,
0.0010558761020826504]
```

```
[ ]:
```

```
[26]: forest_regr = RandomForestRegressor(max_depth=11)
      forest_regr.fit(X_train, y_train)
      y_pred_forest_train = forest_regr.predict(X_train)
      forest_tr_error = mean_squared_error(y_train, y_pred_forest_train)
      y_pred_forest_val = forest_regr.predict(X_val)
      forest_val_error = mean_squared_error(y_val, y_pred_forest_val)
      print(forest_tr_error)
      print(forest_val_error)
```

```
0.00011597125813876844
0.0010797273276710432
```

```
[18]: X_test_poly = poly.fit_transform(X_test)
      y_poly_test_pred = lin_regr.predict(X_test_poly)
      poly_test_error = mean_squared_error(y_test, y_poly_test_pred)
      y_forest_test_pred = forest_regr.predict(X_test)
      forest_test_error = mean_squared_error(y_test, y_forest_test_pred)
      print(poly_test_error)
      print(forest_test_error)
```

```
0.0009929824396548252
0.001514240972891766
```

```
[ ]:
```