Cologne Rent Prediction

Matthew EsosaIgbinehi
Machine Learning and Scientific Computing
F07 - Faculty of Information, Media,
andElectrical Engineering,
Cologne University of Applied
SciencesCologne, Germany.

matthew_esosa.igbinehi@smail.th-koeln.de

***Abstract*— This paper presents the motivation, the approach, and the results of a project in Machine Learning, which was conducted in the course "Machine Learning and Scientific Computing" at the Cologne University of Applied Sciences. The goal of this project was to demonstrate the feasibility of predicting total rents of apartment units in the German city of Cologne via Machine Learning with a special focus on the Feature Importance. It means that it is examined which features have the strongest predictive power regarding the rent of a property. Furthermore, different models are compared to each other based on their errors on the test set to evaluate their quality of prediction. First, data concerning properties in Cologne were extracted from the original dataset that contained data on multiple German cities. This data was graphically illustrated in order to be able to recognize and remove invalid data points or to recognize interesting correlations between features as well as between a feature and the variable to be predicted. Such features with a too small correlation with base-rent were removed from the data set before training the model. The feature "livingSpace" turned out to be the feature with the highest correlation with the total rent. After first creating a preprocessing pipeline that scaled the dataset, 5 different models with the following names were trained: Random Forest Regressor, SGD Regression, SVM Regressor, Neural Network (multilayer perceptron), and XGBoost. After tuning the hyperparameters of the all the models, Random Forest Regressor and XGBoost were the most successful. Here, the average squared deviation from the actual rent was lowest with an RSME of 222.13 and 221, respectively.**

Keywords—machine learning, regression, Random Tree Forest, Stochastic Gradient Descent, XGBoost, Multilayer Perceptron, Support Vector Machine, real estate prediction, rent prediction

I. INTRODUCTION

Machine learning is the automated generation of knowledge from experience, i.e., from data. A system learns from this data and can generalize after this training and, e.g., make predictions or classify objects appropriately. The algorithms used in machine learning build a statistical model for this purpose. This model is tested using the test data to evaluate the quality of the model under investigation. Patterns and regularities in the data should be detected while avoiding overfitting the model to the data. Possible application areas of machine learning include time series analysis, such as of stock prices, automated diagnosis using images from tomography, detection of credit card fraud, speech and text recognition, and weather forecasting. In the project work we are conducting at the Cologne University of Applied Sciences, we would like to investigate whether it is possible to predict rents in Cologne using machine learning methods and a relatively small dataset (less than 3000 instances) with the help of various features.

II. MOTIVATION

Investments in real estate continue to be a popular financial investment. For an investor, it is of particular interest to know how much income can still be expected from renting out the property, as this is a key factor in determining how quickly the investment will pay for itself. In addition, it would be interesting to know what influence certain features like the size, the location and a recent renovation of the property would have on the rent. Predicting the expected rent by exploiting knowledge about other characteristics of a property would enable an investor to identify undervalued properties and use this information when making investment decisions. Since detailed datasets on real estate in Germany are freely available, we decided to use the example of Cologne to investigate whether it is possible to use such a dataset to train a model with machine learning techniques in such a way that good predictions about the rents of apartment units in Cologne are possible, taking into account various characteristics of a property.

III. METHODOLOGY

All computation carried out in this paper were done on Jupyter Notebook (version 6.4.11) using the data manipulation, statistical and machine learning libraries in Python (version 3.9.13). We want to build a model that will be trained on the features of the dataset to predict the total rent (‘totalRent’) including auxiliary charges for a new instance, meaning features of a specific apartment, the model was not trained on, without any information on the rent. To predict the rent price, we looked at the data set and visualized relationships, prepared the data for the machine learning algorithms, and tried different models to compare the results.

A. Looking at the data and preprocessing

The dataset for this project was obtained from the online machine learning platform 'kaggle.com' as referenced (it was scraped from 'immoscout24', one of the biggest real estate platform in Germany). The original data set had 49 features and 268850 instances. An explanation of some important features in the data set are given below.

- totalRent: Total rent (sum of the base rent and other auxilliary charges like heating cost and service charge)
- baseRent: The rent without electricity and heating
- heatingCosts: Heating costs per month in Euro
- regio1: Name of the state (same as geo_bln)
- regio2: Name of the city
- regio3: Name of the district
- geo_plz: ZIP code
- geo_bln: State (same as regio1)
- geo_krs: District (same as regio3)
- energyEfficiencyClass: Energy efficiency class
- lastRefurbish: date last renovated
- electricityBasePrice: price per month for electricity in Euro
- electricityKwhPrice: Electricity price per kWh
- noRoomsRange: Number of rooms (1 to 5)
- livingSpaceRange: living space range (1 to 7)
- yearConstructed: Year of construction
- scoutId: Immoscout Id
- noParkSpaces: Number of parking spaces
- firingTypes: Main energy sources
- yearConstructedRange: Binned construction year (range 1 to 9)
- livingSpace: Living space in square meter
- interiorQual: Interior quality
- street: Street name
- streetPlain: Street name (plain, different formatting)
- lift: Is elevator available
- baseRentRange: Binned base rent, 1 to 9
- typeOfFlat: Type of flat
- noRooms: Number of rooms
- thermalChar: Thermal energy efficiency of the apartment
- floor: describes on which floor the apartment is located
- serviceCharge: Extra costs such as electricity/internet in Euro
- telekomTvOffer: paid TV included or not?
- telekomHybridUploadSpeed: internet hybrid inter upload speed
- picturecount: How many pictures were uploaded for the advertisement?
- pricetrend: Price trend as calculated by Immoscout
- numberOfFloors: Number of floors in the building
- description: simple description of the apartment
- facilities: simple description of the facilities in the apartment

The focus was on predicting the total rent price in cologne hence. In order to achieve this goal, we filtered the dataset to obtain only instances regarding Cologne using the feature 'regio2'. This process significantly reduced the number of instances to work with, bringing the size of the data set down to 2709 instances, each containing 49 feature values. As we observed features with up to 80 percent missing values in the dataset, we decided to drop all features with more than 50 percent missing values. In addition to that we dropped all the unlabeled instances, where our target feature 'totalRent' was missing. Then the following need to be dropped, because they just show a range of a different feature, have too much text in it, that would need to be processed, or are simply not useful anymore, like the state or city (since this project is only about rents in Cologne). We also decided to drop the feature 'baseRent' because of the original motivation of this project. An investor should get an idea of the total rent that could be charged, so that the models that are created should be useful in a real-world scenario and the base rent of an apartment is probably unknown when the goal ist to predict the total rent. Since the base rent is part of the total rent, predictions should be based on other features that are more likely to be known (such as the size or location of an apartment). In conclusion it would not make sense to predict the total rent with the help of the base rent.

- no cheating: 'baseRent', 'baseRentRange'
- unnecessary location data: 'regio1', 'regio2', 'geo_bln', 'geo_krs', 'geo_bln'
- Features containing text: 'description', 'facilities',
- Other features in a different format: 'noRoomsRange', 'yearConstructedRange', 'streetPlain'

The resulting size of the dataset was 2277 instances containing 28 features. The following visualizations help understanding the importance of various features.

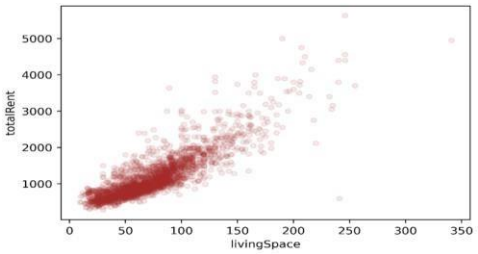

Fig. 1. Scattered plot of livingSpace against totalRent

Weight	Feature
0.7123 ± 0.1006	livingSpace
0.1125 ± 0.0949	serviceCharge
0.0431 ± 0.0216	geo_plz
0.0132 ± 0.0119	yearConstructed
0.0129 ± 0.0139	picturecount
0.0093 ± 0.0106	pricetrend
0.0058 ± 0.0083	floor
0.0052 ± 0.0059	numberOfFloors
0.0049 ± 0.0088	noRooms
0.0044 ± 0.0064	interiorQual_normal
0.0035 ± 0.0128	streetPlain_Am_Vorgebirgstor
0.0031 ± 0.0130	street_Am Vorgebirgstor
0.0031 ± 0.0044	hasKitchen
0.0029 ± 0.0078	interiorQual_luxury
0.0027 ± 0.0045	condition_well_kept
... 1902 more ...	

Fig.2. Feature importances

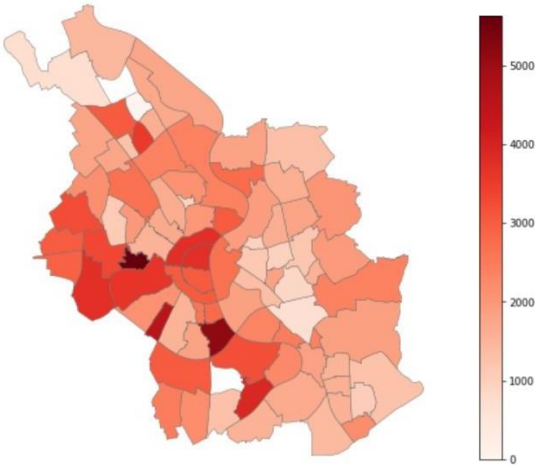

Fig.3: Heatplot: Total rent on a map of Cologne

From our histogram and scatterplot-visualization, the data set had no problem with outliers and as shown in fig.1 and fig.2, we observed that the most important feature that predicts 'totalRent' is 'livingSpace'. Since we already have the feature 'livingSpaceRange' in the data set, we used this categorical feature to split the data into a training set and a test set using Sklearn's 'StratifiedShuffleSplit' class and setting the random value to 29. The stratified shuffle splitting was done to provide a representative data set for both the training and testing. 20 percent of the data was kept as test set while 80 percent of the data was used in training the model. In addition to that, regio3 seems to be an important feature because it was used in fig.3 to create the 'heatplot' on the map of Cologne. By observing this figure, rents seem to be more expensive in the city-center and single other districts of Cologne.

Since most machine learning algorithms work with only numeric values, we decided to convert to convert the text/categorical features in our data to numeric representatives by one hot encoding them using the Sklearn's 'OneHotEncoder' class. This class creates an entirely new feature for each category of the original feature. Each instance then gets assigned a '1' or a '0', depending on the category it represented in the original feature. Each instance then only gets assigned one '1' (true new category) and $n - 1$ times '0' (false new category) with n being the number of new features created from one original feature. For ease of repeatability and proper code organization, we made use of the Sklearn's 'Pipeline' class to build a preprocessing pipeline to provide a well-prepared dataset to be trained on.

B. Model training

We are dealing with a supervised-learning regression problem (with the label [also called y- or dependent value] being the feature 'totalRent'. Hence, from the Sklearn library, we trained the following regression models and evaluated them to compare their results

• RandomForestRegressor: One popular predictive modeling approach used in machine learning is decision tree learning. It goes from observations about an object to inferences about the item's goal value via a decision tree. A decision tree generates an estimate by asking a series of 'yes-or-no' questions, each of which streamlines the range of potential answers until the model arrives at a final prediction. The decision tree regressor algorithm is not the best model to apply to data set with single feature. It is more adaptable to a data set with many features, and it does not require feature scaling. A 'RandomForestRegressor' is a result of many differently trained decision trees and therefore a version of ensemble learning. There are other versions of ensemble learning such as gradient boosting. In ensemble learning models of different types, or models of the same type are trained to later use all their predictions to create only one resulting prediction by using a so-called 'voting-process'. RandomForestRegressors generally perform significantly better than single decision trees and that is why it was preferred and selected.

• SGDRegressor: SGD (Stochastic Gradient Descent) is a straightforward but effective method for fitting linear classifiers and regressors. Sklearn's SGDRegressor class implements a simple stochastic gradient descent learning method that can fit a linear regression model with a variety of loss functions and penalties. For regression problems involving a high number of training instances, SGDRegressor is a good choice.

• SVMRegressor: In machine learning, Support Vector Machines are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. In Support Vector Regression, the straight line that is required to fit the data is referred to as hyperplane. The objective of a support vector machine algorithm is to find a hyperplane in an n-dimensional space that distinctly classifies the data points. The data points on either side of the hyperplane that are closest to the hyperplane are called Support Vectors. These influence the position and orientation of the hyperplane and thus help build the SVM. Hyperplanes are decision boundaries that is used to predict the continuous output. The data points on either side of the hyperplane that are closest to the hyperplane are called Support Vectors. These are used to plot the required line that shows the predicted output of the algorithm. A kernel is a set of mathematical functions that takes data as input and transform it into the required form. These are generally used for finding a hyperplane in the higher dimensional space. The most widely used kernels include Linear, Non-Linear, Polynomial, Radial Basis Function (RBF) and Sigmoid. By default, RBF is used as the kernel. Each of these kernels are used depending on the dataset. Boundary lines are the two lines that are drawn around the hyperplane at a distance ε (epsilon). It is used to create a margin between the data points. Unlike other Regression models that try to minimize the error between the real and predicted value, the SVR tries to fit the best line within a threshold value. The threshold value is the distance between the hyperplane and boundary line [7].

• XGBoost: Gradient Boosting works by sequentially adding predictors to an ensemble, each one correcting its predecessor. This method tries to fit the new predictor to the residual errors made by the previous predictor. It is worth noting that an optimized implementation of Gradient Boosting is available in the Python library XGBoost, (Extreme Gradient Boosting). This package was originally developed by Tianqi Chen as part of the Distributed (Deep) Machine Learning Community (DMLC). It aims to be fast,

scalable, and portable. Interestingly, XGBoost is often an important component of the winning entries in ML competitions so that we decided to use it in our project [8].

• Multilayer perceptron (MLP): An MLP is a neural network and counts as a deep learning model. A single perceptron consists of many threshold logic units (TLUs) where outputs and inputs are numbers, and each input connection has a weight. A TLU computes the weighted sum of inputs and then applies a step-function perceptron resulting in a single layer of TLUs with each TLU connected to all the inputs. The input of a perceptron is fed to special passthrough neurons that just output their input and they form the input layer. Furthermore, there is an extra bias-feature x0 = 1 added (represented as bias-neuron that outputs 1 all the time). A multilayer perceptron consists of more than one layer of perceptron. It has one input layer, one or more TLU-layers (hidden layers) and one output layer (also consisting of TLUs). Every layer after the input layer has one bias neuron and is fully connected to the next layer. An input-signal flows only in one direction (inputs to outputs), that is why it is also called a feedforward neural network. It is necessary to add a nonlinearity to each of these units, so that a non-linear activation modifies every output of a neuron.

C. Hyperparameter tuning

To improve the predictions as much as possible, we decided to run a the GridSearch algorithm on top of all the algorithms that were first with their standard default hyperparameters. To use GridSearch, the developer provides multiple values per hyperparameter that should be tested. The algorithm then trains multiple models using every possible hyperparameter combination and saves the model that provided the best result based on a cross validation that computes the RMSE. The improvements achieved by GridSearch are discussed in part IV.

model	*Initial Hyperparameters*	*Tuned Hyperparameters*
Random Forest Regressor	***max_depth= 9 n_estimators=100, max_features=1.0***	***max_depth=9, max_features=0.5, n_estimators=100***
SGD	***max_iter=1000, tol=1e-3***	***max_iter=1000, tol=0.09***
SVR	***C=1.0, epsilon=0.1, kernel='rbf'***	***C=200, epsilon=4, kernel='rbf'***
MLP	***Activation='relu', batch_size='auto', hidden_layer_sizes=100, learning_rate='constant', learning_rate_init= 0.001, max_iter=400***	***Activation='relu', batch_size='auto', hidden_layer_sizes=132, learning_rate= 'constant', learning_rate_init=0.08, max_iter=20***
XGBoost	***colsample_bytree=1, gamma=0, max_depth=6, min_child_weight=1, reg_alpha=0, reg_lambda=1***	***colsample_bytree=0.5, gamma=0.4, max_depth=4, min_child_weight=1.5, reg_alpha=50, reg_lambda=0.6***

Table 1: Hyperparameters before and after hyperparameter tuning

D. Ideas that did not work

After taking a look at the dataset, we had the idea to reduce the number of features in the dataset that got very large due to the one hot encoded category. Because 90% of the features were many that also did not correlate well (<0.1) we dropped every feature except for 50. After evaluating the performances of the models that were trained on this feature-reduced dataset, it was clear that performances got worse, so we decided to not drop the features.

The second idea was to use the feature 'baseRent' as the label since it might be the even more interesting one to predict for an investor. Unfortunately, the results were similar to those of from the first idea: performance did not increase for all the models so that this action also was reversed.

IV. RESULTS

For the solution of the regression problem, a total of five different types of algorithms or models from the field of machine learning were used: a Random-Forest-Regressor, an SVR-Regressor, an SGD-Regressor, a multilayer perceptron (deep neural network) and an XGBoost. The models are evaluated using the test-dataset that was separated from the training-dataset before the first training-process. It contains 20% of the original dataset. The chosen metric to compare the results of these models to each other is the root mean squared error (RMSE).

$$RMSE = \sqrt{\sum_{i=0}^{m} \frac{1}{m} (y_i - \hat{y})_i^2} \qquad (1)$$

The RMSE is widely used in machine learning and especially in regression because it punishes large outliers and keeps the same unit as the original variable. It should be compared to the mean of the dependent variable (y) of the test-dataset, which is 1190 USD in this case for the total rent. Given a 20% range of of tolerance for this error, a good model should have an RMSE smaller than 238 USD.

In addition to that, the R2- Score is also displayed. It is a measure of how well a model fits the data and especially in regression it shows how well the regression predictions approximate the real data points. It is typically between 0 and 1, with 1 being a perfect result indicating that the predictions made by the model perfectly fit the data [6]. Its definition is shown in (2).

$$R^2 = 1 - \frac{\text{sum squared regression (SSR)}}{\text{total sum of squares (SST)}} = 1 \frac{\sum_{i=1}^{n} (y - \bar{y})_i^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2} \qquad (2)$$

In this equation (2) $\hat{y}$ is a predicted value, $\bar{y}$ is the mean of all the values and y_i ist the real value.

Since two different approaches for training each of the models were used, two tables of information about the performances that were achieved by each model, are presented. First, we trained a baseline model with standard configurations to get a first look on how good the models would perform. Then we ran a hyperparameter-tuning algorithm to find the best hyperparameters for the model to

get the best results possible. Of course, we expected all the models with tuned hyperparameters to perform better than their baselines. In addition to that, regarding to our personal educational experiences, we expected the MLP (deep learning model) or XGBoost to perform the best. The results before hyperparameter-tuning are displayed in the following figure (fig. 4).

Metrics	RandomForeseRegressor	SVR Regressor	SGD Regressor	Multilayer Perceptron	XGBoost
RMSE	236.130695	630.150862	227.015789	269.128117	229.116880
R2_Score	0.861754	0.015449	0.872221	0.820416	0.869844

Fig. 4: First results with standard hyperparameters

After performing the hyperparameter-tuning, the following results, displayed in fig. 5 were achieved.

Metrics	RandomForeseRegressor	SVR Regressor	SGD Regressor	Multilayer Perceptron	XGBoost
RMSE	214.856520	305.594685	204.615773	202.359157	195.092330
R2_Score	0.885542	0.768452	0.896193	0.898470	0.905631

Fig. 5: Results after tuning the hyperparameters

It becomes clear that after tuning the hyperparameters results got better for every model except the SGD-regressor. In addition to that it can be said that performance only increased by a small amount for most models. By far the biggest jump in performance was achieved by the SVR-Regressor, where the RMSE could be reduced by over 50% from 630 USD to 306 USD.

As a general result it can also be stated that XGBoost performed the best with and without hyperparameter-tuning (RMSE: 229 USD; 195 USD). But also, the Random-Forest-Regressor (RMSE: 236 USD; 214 USD) performed well, but only after the tuning-process. The multilayer perceptron on the other hand disappoints especially with the standard configuration, as it just achieves an RMSE of 269 USD. But it performs well with good hyperparameters (RMSE: 202 USD). The SVR-Regressor also performed reasonably well, and the hyperparameter-tuning process also decrease the RMSE significantly (227 USD, 204 USD). Lastly, the by far worst performing model on our dataset is the SVR-Regressor (630 USD; 306 USD).

V. CONCLUSION

In conclusion, we can make suggestions for using the evaluated types of machine learning models for a regression task with the type of the dataset we worked with in this example. It included categorical and numerical attributes for regression, and we had to work with less than 3000 instances, which is typically not enough by machine learning standards. As expected, the XGBoost and the MLP performed the best and hyperparameter tuning should generally be considered to improve performance. Giving the lack of enough instances in the dataset, an RMSE of less than 20% of the mean value of the dependent variable in the test set is satisfactory for this example since the real focus of this project is the learning purpose. If this was a real-world business project, a larger dataset would have been mandatory to make accurate predictions that actually help making an investment decision or set the rent of an apartment unit to a reasonable level.

In the future, larger datasets should be used, and many more architectures and methods should be researched, implemented and then evaluated to try to improve performance even further. Moreover, to make a general statement about which model or algorithm gives the best results for similar problems, many more different datasets would have needed to train the chosen models on to evaluate and compare them in more detail.

REFERENCES

[1] Manasa J, Radha Gupta, Narahari N S, ”Machine Learning based Predicting House Prices using Regression Techniques”

[2] S. Abhishek.:Ridge regression vs Lasso, How these two popular ML Regression techniques work. Analytics India magazine,2018.

[3] REFERENCE-Dataset: https://www.kaggle.com/corrieaar/apartmentrental-offers-in-germany

[4] Jonas Neri - German Rent Avg ”https://www.kaggle.com/code/jonaslneri/german-rent-avg-by-postal-code”

[5] Regression ”https://scikit-learn.org/”

[6] Casella, Georges (2002). Statistical inference (Second ed.). Pacific Grove, Calif.: Duxbury/Thomson Learning. p. 556. ISBN 9788131503942

[7] Ashwin, Raj (2020), Unlocking the True Power of Support Vector Regression, Towards Data Science, https://towardsdatascience.com/unlocking-the-true-power-of-support-vector-regression-847fd123a4a0#:~:text=Support%20Vector%20Regression%20is%20a,the%20maximum%20number%20of%20points.

[8] Aurelien Geron (2019) Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow, Gradient Boosting, p.203, ISBN 978-1-492-03264-