Babes-Bolyai University, Cluj-Napoca, Romania
Faculty of Mathematics and Computer Science

Houses in Iowa, USA – get good deals and avoid scams

- Data Science -
- Intelligent Modeling Report -

Nicolae Morosan
Data Science
nicolae.morosan@stud.ubbcluj.ro

Table of Contents

1. Introduction

1.1. Why? What?

What is a better way to showcase an algorithm's effectiveness than using it for solving real-life situations? This is the case for the current paper and affiliated processes that made this report possible. In short, our problem relates to the real estate industry and how some people take advantage of different factors to get good deals; of course, we also tackle the situation where others get bad deals or simply get scammed, due to a number of subjective and/or objective reasons.

This work, algorithm and study is paramount in stabilizing the industry and creating more equity on the market – so that everybody could get a better chance of making his/her life better and avoid big mistakes. Simply put, become wiser.

Long story short, I will start with the problem, then explore and understand the data; afterwards, I will select from the data only what is relevant (for my future implementation) and, finally, draw some conclusions (regarding the topic itself, the algorithms and their reliability, the process itself).

1.2. How?

Well, I will explain to you the algorithms that I used in a later chapter. The idea is that I used 3 different algorithms, each having a completely different underlying mathematical approach. After we will have done all procedures, the results of the 3 will be showcased and conclusions will be drawn (as a basis, I will use the best algorithm).

1.3. Paper and structure

The current report contains various chapters, as follows: the first part is the introduction; the second part refers to the situation/context that generated the problem and the problem itself; the next part brings other studies to light and other people's contributions; the fourth part, the most detailed, contains my own work, the algorithms I used, information regarding data, and results I obtained; the last part will present the conclusions of this paper and future ideas (improvements, suggestions, different approaches, where the focus should be etc.)

2. Context description (Problem to solve)

The first and most important aspect of my work lies in its motivation – where this problem came from. This being said, we will talk about the real estate industry and hard realities people face on a daily basis.

Real estate has long been one of the sweetest spots in people's investment portfolios (and not only – since everybody needs a roof over their heads) and is part of Maslow's pyramid, being part of the lowest and most foundational layer (Physiological needs)[1]. However, when we think about each individual, from the connections, financial resources, intel[1] and industry/field knowledge points of view, we can easily observe that: 1. some people get very advantageous deals and 2. others find themselves on the other side of the spectrum – they are offered bad deals or, worse, they get scammed. This is why I wanted to create an algorithm (that I hope is as far from the term "biased" as possible) - to help people spot good deals and avoid scams.

Now, if we go even further, some people work their entire lives to pay a loan from a bank that will cover the apartment they live in and just that – the bank gets interest (a lot), the owner of the apartment could be very cunning and obtain an overpriced paycheck from a normal person (or even exploit him/her) etc.; and these things add up for the honest and humble individual.

One the other side, business tycoons get to vastly increase their reach and influence in a specific area (their neighborhood, city, state or even country), therefore increasing the "monopoly coefficient", if there is such a thing (if not, I'm referring to how often monopoly occurs) – the population will polarize even more. And this means the structure of society will get less and less solid and sound – in years, it could lead to crisis.

[1] intel = here I refer to the information that you cannot find on the surface of the internet or newspapers; I refer to information from friends and connections - "tips", "suggestions"

3. State of the art / Related work / Relevant articles and discoveries

3.1. Algorithms

Regarding algorithms, the state of the art for predicting house prices (or similar problems) are Gradient Boosting Machines, Support Vector Machines, Random Forests and Neural Networks (these usually register huge accuracy compared to other algorithms, but require a large number of rows/entries/instances – since it is very complex and customizable, from a mathematical point of view).

3.2. Regularization techniques

Help prevent overfitting and improve generalization. Some popular options are Ridge Regression [2](used for datasets with a lot of features – high correlation), Lasso Regression[3](least absolute shrinking and selection operator – variable selection and regularization) and Elastic Net[4](linearly combines penalties of the lasso and ridge methods).

3.3. Work from others

Others also used Random Forests (like I did – whoops, spoilers); some used SimpleImputer to deal with missing data; some used StandardScaler for normalization; Some used one-hot encoder to deal with categorical features to be able to train the model (all features **must** be numerical); some went further and showcased base models (Ridge, LGBMRegressor – light gradient boosting), meta-models (LinearRegression) and stacking models (StackingRegressor) – apparently, here we can set 2 parameters: 1. Estimators = base models and 2. Final Estimator – meta model. It's some sort of compound model that is much more robust, has more flexibility when it comes to learning the optimal combination and it's predictive power is improved heavily.

4. My own contribution, work and approach

As I've previously stated, here I will detail my approach, the models I used, how I implemented them and some of my personal journey while making this project. I will talk about the data and the business case as well.

4.1. Data

My dataset is from Kaggle and it's about houses in Iowa, USA. The dataset has 1460 rows and 82 features. Out of these 82 features, around half are numerical and half non-numerical. Some examples of features are: if the house has a porch, if the house has an alley, if it has a garage, what is the condition of the garage, the overall quality of the house, the sale price (which is our target variable – we want to predict that) etc.

4.2. Algorithms

4.2.1. Linear Regression

The first, easiest, most basic algorithm I used is linear regression. I took all the processes step by step and understood each individual procedure.

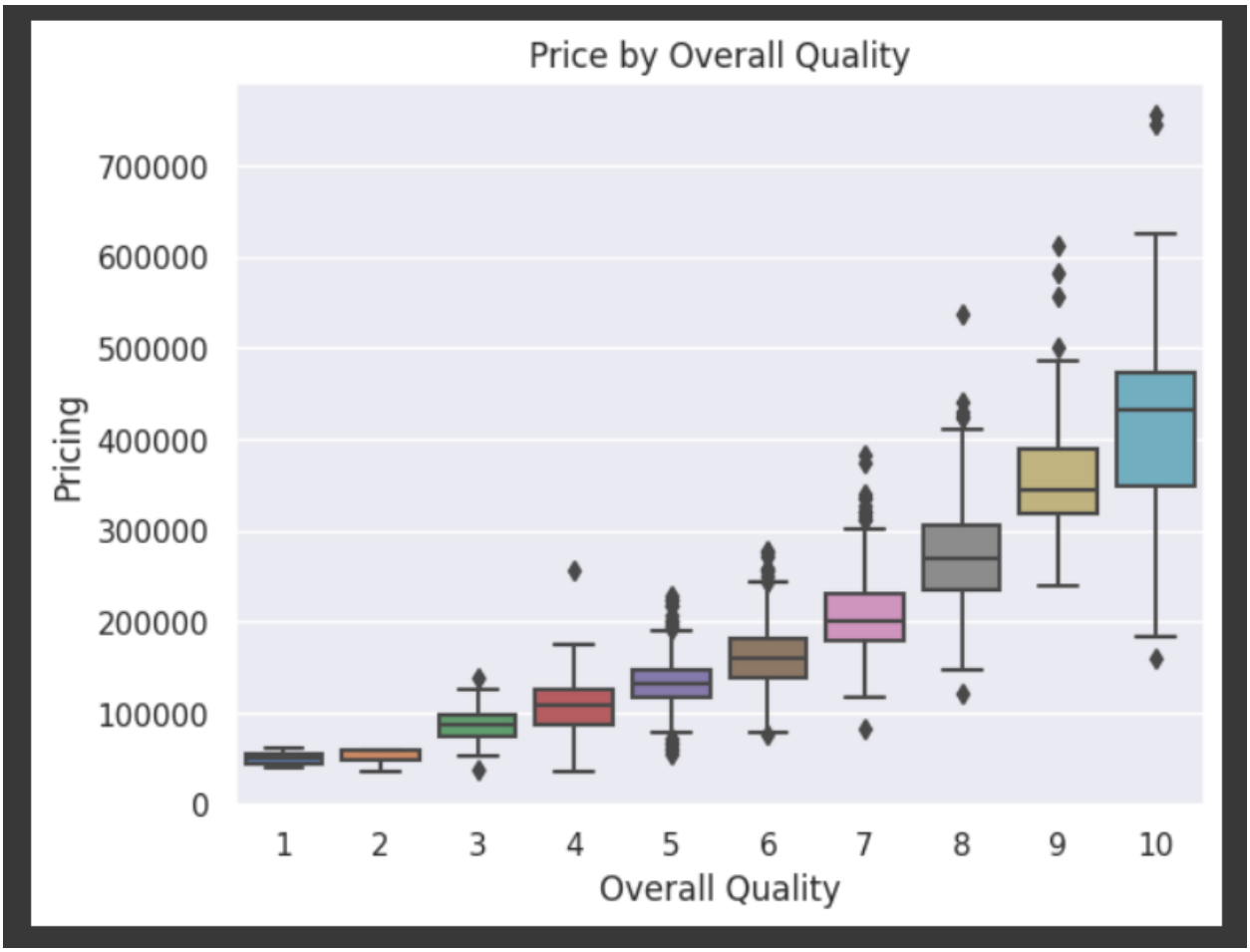
First, I uploaded the dataset.

```
df_house_prices_train = pd.read_csv('/content/sample_data/train.csv')
df_house_prices_test = pd.read_csv('/content/sample_data/test.csv')
#df_clustering = pd.read_csv('/content/sample_data/CC GENERAL.csv')
```

Then I did some basic statistics, just to check what kind of data we work with.

	Id	MSSubClass	LotFrontage	LotArea	OverallQual
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000
mean	730.500000	56.897260	70.049958	10516.828082	6.099315
std	421.610009	42.300571	24.284752	9981.264932	1.382997
min	1.000000	20.000000	21.000000	1300.000000	1.000000
25%	365.750000	20.000000	59.000000	7553.500000	5.000000
50%	730.500000	50.000000	69.000000	9478.500000	6.000000
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000
max	1460.000000	190.000000	313.000000	215245.000000	10.000000

Then I did some visualizations to actually see the data – for example, in the next figure I plotted the sale price of houses by the overall quality of the house – it's important to detect outliers (they can screw your predictions and calculations).



Then I checked the .csv file and I saw I have a lot of missing data. This is where I filled it with the median (this is one of the techniques) – because the model needs all data.

```
df_house_prices_train = df_house_prices_train.fillna(df_house_prices_train.median())
df_house_prices_test = df_house_prices_test.fillna(df_house_prices_test.median())
```

Now, you can see in the picture above that there are some outliers (those diamonds that are out of the boxplots). One way to improve the accuracy of our model is to remove them. Now, the methods might differ or you might choose different thresholds. I used the interquartile range (so, basically everything that is out of the interval [Q1 – 1.5*IQR; Q3 + 1.5*IQR] is removed).

```
# Define the lower and upper bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
```

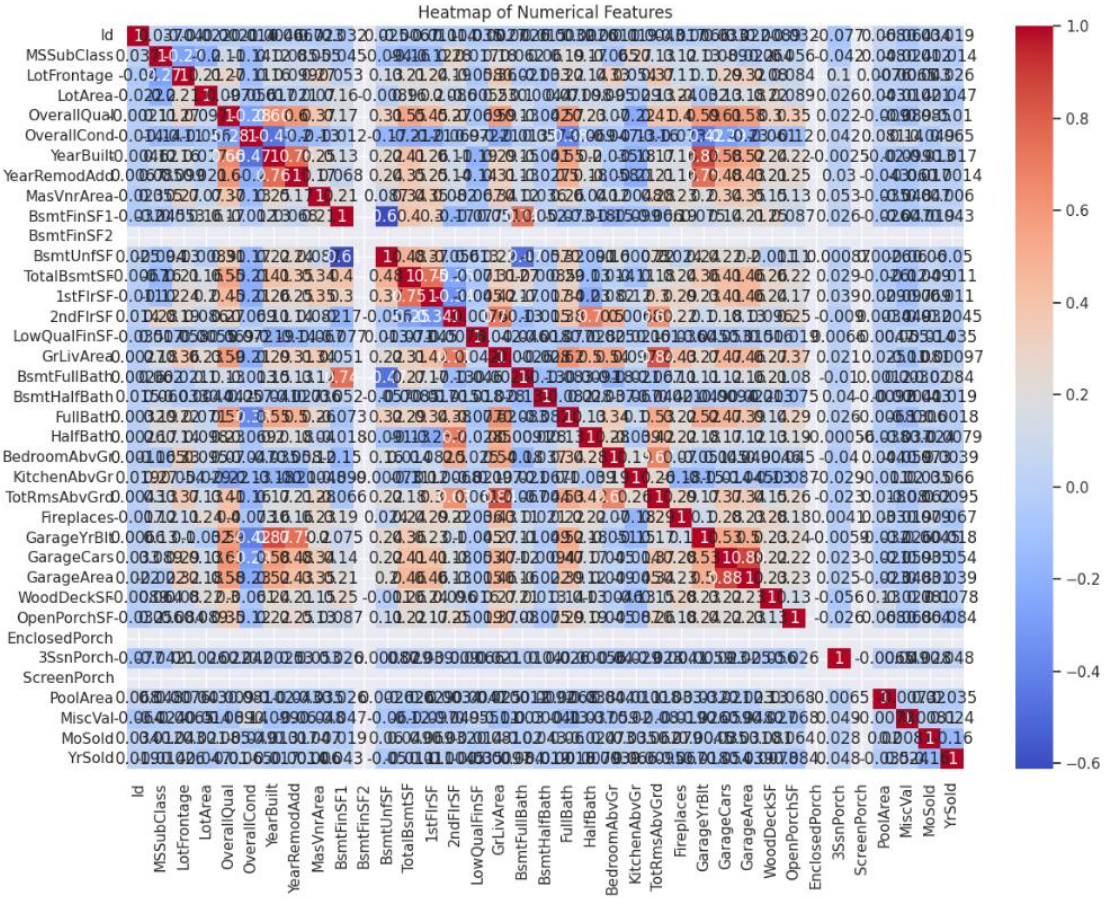
This is how we detected outliers from different features.

```
Feature: EnclosedPorch, Total Number of Outliers: 208
Feature: BsmtFinSF2, Total Number of Outliers: 167
Feature: OverallCond, Total Number of Outliers: 125
Feature: ScreenPorch, Total Number of Outliers: 116
Feature: LotFrontage, Total Number of Outliers: 106
Feature: MSSubClass, Total Number of Outliers: 103
Feature: MasVnrArea, Total Number of Outliers: 98
Feature: BsmtHalfBath, Total Number of Outliers: 82
Feature: OpenPorchSF, Total Number of Outliers: 77
```

I chose to eliminate the outliers where there were more than 100 outliers + the one with 98 – 7 features who had their outliers removed. This left me with 798/1460 rows.

```
Shape of data with outliers: (1460, 81)
Shape of data without outliers: (798, 81)
```

After this, you have to remember that there are outliers, ok, but there are also features that are correlated with each other – and this makes our model biased. We have to take care of them. I chose to remove all features with correlation above 0.6 and under -0.36 (my interval is [-0.6 , 1], so I had to somewhat keep the same proportion – both on positive and negative correlations). This is the initial correlation matrix.



After I performed the cleaning, I was left with a lot less features.

```
Shape of original dataset: (1460, 81)
Shape of x_75 before correlation removal: (798, 80)
Shape of x_75_final (after correlation removal): (798, 62)
```

As you can see, we removed 19 features.

Fun fact: Firstly, I forgot to remove the "SalePrice" column from my dataset and it kind of flagged it as correlated (of course =)))), so it removed it. And I was left wondering why the model wouldn't work. Is there something wrong with my "SalePrice" column? Oooh, apparently I don't even have such a feature. So yes, I returned, split them and then performed the correlation removal.

```
y_train = df_house_prices_train[['SalePrice']].copy()
x_train = df_house_prices_train.drop(['SalePrice'], axis=1)
```

Now, we need some final adjustments. Since half of these features are numerical and half non-numerical, we have 2 options (that I know of): we either use some encoding (such as one-hot encoding), or we simply eliminate all non-numerical features. So, let's detail a bit. If we did one-hot encoding, we will have around 200 features from the 60 we had (search how it works on the internet). The idea is that it keeps a higher accuracy, since we use all features. I did this and my normalized error was ~8.5%, which means an accuracy of 91.5%. And that is impressive. However,

my coefficients were very high and a normalization step was necessary (since the intervals were completely different – some features were between 0 and 120, others between 5000 and 15000, so we'd rather normalize them). Now, since one-hot encoding creates a lot of columns and fills them with 0s and 1s (we would have around 150 0s, 1 column with a 1 and others who were normalized – between 0 and 1). Try to imagine that. So, in my case, one-hot encoding would heavily interfere with the normalization (I used MinMaxScaler).

As a conclusion, I quit one-hot encoding, I eliminated all non-numerical features and I normalized it.

```
#One-hot encoding
#x_train_final = pd.get_dummies(x_train_final)

# We want to see all columns of the final training
#print(x_train_final.columns.tolist())
```

```
# We tried one-hot encoding and it's too much to clean for the time being;  we'll just stick with the numerical features
x_train_final = x_train_final.select_dtypes(include=['number'])
```

```
# Normalization
scaler = MinMaxScaler()
x_train_final = pd.DataFrame(scaler.fit_transform(x_train_final), columns=x_train_final.columns)
```

	Id	MSSubClass	LotFrontage	LotArea	MasVnrArea	BsmtFinSF2	\
0	0.000000	0.4	0.4375	0.049554	0.435556	0.0	
1	0.001375	0.4	0.4750	0.074495	0.360000	0.0	
2	0.002749	0.4	0.6750	0.101308	0.777778	0.0	
3	0.003436	0.3	0.6875	0.100016	0.000000	0.0	
4	0.004124	0.0	0.5625	0.064109	0.413333	0.0	
..	...	...	...	...	...	...	
793	0.996564	0.7	0.3750	0.054453	0.000000	0.0	
794	0.997251	0.0	0.6000	0.056787	0.431111	0.0	
795	0.998625	0.0	0.7500	0.127648	0.000000	0.0	
796	0.999313	0.0	0.4000	0.041091	0.000000	0.0	
797	1.000000	0.4	0.4000	0.044806	0.000000	0.0	

Then I trained the model.

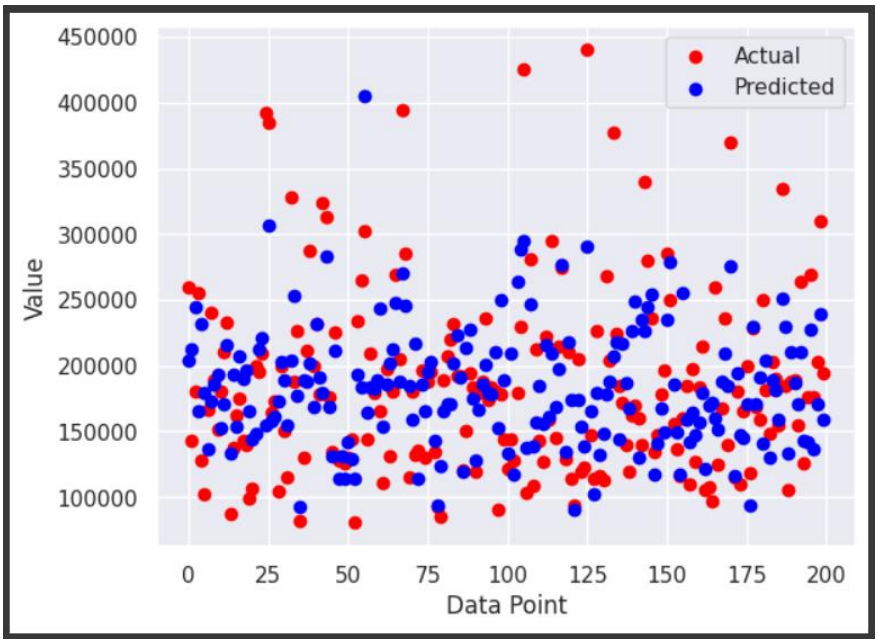
```
# Train the model (just the training set)
model = LinearRegression()
model.fit(x_train_final_75, y_train_final_75)

# Let's see coefficients and the intercept
print(model.intercept_)
```

```
[114589.69611522]
```

An intercept would mean that a price for a house would be ~114.500 if it wasn't influenced in any way by features (let's say each feature has a weight – if the building is older than 20 years, it would impact the price negatively. If it's new, positively. If the building is 7 years old, let's say, it doesn't influence the price. Now, do that for all features. And this is how you get the intercept).

After that, we made predictions, plotted the predicted data in comparison with the actual data (that I had saved beforehand) and evaluated the model using RMSE (and others, but this is more prominent).

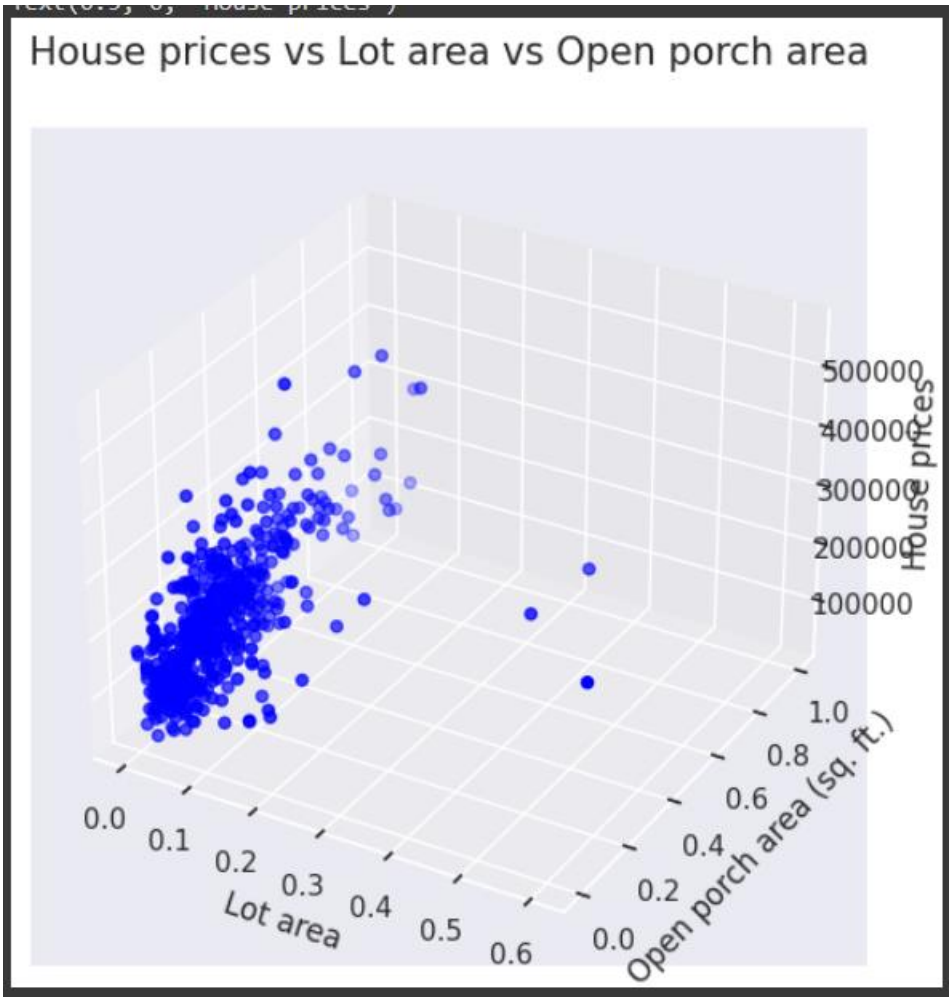



```
Normalized RMSE: 14.552379045773781 %
```

4.2.2. Random Forests

I also used Random Forests, since it is one of the most used algorithms when talking about numerical data.

Just for the record, I **must** use the same data I used for linear regression so that I can compare their performance.



This is a 3d plot that looks cool.

```
model_rand_forest = RandomForestRegressor(n_estimators=100, min_samples_split=2, random_state=42)
```

For the model I used 100 trees (n_estimators).

	Predicted values (Random Forests)	Actual/real values
0	162322	260000
1	189626	143000
2	208815	180000
3	189080	255900
4	192990	128000
..	...	...
195	249204	269500
196	163577	176000
197	192085	202500
198	261495	310000
199	188999	194000

[200 rows x 2 columns]

If you're wondering why there are 200 rows, this is because this is my testing dataset.

```
MSE Random Forests: 2410110614.83
MAE Random Forests: 34009.12
RMSE Random Forests: 49092.87743481736
Mean Random Forests: 182755.265
Normalized RMSE Random Forests: 13.655876894246832 %
```

It has a somewhat lower margin of error, which means a higher accuracy (1%, but still).

4.2.3. SVM (Support Vector Machine)

It's another algorithm commonly used in classification but can be used for numerical predictions as well. This is how I trained the model and got predictions out.

```python
# Create an instance of SVR model
model_svm = SVR(kernel='linear', C=100000, epsilon=0.1) # Linear (most used), Polynomial (more planes / dimensions), RBF (Radial Basis Function) ("clouds"), Sigmoid (0-1)

# Train the model
model_svm.fit(x_train_final_75, y_train_final_75)

# Make predictions on the test set
y_predicted_svm = model_svm.predict(x_train_final_25)

# convert from array to Series
y_predicted_svm = y_predicted_svm.reshape(-1)
y_predicted_svm = pd.DataFrame({'Predicted Sale Price': y_predicted_svm.astype(int)})

# Show predictions and actual values
df_comparison_svm = pd.concat([y_predicted_svm, y_train_final_25], axis=1)
df_comparison_svm.columns = ['Predicted values (SVM)', 'Actual/real values']
print(df_comparison_svm)
```

```
     Predicted values (SVM)  Actual/real values
0                    189407              260000
1                    197265              143000
2                    224325              180000
3                    154804              255900
4                    239544              128000
```

```
MSE SVM: 2895392741.14
RMSE SVM: 53808.85374304121
R-squared SVM: 0.3857151057877456
```

The RMSE are: ~5400 for SVM, ~52000 for linear regression, ~49000 for random forests. So, the differences aren't that big. The best remains the random forests so far.

Now, I played a little bit with the regularization parameter.

```python
# Change in C (regularization parameter) - RESULTS
# NOTE: R SQUARED MUST BE POSITIVE, otherwise we have a problem

# EPSILON = 0.1
# 0.1: r squared is -0.0222
# 1 / 2: r squared is -0.0217
# 10: r squared is -0.0195
# 50: r squared is -0.0079
# 100: r squared is 0.008969
# 500: r is 0.1
# 1.000: r is 0.1863
# 5.000: r is 0.33966
# 10.000: r is 0.37119
# 50.000: r is 0.38363
# 100.000: r is 0.3857

# Interpretation: An R^2 of 0.3857 means that about 38.57% of the total
# The rest (very large - 61.43%) cannot be explained using our model
```

As you can see, the R^2 score increased drastically until c reached 5000-1000. Afterwards, the increase was too small to be relevant. So, it settled to around 38%. This means that around 38% of the variance in our target variable (sale price) can be explained using the features (which is very small – like if those 30 columns can barely explain why prices fluctuate, than why would we need such a model). Well, it probably would've been higher if I kept all features (the initial 62) and found a way to normalize them without interfering with my features. But, this is another purpose of this research paper. To show people what **not to do**.

Now, the final step was to answer to the main problem. And how do we do that, exactly? Well, we get the predicted prices and we compare them to the actual prices. So, I subtracted the actual prices from the predicted prices and obtained a number for each row. That number, if positive, means that the predicted value is higher than the actual value, which means it's a good deal. If the actual value is higher, it means the model didn't find any mathematical reason to ask that price – a scam or overevaluation by the owner.

So, I created this "gap" for each of the 3 models and ordered them from largest to smallest by the **random forests algorithm**, since it was the best. And then I pulled the top 10 highest differences (both negative and positive).

Good deals / Steals:

	Gap (Regression)	Gap (Random Forests)	Gap (SVM)
103	83979	120357	87172
98	71888	110836	68684
18	56382	93166	55626
163	61363	83022	56396
87	63330	80764	47335
19	65849	78289	59631
188	27402	75069	47973
142	95203	73093	88745
151	28795	71203	22266
164	75603	66426	89225

Bad deals / Scams:

	Gap (Regression)	Gap (Random Forests)	Gap (SVM)
38	-85067	-99937	-93085
186	-83412	-100009	-90934
180	-108675	-107187	-116784
143	-113067	-113345	-118025
67	-124503	-115925	-152268
105	-130736	-123457	-134925
133	-170143	-128091	-178954
42	-144406	-133586	-152032
32	-124225	-133918	-125245
25	-78026	-161012	-100378
24	-237245	-258982	-241774

Let's talk about it a little. Let's take the house with id 103. Its gap is ~+120000, which means the owner heavily undervalued it and the model thinks it has much more value that what the seller is

asking. So, clearly this is a good deal – if you have the money, grab it; even if you don't need it; you can sell it.

On the other hand, look at house number 24. Its gap is -~250000, which means that the seller asked 250k more than what it's worth. So, you either show them the model and real value (to negotiate), or simply avoid it. Don't get scammed.

5. SWOT Analysis

5.1. Strengths

1. Pretty accurate predictions: Well, right now the accuracy is at ~87% (random forests), which is pretty good. If I would've let all 80 features and just performed one-hot encoding, it would've increased to ~93%.
2. Feature importance / impact: These models cand find features that weigh more than others, meaning we can deduce which features should be taken into account more than others (like the year the house was built – if it's old, it heavily (negatively) impacts the price of the house).
3. Robust Evaluation: Well, I obtained some metrics that are used for evaluation (this is an important step in validating the model), which tell me a lot about its reliability - these metrics have been used since the dawn of Data Science and even before (after all, mathematics is very old). If they were not working or relevant, there would've been other metrics/measurements already.

5.2. Weaknesses

1. Data privacy: I used Google Colab and I introduced a dataset from Kaggle. But imagine if I were to upload an internal database with a lot of confidential information. What if those were leaked, due to a cyberattack or something? Fines for GDPR data loss or mishandling are absolutely huge.
2. Computational resources: Well, my dataset is actually pretty small and the data is not complex (text and numbers). But if my dataset was larger, I would've needed a lot more raw power to process them – or alternative methods (parallel computing).
3. What I've done mostly assumes that the phenomenon can be modeled in a linear way, which might not be necessarily true. For example, in real estate, the prices by number of rooms don't follow a straight line, but rather a parabolic distribution (because you take into account demand and offer; apartments with 2 rooms are usually the most requested ones, so their prices might be, let's say, more than half of an apartment with 4 rooms).

5.3. Opportunities

1. A lot of people could be looking for these predictions, especially that in the modern era society tends to be polarized. Also, real estate properties represent a fundamental layer in our lives.
2. My project is about houses in Iowa, USA, but it can be replicated for any area that you wish – after all, anyone on the globe wants to get good deals and avoid scams.
3. Even some partnerships could arise from this – maybe with certain real estate agencies – expand the dataset (therefore training the model with more data or validate it on a larger scale), access to domain expertise (which could confirm or deny the impact of a feature, for example – or adjust its importance, in order to make the model more accurate).

5.4. Threats

1. These calculations have probably been done already in your area (by someone). There could be a high level of competitiveness in the area. So we must make our model pretty accurate, so people can trust it.
2. Some people might not appreciate the honesty of this model, if you get what I'm saying.
3. Regulatory changes: A lot of external factors contribute to the fluctuations in the market (wars, pandemics, students in the city, birth rate, mortality rate, growth of the city, population, number of buildings etc.). So this model has to be updated every so often to remain relevant.

6. Conclusions and future work

6.1. Conclusions

So, for some obvious rows, it can at least predict if the houses are overpriced or good deals. The accuracy is at 85%ish, so it's not that bad. This means that for most of the houses it will predict right. The value might not be exactly the one predicted, but still – works. Just be careful around houses where the gap is lower that 25000 (this is not calculated somewhere, I'm saying it). You can look at the RMSE, which is ~50k. So, any gap lower than 50k should be investigated deeper.

6.2. Future ideas

Use a different number of estimators (trees) in Random Forests (less than and more than 100, which is the default).

Use Ridge, Lasso Regressions or Elastic Net.

I could use a StackingRegressor to input 2/3 base models as estimators and another model as the final estimator.

7. Bibliography

Data - https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/data?select=data_description.txt

[1] - https://www.simplypsychology.org/maslow.html

[2] - https://en.wikipedia.org/wiki/Ridge_regression

[3] - https://en.wikipedia.org/wiki/Lasso_(statistics)

[4] - https://en.wikipedia.org/wiki/Elastic_net_regularization

https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

https://en.wikipedia.org/wiki/Random_forest

https://en.wikipedia.org/wiki/Support_vector_machine

https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

https://scikit-learn.org/stable/modules/svm.html

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

Google Images