Universitat Autònoma de Barcelona

Applied Machine Learning: Analysis and Implementation

Research Innovation (Python Report)

Carlos Mougan *Jeremy J. Williams*

October 2018

Abstract

In a world of growing volumes and varieties of data, coding manually can become a problem to achieve relevant insight and knowledge. Using machine learning, we are able develop a solution by understanding the construction of algorithms and applications in the right scope of data sources. And based on the awareness of improvement of computer programs, we can now access data, analyze it and learn from it.

The focus will be to study supervised machine learning algorithms on the process of predicting a continuous variable through machine learning techniques.

An official dataset will be used with python libraries and original implementations will be developed using regression algorithms and models. More specifically, prediction of house prices based on single and multiple features will be conducted.

The regression analysis, study and implementation were explained using selected supervised machine learning algorithmic models. Results of applied machine learning discovered were also deliberated.

Contents

1 Introduction

SINCE the dawn of time, computers were known as a machine used to input data and process it to generate a relevant output. Today, they have made human life stress-free and over time have developed into the key solution to all daily computational tasks. Over the last decade, these computational tasks begin to establish a steadily increasing in the amounts of data that became available for all kinds of business needs. This kind of data growth reduces the amount of knowledge that we need to achieve today´s normal relevant objectives.

It is widely known, that what we lack in knowledge, we make up for in data. This is where we can establish a form of learning, not only for technical valuation and data recovery, but a combination of two capabilities of a computer system to make it perform learning task and make coherent results according to previously observed conditions and previous actions or responses, and not only act according to an immovable strategy. We call this “Machine learning”. A kind of Learning that is imperatively needed when the task to be executed by the computer system or machine is too complicate to be explained in code or within any unknown conditional data mapping.

Within the realm of machine learning, it is very imperative when the task to be executed by the computer system is too complicate a relevant type of machine learning algorithm must be selected.

2 Definitions & Concepts

To conduct our study of applied machine learning, it is important to declare all essential definitions and concepts.

Generally, there are three (3) types of machine Learning algorithms.

They are as followed:

- **Supervised Learning**
- **Unsupervised Learning**
- **Reinforcement Learning**

2.1 Supervised Learning

Supervised Learning is a type of machine learning algorithm the consist of a dependent variable which that is to be predicted from a given set of independent variables. Within this set of independent variables, a function must be generated to map inputs to the desired output. This in theory is called "training process". This process continues until the model achieves a successful level of accuracy on the training data. Illustrations of this type of learning algorithm are Regression, Decision Tree, Random Forest, KNN, Logistic Regression etc.

2.2 Unsupervised Learning

Unsupervised Learning does not have any dependent variable to predict. It is normally used for clustering analysis and segmentation. Illustrations of this type of learning algorithm are Apriori algorithm, K-means, Mean-Shift, DBSCAN, Expectation–Maximization, Hierarchical Clustering etc.

2.3 Reinforcement Learning

Reinforcement Learning is a type of machine learning algorithm that is trained to make specific decisions when a machine is exposed to an environment with continuous training using trial and error. This machine learns previous knowledge and attempts to capture the best possible outcome to create precise business resolutions. Illustrations of this type of learning algorithm are Markov Decision Process, Q-Learning, State-Action-Reward-State-Action, Deep Q Network, Deep Deterministic Policy Gradient etc.

3 Objectives

Our focus will be to study supervised machine learning algorithms on the process of predicting a continuous variable through machine learning techniques. More specifically, we want to predict house prices based on single and multiple features using regression analysis.

We will use the dataset from house sales in King County in Seattle, USA, with programming libraries and original implementations will be developed using regression algorithms and models.

In this report, we will first apply some data analysis techniques to summarize the main characteristics of the dataset. Then we will apply various machine learning algorithms, change some of the tuning parameters to see if we can make an improvement of the code.

4 Algorithmic Methods & Models

Regression analysis provides a "best-fit" mathematical equation for the value of variables. The equation maybe linear (a straight line) or curvilinear, but we will be concentration on the linear type.

The focus of this report is on just two types of variables, "y" and "x". They are called the dependent variable (y) and independent variable (x), since the typical purpose of this type of analysis is to estimate or predict what "y" will be a given value or values of "x".

We will use the following six (6) methods and/or models:

- **Simple Regression**
- **Multiple Regression**
- **Polynomial Regression**
- **Ridge Regression and Gradient Descent**
- **Lasso**
- **k-Nearest Neighbors**

We will now define our selected supervised machine learning algorithmic models.

4.1 Simple Regression

The simple regression model is a linear equation having a y-intercept and a slope with approximations of these population parameters based on sample data and determined by standard formulas.

The formula for the simple regression model is:

$$y_j = \beta_o + \beta_1 x_i + \varepsilon_i \tag{1}$$

where

y_j = a value of the dependent variable, y,

x_i = a value of the independent variable, x,

β_o = the y-intercept of the regression line,

β_1 = the slope of the regression line and

ε_i = random error, or residual

4.2 Multiple Regression

The multiple regression model is an extension of the simple linear regression model. However, there are two or more independent variables instead of just one. As before, estimates of the population parameters in the model are made on the basis of sample data.

The formula for the multiple regression model is:

$$y_j = \beta_o + \beta_1 x_{1i} + \beta_2 x_{2i} + ... + \beta_k x_{ki} + \varepsilon_i \tag{2}$$

where

y_j = a value of the dependent variable, y,

$x_{1i} + x_{2i} + ... + \beta_k x_{ki}$ = a value of the independent variable, x,

β_o = a constant,

$x_{1i} + x_{2i} + ... + x_{ki}$ = the slope of the regression line,

$\beta_1 + \beta_2 + ... + \beta_k$ = partial regression coefficients for the independent variables, $x_{1i} + x_{2i} + ... + x_{ki}$ and

ε_i = random error, or residual

4.3 Polynomial Regression

Continuing from a multiple regression model, a plot of the residuals versus a predictor may sometimes suggest there is a nonlinear relationship. It is called a *"polynomial regression model"*.

Such a model for a single predictor, x, is:

$$y_j = \beta_o + \beta_1 x_{1i} + \beta_2 x_{2i}^2 + ... + \beta_k x_{ki}^h + \varepsilon_i \tag{3}$$

where

h is called the degree of the polynomial.

Although this model is considered a nonlinear relationship between y and x, polynomial regression is still sometimes called a linear regression since, it is linear in the regression coefficients, β_1,β_2,...,β_h.

4.4 Ridge Regression and Gradient Descent

4.4.1 Ridge Regression

Ridge regression is type of shrinkage methods used to construct simple models with excessive descriptive extrapolative power. Such simple models explain data with minimum number of parameters or predictor variables. Within the process of ridge regression, it is normally used to create these simple model when the data set has multicollinearity or correlation exist between predictor variables.

The formula used for the ridge regression is:

$$\sum_{i=1}^{n}\left(y_i - \beta_0 + \sum_{j=1}^{p}\beta_i x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2 = RSS + \lambda\sum_{j=1}^{p}\beta_j^2 \tag{4}$$

where

RSS = Residual Sum Of Squares, which is a statistical procedure used to quantify the volume of variance in a data set that is not described by a regression model and

$\lambda\sum_{j=1}^{p}\beta_j^2$ = the shrinkage penalty with $\lambda \geq 0$ as a tuning parameter.

4.4.2 Gradient Descent

Gradient descent is used to minimize some function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient. Gradient descent is to update the parameters of our model.

The gradient is the direction of increase and therefore the negative gradient is the direction of decrease and we're trying to minimum the number of parameters or predictor variables.

The move in the negative gradient direction is called the *'step size'*.

4.5 Lasso

When using the ridge regression, there exist some disadvantages located in the penalty (3). The idea of the shrinking process is to shrink all coefficients to zero however, not all will be set to exactly zero. This will only happen when $\lambda = \infty$. This leads to the need of the *Lasso*, which is an alternative version to ridge regression that is used to correct this disadvantage.

The formula used for the lasso is:

$$\sum_{i=1}^{n}\left(y_i - \beta_0 + \sum_{j=1}^{p}\beta_i x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j| = RSS + \lambda\sum_{j=1}^{p}|\beta_j| \tag{5}$$

where

RSS = Residual Sum Of Squares, which is a statistical procedure used to quantify the volume of variance in a data set that is not described by a regression model and

$\lambda \sum_{j=1}^{p} |\beta_j|$ = the lasso penalty with $\lambda \geq 0$ as a tuning parameter.

In (3), the β_j^2 term in the ridge regression penalty changes to $|\beta_j|$ lasso penalty term to correct the disadvantage in the ridge regression.

4.6 k-Nearest Neighbors

k-nearest neighbors or k-NN is an algorithm that classifies an input by using its k nearest neighbors.

k-NN is known as data classification and regression algorithm that tries to govern what collection of data points its in by looking at the data points surrounding it.

Since our focus is on the regression case, in k-NN regression, the output is the property amount for the item. This amount is the mean of the values of its k nearest neighbors.

The k-NN regression method is closely related to the k-NN classifier method and using the formula:

$$\widehat{f}(x_o) = \frac{1}{K} \sum_{x_i \in G_o} \tag{6}$$

where

x_o = a prediction point,

K = given value or amount,

G_o = a group of training responses and

x_i = a trained response

5 Data Overview

As discussed earlier, we use a dataset for the sales coming from an approved public records of home sales in the King County Area, Washington State, USA.

The data set comprises of 21,613 rows.

Each characterizes of a home sold from May 2014 through May 2015.

Below is a breakdown of the variables involved:

- **Id:** Unique ID for each home sold
- **Date:** Date of the home sale
- **Price:** Price of each home sold
- **Bedrooms:** Number of bedrooms
- **Bathrooms:** Number of bathrooms, where .5 accounts for a room with a toilet but no shower
- **Sqft-living:** Square footage of the apartments interior living space
- **Sqft-lot:** Square footage of the land space
- **Floors:** Number of floors
- **Waterfront:** A dummy variable for whether the apartment was overlooking the waterfront or not
- **View:** An index from 0 to 4 of how good the view of the property was
- **Condition:** An index from 1 to 5 on the condition of the apartment
- **Grade:** An index from 1 to 13, where 1-3 falls short of building construction and design, 7 has an average level of construction and design, and 11-13 have a high quality level of construction and design.
- **Sqft-above:** The square footage of the interior housing space that is above ground level
- **Sqft-basement:** The square footage of the interior housing space that is below ground level
- **Yr-built:** The year the house was initially built
- **Yr-renovated:** The year of the house's last renovation
- **Zipcode:** What zipcode area the house is in
- **Lat:** Latitude
- **Long:** Longitude
- **Sqft-living15:** The square footage of interior housing living space for the nearest 15 neighbors
- **Sqft-lot15:** The square footage of the land lots of the nearest 15 neighbors

6 Empirical Analysis & Results

We now presents a thought process of predicting a continuous variable through applied machine learning methods.

More specifically, we want to predict house prices based on single and multiple features using regression analysis.

6.1 Data Characteristics

We now explore the data set and study it's characteristics:

- Column Types: We change the data set columns to categorical data
- Missing Values: We decided to drop them.
- Lengths and shapes: To see what's the amount of data that we are currently managing.

We start our analysis by loading some of the libraries that we will use later.

```
1
2  import pandas as pd
3  import numpy as np
4  %matplotlib inline
5  from scipy import stats
6  import matplotlib.pyplot as plt
7  import numpy as np
8  from sklearn.model_selection import train_test_split
9  from sklearn.neighbors import KNeighborsClassifier
10 import time
11 from sklearn.linear_model import LinearRegression
12 from sklearn.linear_model import Ridge
13 from sklearn.preprocessing import MinMaxScaler
14 from sklearn.linear_model import Lasso
15 import seaborn as sns
16 from sklearn.preprocessing import PolynomialFeatures
17 from sklearn.svm import SVC
18 from sklearn.svm import LinearSVC
19 from sklearn.tree import DecisionTreeClassifier
20 from sklearn.svm import SVR
21 from sklearn.dummy import DummyRegressor
22 from sklearn import metrics
23 from sklearn.metrics import r2_score
24 import warnings
25 warnings.filterwarnings('ignore')
26
27 #We read the data set
28 df=pd.read_csv('kc_house_data.csv')
```

Lets convert features to categorical type

```

df['waterfront'] = df['waterfront'].astype('category',ordered=True)
df['view'] = df['view'].astype('category',ordered=True)
df['condition'] = df['condition'].astype('category',ordered=True)
df['grade'] = df['grade'].astype('category',ordered=False)
df['zipcode'] = df['zipcode'].astype(str)
df.head(2) # Show the first 2 lines
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	0	1955
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	400	1951

2 rows × 21 columns

Figure 1: Converted features to categorical showing first 2 lines

We format the date

```

df.drop(columns=['date'],inplace=True)
```

Descriptive Analysis

```

df.describe(include='all')
```

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above
count	2.161300e+04	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+04	21613.000000	21613.0	21613.0	21613.0	21613.0	21613.000000
unique	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2.0	5.0	5.0	12.0	NaN
top	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0	0.0	3.0	7.0	NaN
freq	NaN	NaN	NaN	NaN	NaN	NaN	NaN	21450.0	19489.0	14031.0	8981.0	NaN
mean	4.580302e+09	5.400881e+05	3.370842	2.114757	2079.899736	1.510697e+04	1.494309	NaN	NaN	NaN	NaN	1788.390691
std	2.876566e+09	3.671272e+05	0.930062	0.770163	918.440897	4.142051e+04	0.539989	NaN	NaN	NaN	NaN	828.090978
min	1.000102e+06	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02	1.000000	NaN	NaN	NaN	NaN	290.000000
25%	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	NaN	NaN	NaN	NaN	1190.000000
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	NaN	NaN	NaN	NaN	1560.000000
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	NaN	NaN	NaN	NaN	2210.000000
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	NaN	NaN	NaN	NaN	9410.000000

Figure 2: Description of the dataset

With a simple correlation, we can see that there are some variables with a higher correlation like the *sqft living* and some others like *zipcode* or *id* or *longitude*(*earth*) have lower correlation.

Data Map Visualization

```

sns.heatmap(df.corr(),center=True,square=True)
```

We now look for the variables with the strongest correlation with price

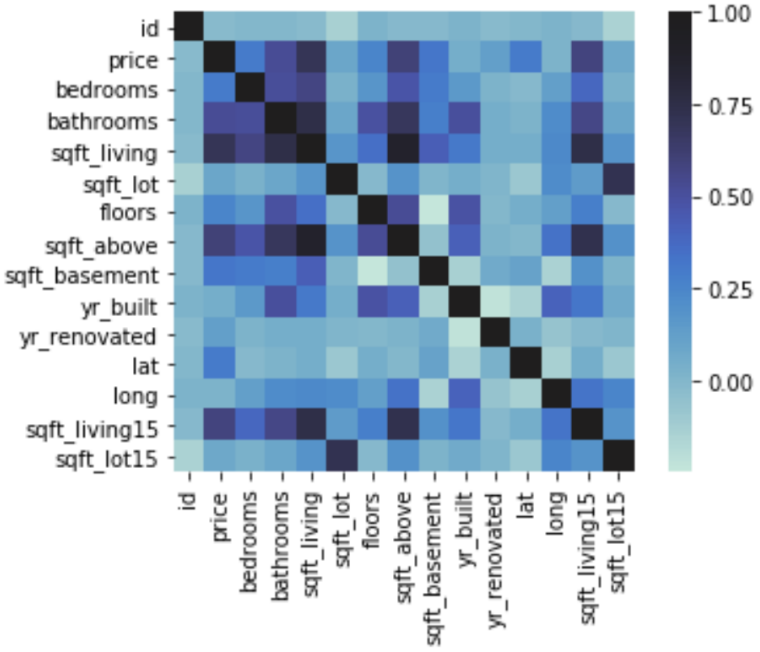

Figure 3: Heat Map of the correlations of some of the variables of the data set

```
1
2 correlation=df.corr()
3 correlation.sort_values(by='price',inplace=True)
4 price_correlation=pd.DataFrame(correlation['price'])
```

We now study a boxplot of the dataset.

```
1
2 fig, ax = plt.subplots(figsize=(12,4))
3 sns.boxplot(x = 'price', data = df, orient = 'h', width = 0.8,
4                fliersize = 3, showmeans=True, ax = ax)
5 plt.show()
```

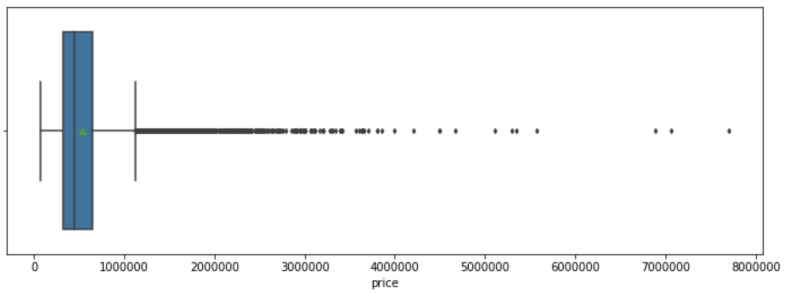

Figure 4: Box Plot of price in the data set.

There seems to be a lot of outliers at the top of the distribution, with a few houses above the 5000000 value.

If we ignore outliers, the range is illustrated by the distance between the opposite ends of the whiskers (1.5 IQR) - about 1000000 here.

Also, we can see that the right whisker is slightly longer than the left whisker and that the median line is gravitating towards the left of the box. The distribution is therefore slightly skewed to the right.

In figure 5, we can see the bi-variate relation of price in the data set.

```
1
2 sns.jointplot(x="sqft_living", y="price", data=df, kind = 'reg', size = 7)
3 plt.show()
```

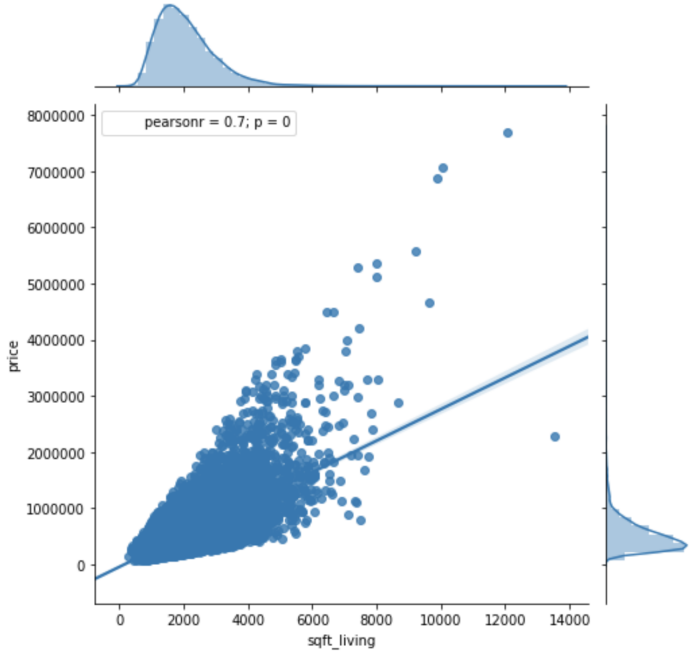

Figure 5: A joint plot of the bi-variate distribution of our higher related variable

6.2 Preparation

We now will apply some regression models.

We will study the model complexity and try to make selections of the best predictive model using different model tuning variables, a validation set or cross-validation techniques.

We first start to split the model in two different sets: training and testing.

```
1 from sklearn.cross_validation import train_test_split
2 X_train, X_test, y_train, y_test =
3 train_test_split(df_2, df_y, random_state=0)
```

We select Random State instance, random state is the random number generator; if random_state=None, the random number generator is the Random State instance used by np.random. But we select random_state=0 to be able to replicate the training of our models.

The test size is 0.25 by default and we let it be like that.

6.3 Regression Models

Now we start to train the different skclearn models with the training and scoring them with both splits training and testing.

In this section, we are going to use Liner Regression, Ridge's Regression and Lasso Regression, we study the difference when we applying the cost function.

The relevant formulas are as followed:

$$J_{(\theta)} = \frac{1}{n}\sum_{i=1}^{n}(y_i - y_i^{\wedge})^2 \tag{7}$$

Linear cost function

$$J_{(\theta)} = \frac{1}{n}\sum_{i=1}^{n}(y_i - y_i^{\wedge})^2 + \lambda\sum_{i=1}^{n} w_i^2 \tag{8}$$

Rigde Cost Function

$$J_{(\theta)} = \frac{1}{n}\sum_{i=1}^{n}(y_i - y_i^{\wedge})^2 + \lambda\sum_{i=1}^{n} |w_i| \tag{9}$$

Lasso Cost Function

6.3.1 Linear Regression

We now study the regression model with one feature.

```
1
2 linreg = LinearRegression().fit(X_train, y_train)
3
4 print('linear␣model␣coeff␣(w):␣{}'
5       .format(linreg.coef_))
6 print('linear␣model␣intercept␣(b):␣{:.3f}'
7       .format(linreg.intercept_))
8 print('R-squared␣score␣(training):␣{:.3f}'
9       .format(linreg.score(X_train, y_train)))
```

Figure 6: Graph of the linear regression for one feature

```
print('R-squared score (test): {:.3f}'
     .format(linreg.score(X_test, y_test)))

#Linear regression one feature
linreg_one_feature = LinearRegression().fit(X_train_one_feature,
    ↪ y_train_one_feature)

print('R-squared score (training): {:.3f}'
      .format(linreg_one_feature.score(X_train_one_feature, y_train_one_feature)))
print('R-squared score (test): {:.3f}'
      .format(linreg_one_feature.score(X_test_one_feature, y_test_one_feature)))

plt.figure(figsize=(5,4))
plt.scatter(df_3, df_y, marker= 'o', s=5, alpha=0.8)
df_3=pd.DataFrame(df_3)
b=linreg_one_feature.coef_ *df_3 + linreg_one_feature.intercept_
plt.plot(df_3,b, 'r-')
plt.title('Least-squares linear regression for one feature')
plt.xlabel('Feature value (x)')
plt.ylabel('Target value (y)')
#plt.ylim((0,600))
plt.tick_params(top=False, bottom=False, left=False, right=False,
    ↪ labelleft=False, labelbottom=False)
plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)
plt.gca().spines['left'].set_visible(False)
plt.gca().spines['bottom'].set_visible(False)
plt.show()
```

According to Table 1 & Figure 7, we can see that the best fit for our model is normal lineal regression with all features. After applying the algorithm to predict, instead of classifying, we obtain that R^2 is 0.537 (all features) and 0.462 for one feature (*sqft living*).

Model	R^2 Test	R^2 Test
LinReg - One Feature	0.490	0.483
LinReg - All Features	0.703	0.69
Ridge Regr	0.702	0.691
Ridge Regr Scaled	0.696	0.681
Lasso Reg	0.703	0.69

Table 1: The results obtained with this cost functions applied to this models are seen in the data table

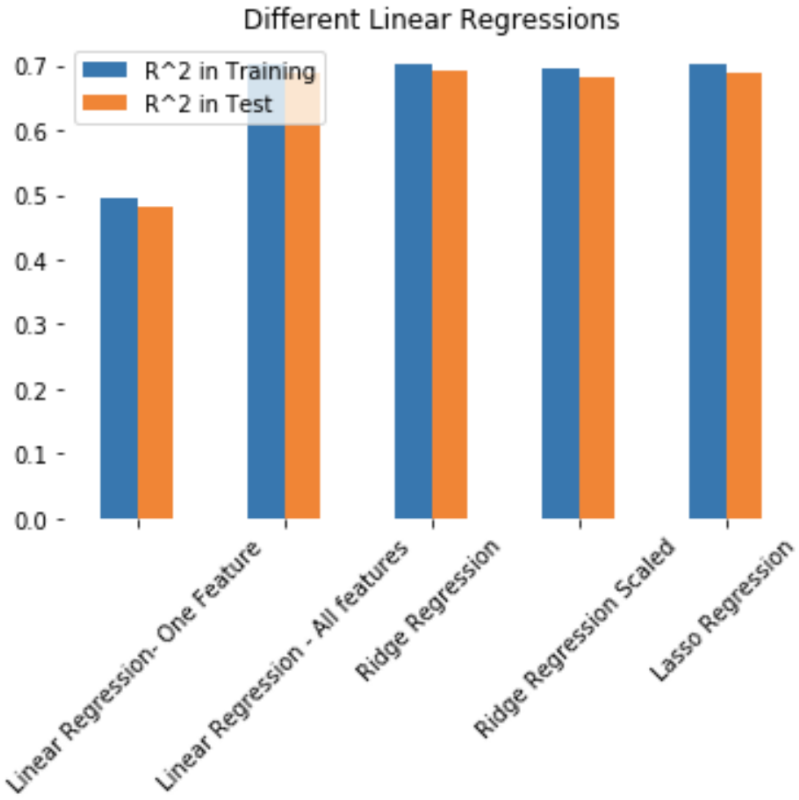

Figure 7: A bar representation of the scores of the different linear regression models

6.3.2 Polynomial Regression

In this section, we now study the data set to confirm if it fits better in a linear and/or nonlinear function.

```
# Degree = 2
print('\nNow we transform the original input data to add\n\
polynomial features up to degree 2 (quadratic)\n')
poly = PolynomialFeatures(degree=2)
X_F1_poly = poly.fit_transform(df_2)
```

```
X_train, X_test, y_train, y_test = train_test_split(X_F1_poly, df_y,
                                                    random_state = 0)
linreg = LinearRegression().fit(X_train, y_train)


print('(poly deg 2) R-squared score (training): {:.3f}'
     .format(linreg.score(X_train, y_train)))
print('(poly deg 2) R-squared score (test): {:.3f}\n'
     .format(linreg.score(X_test, y_test)))

print('\nAddition of many polynomial features often leads to\n\
overfitting, so we often use polynomial features in combination\n\
with regression that has a regularization penalty, like ridge\n\
regression.\n')

X_train, X_test, y_train, y_test = train_test_split(X_F1_poly, df_y,
                                                    random_state = 0)
linreg = Ridge().fit(X_train, y_train)


print('(poly deg 2 + ridge) R-squared score (training): {:.3f}'
     .format(linreg.score(X_train, y_train)))
print('(poly deg 2 + ridge) R-squared score (test): {:.3f}'
     .format(linreg.score(X_test, y_test)))
```

```
Now we transform the original input data to add
polynomial features up to degree 2 (quadratic)

(poly deg 2) R-squared score (training): 0.700
(poly deg 2) R-squared score (test): 0.660


Addition of many polynomial features often leads to
overfitting, so we often use polynomial features in combination
with regression that has a regularization penalty, like ridge
regression.

(poly deg 2 + ridge) R-squared score (training): 0.644
(poly deg 2 + ridge) R-squared score (test): 0.449
```

Figure 8: Results with degree = 2

```
# Degree = 3
print('\nNow we transform the original input data to add\n\
polynomial features up to degree 3 (cubic)\n')
poly = PolynomialFeatures(degree=3)
X_F1_poly = poly.fit_transform(df_2)
X_train, X_test, y_train, y_test = train_test_split(X_F1_poly, df_y,
```

```
                                                random_state = 0)
linreg = LinearRegression().fit(X_train, y_train)


print('(poly deg 3) R-squared score (training): {:.3f}'
     .format(linreg.score(X_train, y_train)))
print('(poly deg 3) R-squared score (test): {:.3f}\n'
     .format(linreg.score(X_test, y_test)))

print('\nAddition of many polynomial features often leads to\n\
overfitting, so we often use polynomial features in combination\n\
with regression that has a regularization penalty, like ridge\n\
regression.\n')

X_train, X_test, y_train, y_test = train_test_split(X_F1_poly, df_y,
                                                random_state = 0)
linreg = Ridge().fit(X_train, y_train)


print('(poly deg 3 + ridge) R-squared score (training): {:.3f}'
     .format(linreg.score(X_train, y_train)))
print('(poly deg 3 + ridge) R-squared score (test): {:.3f}'
     .format(linreg.score(X_test, y_test)))
```

```
Now we transform the original input data to add
polynomial features up to degree 3 (cubic)

(poly deg 3) R-squared score (training): 0.278
(poly deg 3) R-squared score (test): 0.236


Addition of many polynomial features often leads to
overfitting, so we often use polynomial features in combination
with regression that has a regularization penalty, like ridge
regression.

(poly deg 3 + ridge) R-squared score (training): 0.602
(poly deg 3 + ridge) R-squared score (test): 0.545
```

Figure 9: Results with degree = 3

Model	R^2 Test	R^2 Test
LinReg	0.703	0.69
Pol Reg =2	0.700	0.660
Pol Reg=3	0.280	0.240
Pol Reg=2 + Ridge	0.644	0.449

Table 2: The results obtained with this cost functions applied to this models are seen in the data table.

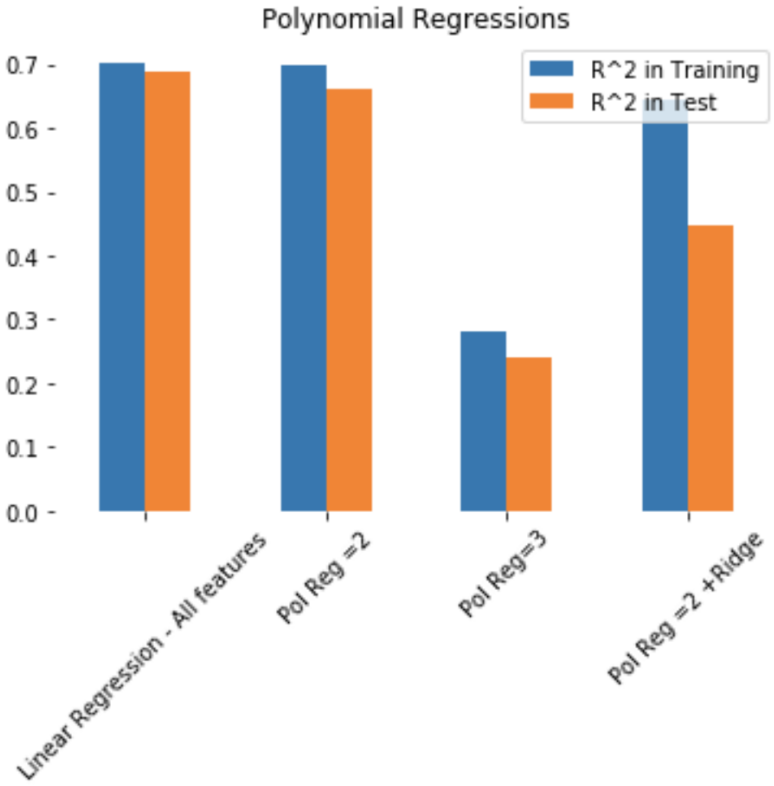

Figure 10: A bar representation of the scores of the different linear regression models

According to Table 2 & Figure 10, we can also see that the best fit for our model is lineal regression with all features.

However, looking at the polynomial regression model with degree = 2, the results is very close to the result of lineal regression with all features. This tell us that polynomial regression model with degree = 2 can be selected as a good fit model.

Now, we will try to score the best options with different alpha parameters for Ridge and Lasso.

6.3.3 Ridge Regression Alpha Parameter Tuning

```
ridge=pd.DataFrame([scores.iloc[8],scores.iloc[9],scores.iloc[10],
    ↪ scores.iloc[11],scores.iloc[12],scores.iloc[13],
                    scores.iloc[14],scores.iloc[15]]
g=ridge.plot()
```

Model	R^2 Test	R^2 Test
Ridge Alpha=0.5	0.7	0.69
Ridge Alpha=1	0.7	0.69
Ridge Alpha=10	0.7	0.685
Ridge Alpha=20	0.7	0.681
Ridge Alpha=50	0.69	0.67
Ridge Alpha=100	0.67	0.652
Ridge Alpha=1000	0.44	0.432

Table 3: Results obtained for Ridge Alpha data table

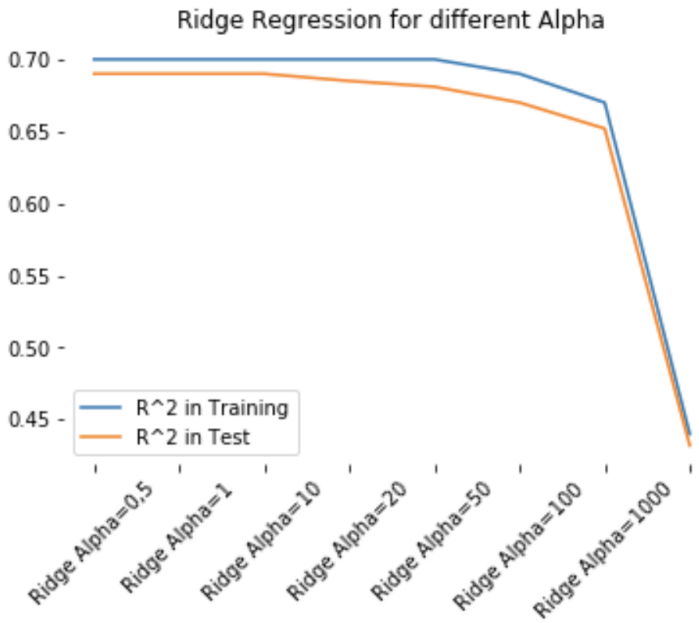

Figure 11: Plot of the score of Ridge's regression with different alpha parameters

6.3.4 Lasso Alpha Parameter Tuning

```
lasso=pd.DataFrame([scores.iloc[17],
scores.iloc[18],
scores.iloc[19],scores.iloc[20],
scores.iloc[21],scores.iloc[22],
scores.iloc[23]])

g=lasso.plot()
```

Model	R^2 Test	R^2 Test
Lasso Alpha=10	0.7	0.69
Lasso Alpha=100	0.7	0.69
Lasso Alpha=500	0.69	0.68
Lasso Alpha=1000	0.69	0.68
Lasso Alpha=5000	0.61	0.61
Lasso Alpha=10000	0.48	0.48

Table 4: Results obtained for Lasso Alpha data table

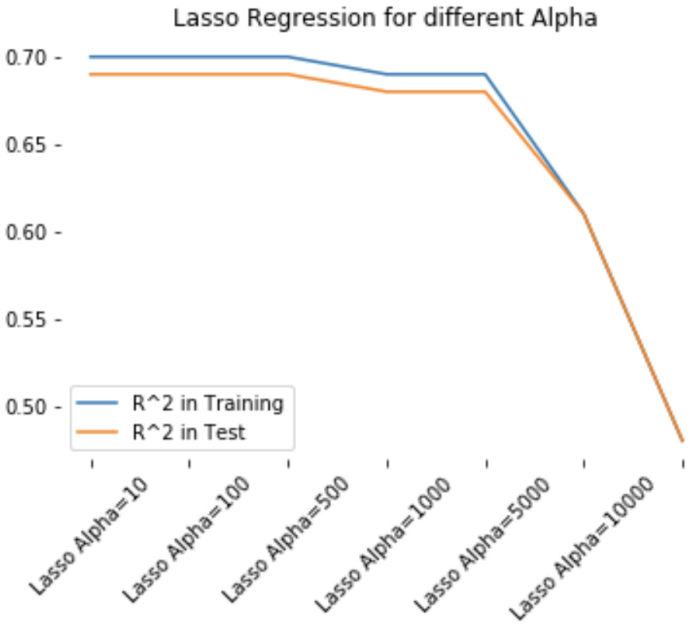

Figure 12: Plot of the score of Lasso's regression with different alpha parameters

6.3.5 k-Nearest Neighbors

We now start loading the model from the sklearn library to study k-Nearest Neighbors of our data set.

```
from sklearn.neighbors import KNeighborsClassifier
```

We have trained 4 different models (classifier one feature, classifier all feature, regression one feature, regression all features)

	R^2 Test	R^2 Test		
Class - One Feature	0.04	0.005		
Class - All Features	0.18	0.01		
Regr - One Feature	0.45	0.458		
Regr - All features	0.410	0.357		

Table 5: Results obtained for k-Nearest Neighbors

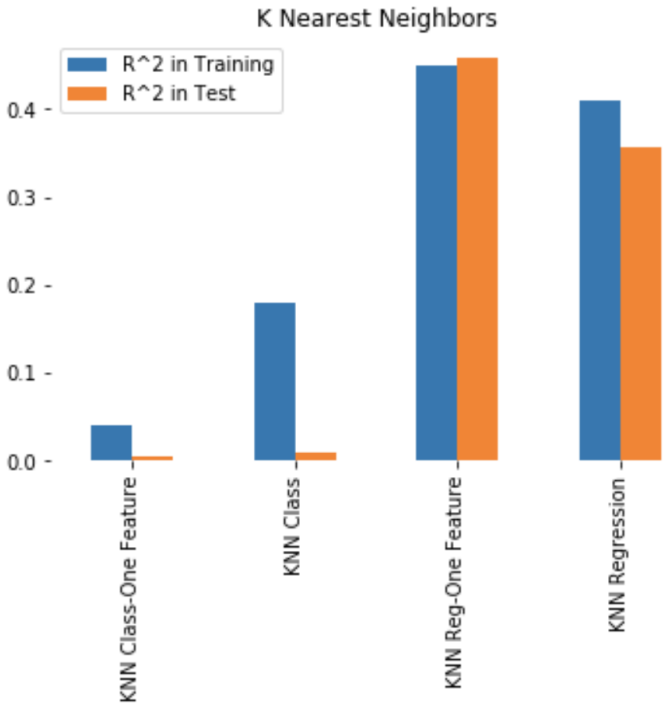

Figure 13: A bar representation of the scores of the different k-NN models

7 Discussion and Conclusions

7.1 Linear, Multiple and Polynomial Regression

We have applied the algorithm to predict, instead of classify, we have obtain $R^2 = 0.537$ (all features) and 0.462 for just one feature ($sqft\ living$).

We have applying the polynomial regressions of degree 2 and 3, we can see that the performance of the model is better with degree = 2 than degree = 3.

Applying the k-NN class, k-NN regression, k-NN regression one feature, linear, linear ridge, linear lasso, linear one feature, linear ridge one feature, square regression, square with ridge, cubic regression and cubic regression with ridge, we can see that the best model is linear regression with all features.

Using regression implementation, the (predicted price) estimated price for a house with 3500 square-feet is 939739.86. Also, with the inverse regression estimate, the estimated square-feet for a house worth 5500000.00 is 19673 square-feet. This provides satisfactory results.

7.2 Ridge Regression

We have executed the following 3 different models:

- **Ridge Linear Model:** all features $R^2 = 0.537$
- **Ridge Linear Model:** all features scaled $R^2 = -2.6$
- **Ridge Linear Model:** all features with different alphas, we obtain the best model with an alpha parameterization between 0.5-1 $R^2 = 0.540$

The result shows that the Ridge regression doesn't work much better than linear regression for this case.

In the implementation of the ridge regression using the gradient, we have predicted the house price for the **1st** house in the test set using the no regularization and high regularization models.

- **The Predicted 1st house (No Regularization):** 387465.47605823533
- **The Predicted 1st house (High Regularization):** 270453.53032194055
- **Actual house price:** 310000.0

We have also predicted the house price for the **11th** house in the test set using the no regularization and high regularization models.

- **The Predicted 11th house (No Regularization):** 478551.3475892041
- **The Predicted 11th house (High Regularization):** 315939.5228610413
- **The Actual house price:** 349000.0

As we study figure 14 & 15, it shows a very similar learned weights for both no regularization and high regularization using the RSS on the test data.

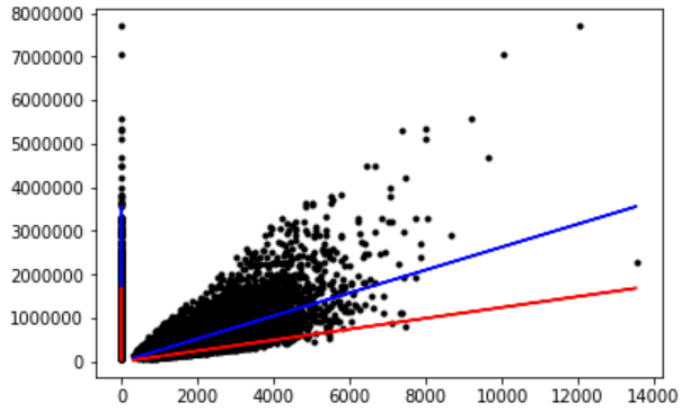

Figure 14: Graphical representation of the ridge regression using gradient descent implementation with 1 feature where blue is for no regularization and red is for high regularization

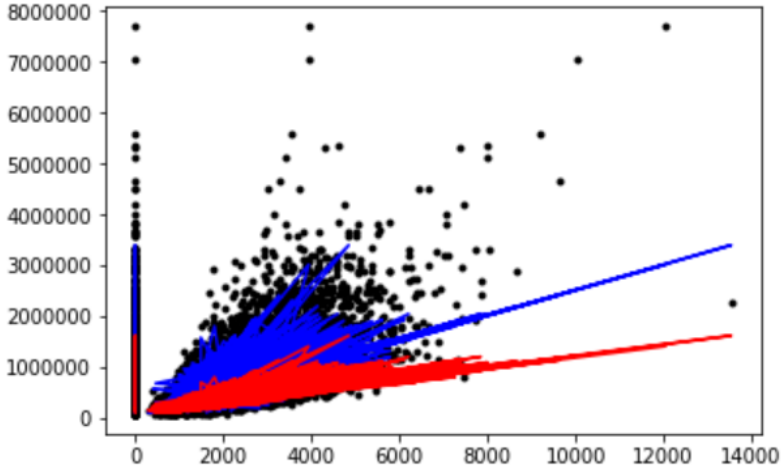

Figure 15: Graphical representation of the ridge regression using gradient descent implementation with 2 features where blue is for no regularization and red is for high regularization

7.3 Lasso

We have obtained results for Lasso (regression) of $R^2 = 0.69$. And the best alpha parameter is between $1 - 500$. This shows that the low alphas the results are better. For instance, when the alpha value is 0, Lasso (regression) produces the same coefficients as a linear regression. When alpha is very large, all coefficients are zero.

So, the Lasso (regression) is not working better than the linear regressions.

7.4 k-Nearest Neighbor

We have used the k-NN algorithm to predict, instead of classifying. This process obtained $R^2 = 0.36$ (all features) and 0.458 for just one feature (sqft-living). It works better with just one feature. We have calculated almost 50 percent of predicting prices with just one variable in k-NN regression.

And after studying the graphical representation, we can see that the best k-NN neighbors parameter is $k = 3$ with we obtain a $R^2 = 0.536$.

7.5 Conclusion

Within this report, we have conducted an empirical study to give on overview of regression methods using the given data set.

We have studied the Simple Regression, Multiple, Polynomial and Ridge Regression, Gradient Descent, Lasso, Coordinate Descent and k-Nearest Neighbors algorithms and models.

There is many more algorithms and/or models to used i.e. Elastic Net, Kernel Regression, Bayesian Regression, Support Vector Machine (using classification and/or regression analysis), Artificial Neural Network, Convolutional Neural Network, Random Forest, Decision Tree etc.

Based on the study of the python implementations developed, it is recommended to continue our experiments for auxiliary future progressive exploration and/or investigations.

However, at this moment, we are satisfied with our overall results to have achieved $R^2 = 0.700$ from our test data set without applying any extremely difficult and/or complex Artificial Intelligence algorithm.

References

[1] T. Hastie, R. Tibshirani and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. Springer-Verlag New York, 2009.

[2] V. K. Ayyadevara. *Pro Machine Learning Algorithms: A Hands-On Approach to Implementing Algorithms in Python and R*. Apress, 1st edition, 2018.

[3] G. James et al. *An Introduction to Statistical Learning : with Applications in R*. New York :Springer, 2013.

[4] J. Williams. Machine Learning: Regression - Implementation License QMG2VNAPF56C. 9/16/2018. URL: `https://www.coursera.org/account/accomplishments/verify/QMG2VNAPF56C`.

[5] J. Williams. Machine Learning Specialization: Implementation License 2BAL5JR9R5DS. 9/23/2018. URL: `https://www.coursera.org/account/accomplishments/specialization/2BAL5JR9R5DS`.

[6] M. Swamynathan. *Mastering Machine Learning with Python in Six Steps: A Practical Implementation Guide to Predictive Data Analytics Using Python*. Apress, 1st edition, 2017.

[7] P.-N. Tan, M. Steinbach and V. Kumar. *Introduction to Data Mining*. New York: Addison-Vesley, 1st edition, 2004.

[8] T. Mitchell. *Machine Learning*. New York: Mc Graw-Hill, 1997.

[9] R. Garreta and G. Moncecchi. *Learning scikit-learn: Machine Learning in Python*. Packt Publishing Limited, 2013.

[10] P. K. Janert. *Data Analysis with Open Source Tools: A Hands-On Guide for Programmers and Data Scientists*. O'Reilly Media, 1st edition, 2010.