Table of Contents

1

5. Predicting numeric outcomes with Linear Regression

Linear Regression is used to predict a continuous numeric value from a set of input features. This machine learning algorithm is fundamental to statisticians when it comes to predicting numeric outcomes. Although advanced algorithms such as neural networks and deep learning has taken the place of linear regression in modern times, this algorithm is key when it comes to providing you with the foundations of neural networks and deep learning.

The key benefit of building machine learning models with the linear regression algorithm opposed to neural networks/deep learning is that it is highly interpretable. Interpretability helps you, as the machine learning practitioner, understand how the different input variables behave when it comes to predicting the output.

The linear regression algorithm finds application in the financial industry in order to predict stock prices and the real estate industry in order to predict housing prices. In fact, the linear regression algorithm can be applied in any field where we need to predict a numeric value given a set of input features!

In this chapter you will learn about:

- The inner mechanics of the linear regression algorithm.
- Building and evaluating your first linear regression algorithm using scikit-learn.
- Scaling your data for a potential performance improvement.
- Optimizing your linear regression model

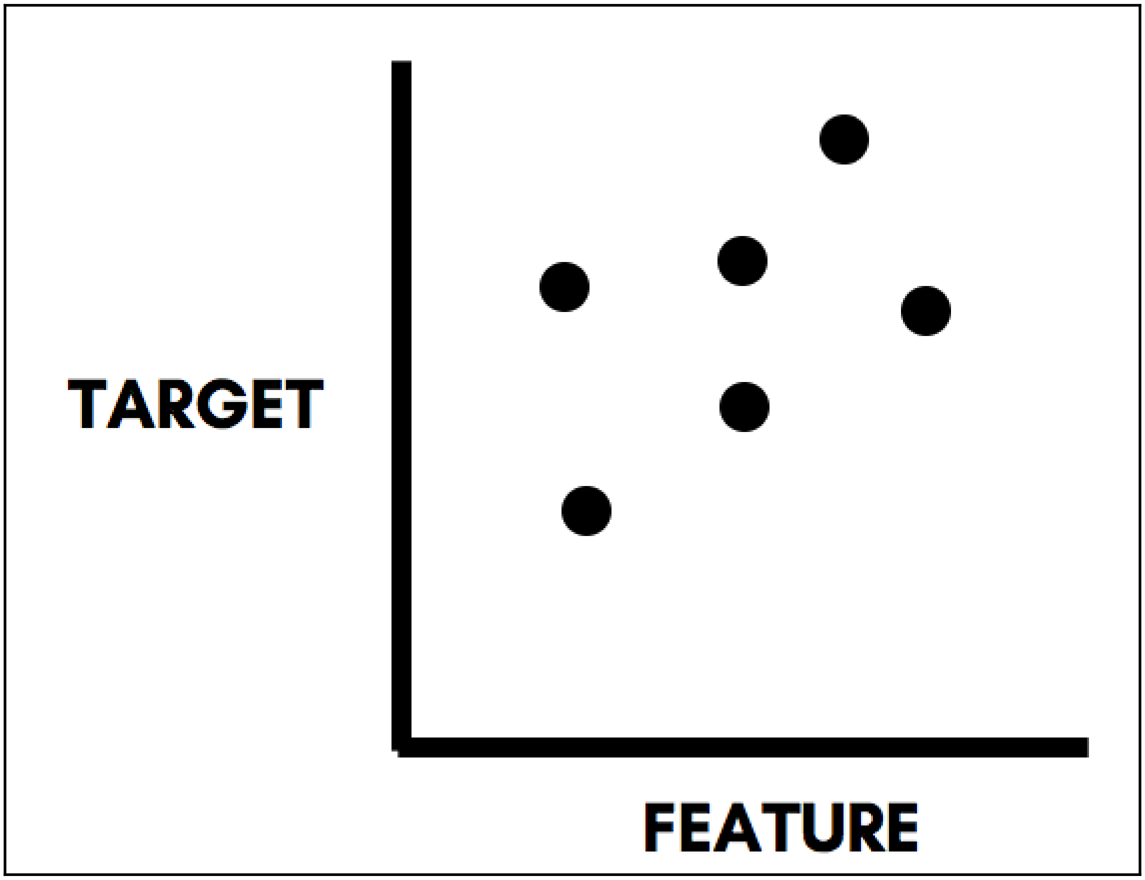
The inner mechanics of the linear regression algorithm

In it's most fundamental form, the expression for the linear regression algorithm can be written as:

$$NumericOutput = \sum(InputFeatures \times Parameter1) + Parameter2$$

In the equation above, the output of the model is a numeric outcome. In order to obtain this numeric outcome we require each input feature to be multiplied with a parameter called 'Parameter1' and we add the second parameter called 'Parameter2' to this result.

So in other words, our task is to find the values of the two parameters that can predict the value of the numeric outcome as accurately as possible. In visual terms, consider the image shown below:

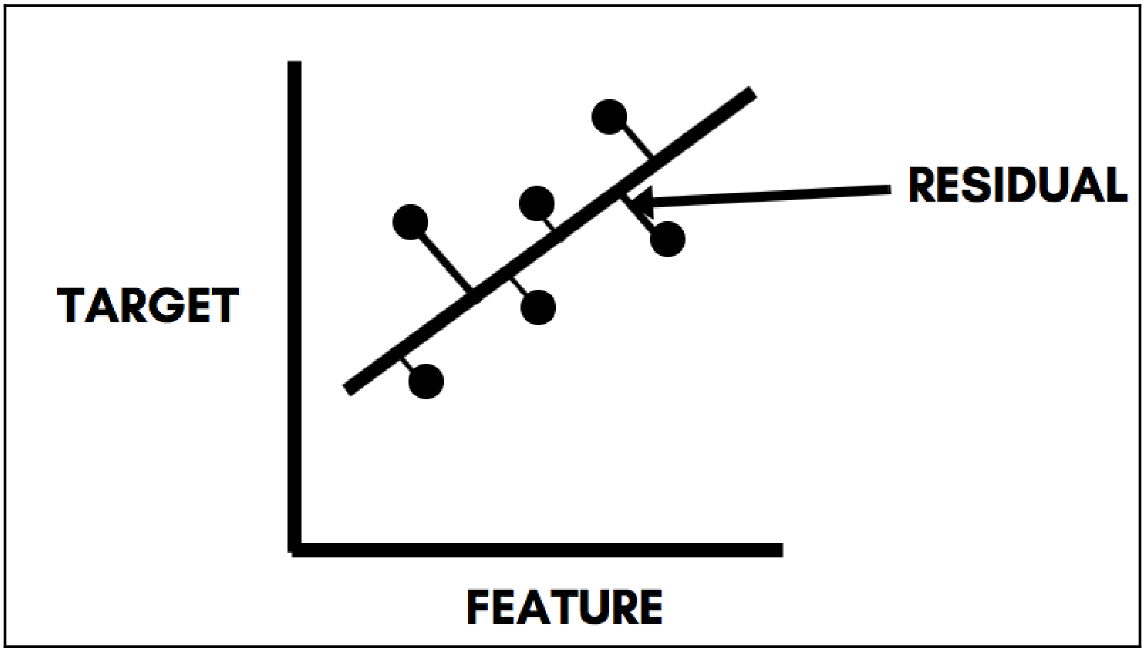


2-D plot between the target and input feature

The plot illustrated above represents a two dimensional plot between the target that we want to predict on the y-axis and the input feature along the x-axis. The goal of linear regression is to find the most optimal values of the two parameters mentioned in the equation above in order to fit a line through the given set of points.

This line is known as the line of best fit. A line of best fit is one that fits the given sets of points very well so that it can make accurate predictions for us. Therefore, in order to find the optimal values of the parameters that will result in the line of best fit we need to define a function that can do this for us.

This function is known as the **loss function**. The goal of the loss function, as the name suggests is to minimize the loss/error as much as possible so that we can obtain a line of best fit. In order to understand how this works visually, consider the image shown below:



Line of best fit

In the figure above, the line is fit through the set of data points. The distance between each point in the plot and the line is known as the residual. Therefore, the loss/error function in the sum of the squares of these residuals and the goal of the linear regression algorithm is to minimize this value. The sum of the squares of the residuals is known as **Ordinary Least Squares** or OLS.

Implementing Linear Regression in scikit-learn

In this section, you will implement your first linear regression algorithm in scikit-learn. To make this easy to follow, this section will be divided into three sub-sections under which you will learn about:

- Implementing and visualizing a simple linear regression model in 2-dimensions
- Implementing linear regression to predict the mobile transaction amount
- Scaling your data for a potential increase in performance

Linear Regression in 2-dimensions

In this sub-section you will learn how to implement your first linear regression algorithm in order to predict the amount of a mobile transaction by using one input feature - the old balance amount of the account holder. We will be using the same fraudulent mobile transaction dataset that we used in chapter-2 of this book for this chapter as well.

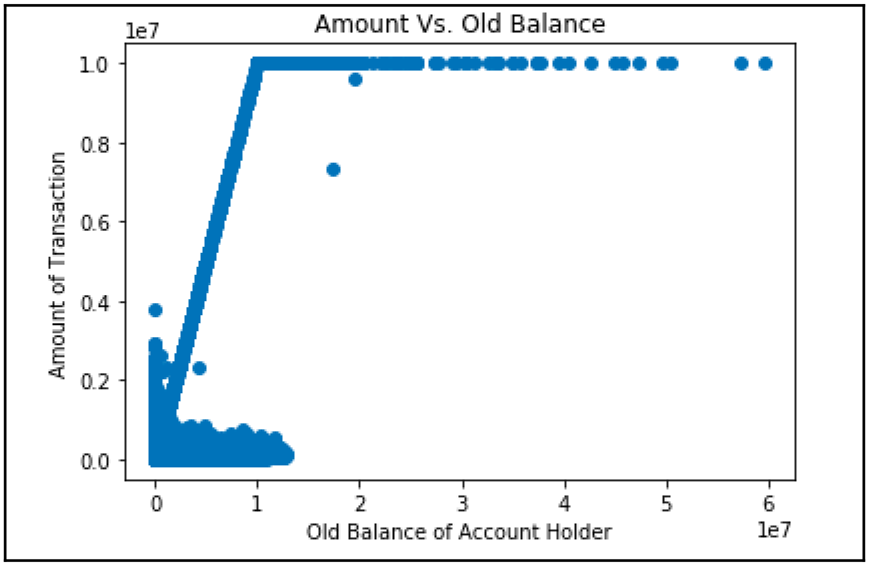
The first step is to read in the dataset and define the feature and target variable. This can be done by using the code shown below:

```
import pandas as pd

#Reading in the dataset

df = pd.read_csv('fraud_prediction.csv')

#Define the feature and target arrays

feature = df['oldbalanceOrg'].values
target = df['amount'].values
```

Next, we will create a simple scatter plot between the amount of the mobile transaction on the y-axis which is the outcome of the linear regression model and the old balance of the account holder along the x-axis which is the input feature. This can be done by using the code shown below:

```
import matplotlib.pyplot as plt

#Creating a scatter plot

plt.scatter(feature, target)
plt.xlabel('Old Balance of Account Holder')
plt.ylabel('Amount of Transaction')
plt.title('Amount Vs. Old Balance')
plt.show()
```

In the code above we use the *plt.scatter()* function in order to create a scatter plot between the *feature* on the x-axis and the *target* on the y-axis. This results in a scatter plot as illustrated in the figure below:



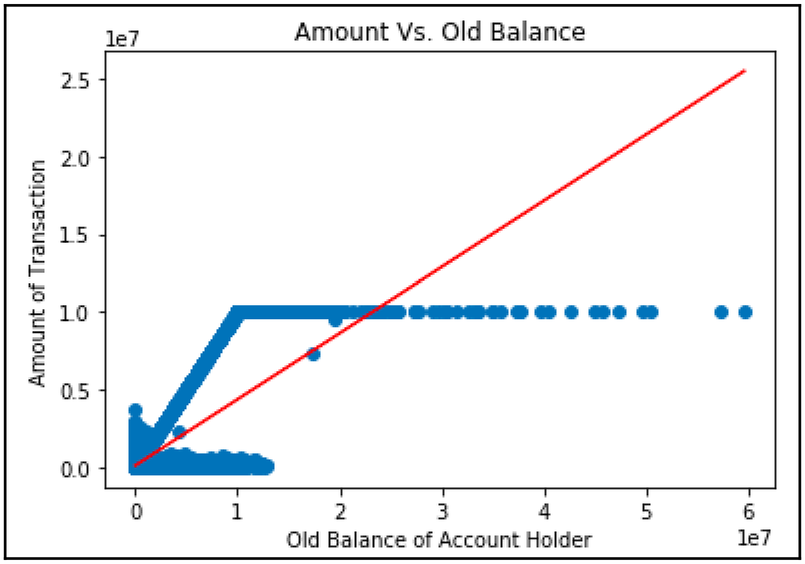
2-dimensional space of linear regression model

Now, we are going to fit a linear regression model in the 2-dimensional space illustrated in the image above. In order to do this we use the code shown below:

```
#Initializing a linear regression model

linear_reg = linear_model.LinearRegression()

#Reshaping the array since we only have a single feature

feature = feature.reshape(-1, 1)
target = target.reshape(-1, 1)

#Fitting the model on the data

linear_reg.fit(feature, target)
```

```
#Define the limits of the x-axis

x_lim = np.linspace(min(feature), max(feature)).reshape(-1, 1)

#Creating the scatter plot

plt.scatter(feature, target)
plt.xlabel('Old Balance of Account Holder')
plt.ylabel('Amount of Transaction')
plt.title('Amount Vs. Old Balance')

#Creating the prediction line

plt.plot(x_lim, linear_reg.predict(x_lim), color = 'red')

#Show the plot

plt.show()
```

This results in a line of best fit as illustrated in the image below:



Line of best fit

In the code above, we first initialize a linear regression model and fit the training data into this model. Since we only have a single feature, we need to reshape the feature and target for scikit-learn. Next, we define the upper and lower limits of the x-axis which contains our feature variable.

Finally, we create a scatter plot between the feature and the target variable and include the line of best fit with the color red as indicated in the image above.

Linear Regression to predict mobile transaction amount

Now that we have visualized how a simple linear regression model works in two dimensions, we can use the linear regression algorithm in order to predict the total amount of a mobile transaction using all the other features in our mobile transaction dataset.

The first step is to import our fraud prediction dataset into our workspace and divide it into training and test sets. This can be done by using the code shown below:

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Reading in the dataset

df = pd.read_csv('fraud_prediction.csv')

#Creating the features

features = df.drop('isFraud', axis = 1).values
target = df['isFraud'].values

X_train, X_test, y_train, y_test = train_test_split(features, target,
test_size = 0.3, random_state = 42, stratify = target)
```

We can now fit the linear regression model and evaluate the initial accuracy score of the model by using the code shown below:

```
from sklearn import linear_model

#Initializing a linear regression model

linear_reg = linear_model.LinearRegression()

#Fitting the model on the data

linear_reg.fit(X_train, y_train)

#Accuracy of the model

linear_reg.score(X_test, y_test)
```

In the code above, we first initialize a linear regression model which we then fit into the training data using the *.fit()* function and evaluate the accuracy score on the test data by using the *.score()* function. This resulted in an accuracy score of 98%, which is fantastic!

Scaling your data

Scaling your data and providing a level of standardization is a vital step in any linear regression pipeline as it could offer a potential way to enhance the performance of your model. In order to this, we use the code shown below:

```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

#Setting up the scaling pipeline

pipeline_order = [('scaler', StandardScaler()), ('linear_reg',
linear_model.LinearRegression())]

pipeline = Pipeline(pipeline_order)

#Fitting the classfier to the scaled dataset

linear_reg_scaled = pipeline.fit(X_train, y_train)

#Extracting the score

linear_reg_scaled.score(X_test, y_test)
```

We use the same scaling pipeline that we have used in all the previous chapters. In the code above, we replace the model name with the linear regression model and evaluate the scaled accuracy scores on the test data.

In this case, scaling the data did not lead to any improvement in accuracy score but it is vital to implement scaling into your linear regression pipeline as in most cases it does lead to an improvement in the accuracy scores.

Model optimization

The fundamental objective of the linear regression algorithm is to minimize the loss/cost function. In order to do this, the algorithm tries to optimize the values of the coefficients of each feature (*Parameter1)* such that the loss function is low.

Sometimes, this leads to overfitting as the coefficients of each variable are optimized for just the data that it is trained on. This means that your linear regression model will not generalize well beyond your current training data.

The process by which we penalize hyper-optimized coefficients in order to prevent this type of overfitting is called - **Regularization**.

There are two broad types of regularization methods:

1. Ridge Regression
2. Lasso Regression

In the sub-sections below, the two types of regularization techniques will be discussed in detail and you will learn about how you can implement them into your model.

Ridge Regression

The equation for the ridge regression is illustrated below:

$$RidgeLossFunction = OLSFunction + (Alpha \times \sum Parameter1^2)$$

In the equation above, the ridge loss function is equal to the Ordinary Least Squares loss function plus the product of the square of the coefficients of each feature and alpha.

Alpha is a parameter that we can optimize in order to control the amount by which the ridge loss function penalizes the coefficients in order to prevent overfitting. Obviously, if alpha is equal to 0 the ridge loss function is equal to the ordinary least squares loss function thereby making no difference to the initial overfit model.

Therefore, optimizing this value of alpha provides the most optimal model that generalizes well beyond the data that it has trained on.

In order to implement ridge regression into the fraud prediction dataset we use the code shown below:

```
from sklearn.linear_model import Ridge
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge

# Reading in the dataset
```

```
df = pd.read_csv('fraud_prediction.csv')

#Creating the features

features = df.drop('isFraud', axis = 1).values
target = df['isFraud'].values

X_train, X_test, y_train, y_test = train_test_split(features, target,
test_size = 0.3, random_state = 42, stratify = target)

#Initialize a ridge regression model

ridge_reg = Ridge(alpha = 0, normalize = True)

#Fit the model to the training data

ridge_reg.fit(X_train, y_train)

#Extract the score from the test data

ridge_reg.score(X_test, y_test)
```

In the code above, we first read in the dataset and divide it into training and test sets as usual. Next, we initialize a ridge regression model by using the *Ridge()* function with the parameters of alpha set to 0 and the normalize set to True in order to standardize the data.

Next, the ridge model is fit into the training data and the accuracy score is extracted from the test data. The accuracy of this model is exactly the same as the accuracy of the model that we built without the ridge regression as the parameter that controls how the model is optimized - alpha is set to zero.

In order to obtain the most optimal value of alpha using the GridSearchCV algorithm we use the code shown below:

```
from sklearn.model_selection import GridSearchCV

#Building the model

ridge_regression = Ridge()

#Using GridSearchCV to search for the best parameter

grid = GridSearchCV(ridge_regression, {'alpha':[0.0001, 0.001, 0.01, 0.1,
10]})
grid.fit(X_train, y_train)

# Print out the best parameter
```

```
print("The most optimal value of alpha is:", grid.best_params_)

#Initializing an ridge regression object

ridge_regression = Ridge(alpha = 0.01)

#Fitting the model to the training and test sets

ridge_regression.fit(X_train, y_train)

#Accuracy score of the ridge regression model

ridge_regression.score(X_test, y_test)
```

In the code above:

1. We first initialize a ridge regression model and then use the GridSearchCV algorithm to search for the most optimal value of alpha from a range of values.
2. After we obtain this optimal value of alpha we build a new ridge regression model with this optimal value on the training data and evaluate the accuracy score on the test data.

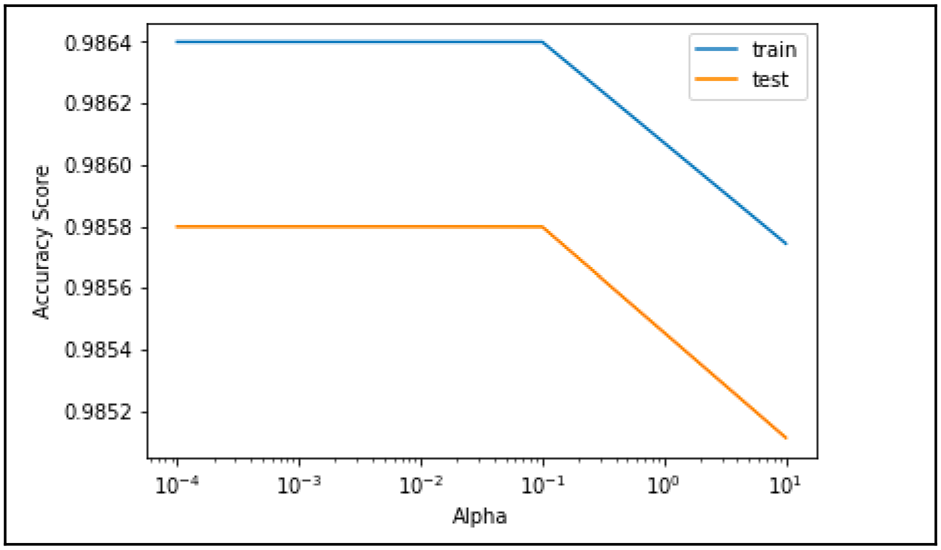
Since our initial model, was already well optimized, the accuracy score did not increase by an observable amount. However, on datasets with larger dimensions/features, ridge regression holds immense value in providing you with a model that generalizes well without overfitting.

In order to verify the results that the GridSearchCV algorithm has provided us with, we will construct a plot between the accuracy scores on the y-axis and the different values of alpha along the x-axis for both the training and test data. In order to do this, we use the code shown below:

```
import matplotlib.pyplot as plt

train_errors = []
test_errors = []

alpha_list = [0.0001, 0.001, 0.01, 0.1, 10]

# Evaluate the training and test classification errors for each value of
alpha

for value in alpha_list:
    # Create Ridge object and fit
    ridge_regression = Ridge(alpha= value)
    ridge_regression.fit(X_train, y_train)
```

```
        # Evaluate error rates and append to lists
        train_errors.append(ridge_regression.score(X_train, y_train) )
        test_errors.append(ridge_regression.score(X_test, y_test))
    # Plot results
    plt.semilogx(alpha_list, train_errors, alpha_list, test_errors)
    plt.legend(("train", "test"))
    plt.ylabel('Accuracy Score')
    plt.xlabel('Alpha')
    plt.show()
```

This results in an output as illustrated below:



Accuracy vs. Alpha

From the plot above, it is clear that a value of 0.01 does indeed provide the highest value of accuracy for both the training and test data and therefore the results from the GridSearchCV algorithm make logical sense.

In the code above, we first initialize two empty lists to store the accuracy scores for both the training and test data. We then evaluate the accuracy scores for both training and test sets for different values of alpha and create the plot which is displayed above.

Lasso Regression

The equation for the lasso regression is illustrated below:

$$LassoLossFunction = OLSFunction + (Alpha \times \sum |Parameter1|)$$

In the equation above, the lasso loss function is equal to the Ordinary Least Squares loss function plus the product of the absolute value of the coefficients of each feature and alpha.

Alpha is a parameter that we can optimize in order to control the amount by which the lasso loss function penalizes the coefficients in order to prevent overfitting. Once again, if alpha is equal to 0 the lasso loss function is equal to the ordinary least squares loss function thereby making no difference to the initial overfit model.

Therefore, optimizing this value of alpha provides the most optimal model that generalizes well beyond the data that it has trained on.

In order to implement the lasso regression into the fraud prediction dataset we use the code shown below:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Lasso
import warnings

# Reading in the dataset

df = pd.read_csv('fraud_prediction.csv')

#Creating the features

features = df.drop('isFraud', axis = 1).values
target = df['isFraud'].values

X_train, X_test, y_train, y_test = train_test_split(features, target,
test_size = 0.3, random_state = 42, stratify = target)

#Initialize a lasso regression model

lasso_reg = Lasso(alpha = 0, normalize = True)

#Fit the model to the training data

lasso_reg.fit(X_train, y_train)

warnings.filterwarnings('ignore')

#Extract the score from the test data
```

```
lasso_reg.score(X_test, y_test)
```

The code above is very similar to the code we used to build the ridge regression model as the only difference is the *Lasso()* function that we use in order to initialize a lasso regression model. Additionally, the warnings package is used in order to suppress the warning that is generated as we set the value of alpha to 0.

In order to optimize the value of alpha we use the GridSearchCV algorithm. This is done by using the code shown below:

```
from sklearn.model_selection import GridSearchCV

#Building the model

lasso_regression = Lasso()

#Using GridSearchCV to search for the best parameter

grid = GridSearchCV(lasso_regression, {'alpha':[0.0001, 0.001, 0.01, 0.1,
10]})
grid.fit(X_train, y_train)

# Print out the best parameter

print("The most optimal value of alpha is:", grid.best_params_)

#Initializing an lasso regression object

lasso_regression = Lasso(alpha = 0.0001)

#Fitting the model to the training and test sets

lasso_regression.fit(X_train, y_train)

#Accuracy score of the lasso regression model

lasso_regression.score(X_test, y_test)
```

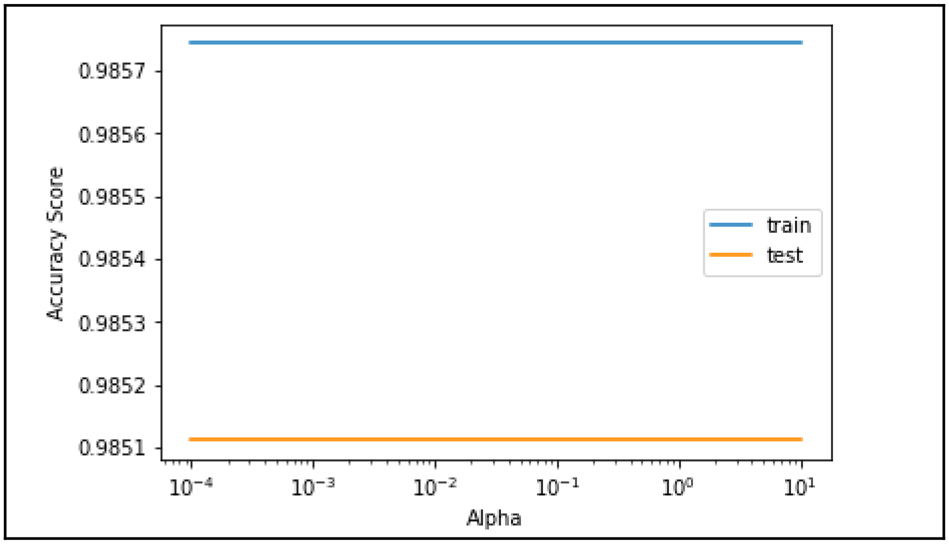
The code above is similar to the alpha optimization we implemented for the ridge regression. Here, we use the lasso regression model instead of the ridge regression model .

In order to verify the results of the GridSearchCV algorithm we construct a plot between the accuracy scores and the value of alpha for the training and test sets. This is illustrated in the code shown below:

```
train_errors = []
test_errors = []
```

```
alpha_list = [0.0001, 0.001, 0.01, 0.1, 10]

# Evaluate the training and test classification errors for each value of
alpha

for value in alpha_list:
    # Create Lasso object and fit
    lasso_regression = Lasso(alpha= value)
    lasso_regression.fit(X_train, y_train)
    # Evaluate error rates and append to lists
    train_errors.append(ridge_regression.score(X_train, y_train) )
    test_errors.append(ridge_regression.score(X_test, y_test))
# Plot results
plt.semilogx(alpha_list, train_errors, alpha_list, test_errors)
plt.legend(("train", "test"))
plt.ylabel('Accuracy Score')
plt.xlabel('Alpha')
plt.show()
```

This results in an output as illustrated below:



Accuracy vs. Alpha

All values of alpha provide the same value of accuracy scores and we can thus pick the value given to us by the GridSearchCV algorithm.

Ridge vs. Lasso

The key question in the process of implementing the ridge and lasso regression is when to use one of them when building your linear regression model. This depends on the dataset and the features under consideration.

Therefore, a good practice is to implement both the ridge and lasso regression and asses which type of regularization method provides the best performance.

Summary

In this chapter, you have learnt how the linear regression algorithm works internally by understanding key concepts such as residuals and ordinary least squares. You have also learnt how to visualize a simple linear regression model in two dimensions.

Implementing the linear regression model in order to predict the amount of a mobile transaction was also done along with scaling your data in an effective pipeline for a potential improvement in performance.

Finally, you have learnt how to optimize your model by using the concept of regularization in the form of ridge and lasso regression.

Index