Open in Colab

Deepcamp: Codelab 2

In this tutorial we will cover:

- Outliers and what to do with them

Author:

- Alessio Devoto (alessio.devoto@uniroma1.it)

Duration: 30 mins

Sales prediction

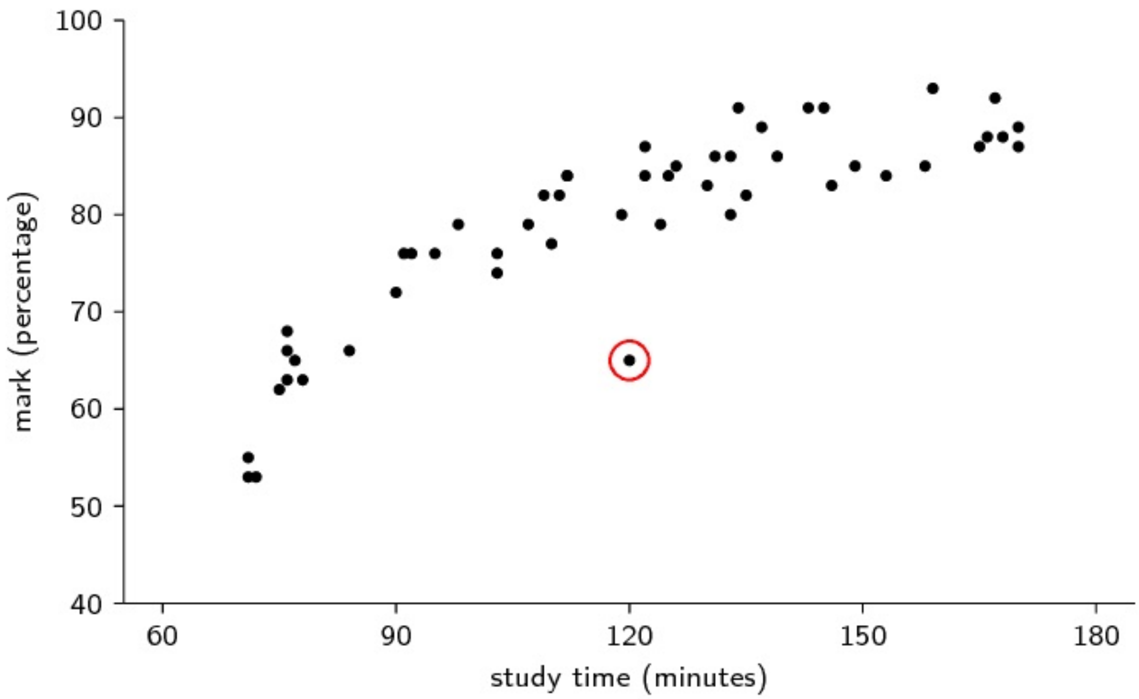
- You company collected a dataset containing information about investments in commercials via TV, Radio and Newspapers.
- Given the amount of money invested in different advertisement media (TV, Radio, Newspaper) predict sales.

In this notebook we are going to meet **outliers** 😦 for the first time!

What are we going to do? 🤔

1. Data import , analysis & preprocessing ⚙️
2. Train an ML model
3. Treat outliers
4. Check performance degradation due to outliers

But wait... what actually is an outlier?



From Wikipedia: *in statistics, an outlier is a data point that differs significantly from other observations.*

First, we import the necessary libraries as usual...

In [ ]:
```
!pip install pandas
!pip install scikit-learn
!pip install plotly
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab
-wheels/public/simple/
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-pack
ages (1.5.3)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/pyth
on3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dis
t-packages (from pandas) (2022.7.1)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/di
st-packages (from pandas) (1.22.4)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-pa
ckages (from python-dateutil>=2.8.1->pandas) (1.16.0)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab
-wheels/public/simple/
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dis
t-packages (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/di
st-packages (from scikit-learn) (1.22.4)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dis
t-packages (from scikit-learn) (1.10.1)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/di
st-packages (from scikit-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python
3.10/dist-packages (from scikit-learn) (3.1.0)
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab
-wheels/public/simple/
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-pack
ages (5.13.1)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/
dist-packages (from plotly) (8.2.2)
```

In [ ]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.linear_model import LinearRegression
```

... and download the data

In [ ]:

```
!wget https://raw.githubusercontent.com/alessiodevoto/deepers/main/data/Adve
```

```
--2023-05-12 16:03:28--  https://raw.githubusercontent.com/alessiodevoto/dee
pers/main/data/Advertising_outliers.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.1
10.133, 185.199.109.133, 185.199.108.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.
110.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 4019 (3.9K) [text/plain]
Saving to: 'Advertising_outliers.csv'

Advertising_outlier 100%[===================>]   3.92K  --.-KB/s    in 0s

2023-05-12 16:03:28 (37.9 MB/s) - 'Advertising_outliers.csv' saved [4019/401
9]
```

1. ⬇️ Data import & analysis

Aftern the first codelab we should be quite good at this 😀

In [ ]:
```python
# read dataframe from csv
sales_data = pd.read_csv('Advertising_outliers.csv')
```

In [ ]:
```python
sales_data.head()
```

Out[ ]:

	TV	Radio	Newspaper	Sales
0	517.0	37.8	69.2	22.1
1	616.0	39.3	45.1	10.4
2	668.0	45.9	69.3	9.3
3	775.0	41.3	58.5	18.5
4	658.0	10.8	58.4	12.9

In [ ]:
```python
sales_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   TV         200 non-null    float64
 1   Radio      200 non-null    float64
 2   Newspaper  200 non-null    float64
 3   Sales      200 non-null    float64
dtypes: float64(4)
memory usage: 6.4 KB
```

Let us explore the correlation between each of the three features and the target.

In [ ]:
```python
px.scatter(sales_data, x=['TV', 'Radio', 'Newspaper'], y='Sales')
```

It looks like we have a quite clear ***linear correlation*** between the money spent for TV advertisment and the sales, if not for all those awful points on the right hand side.

Our goal: use the info amount of money invested in TV commercial to predict the revenues (i.e. we don't care about Radio and Newspapers for now)

Let's have a look at the distribution of each indipendent variable via a boxplot.

In [ ]:
```
fig = px.box(sales_data, y=["TV", "Radio", 'Newspaper'])
fig.show()
```

Bad news: we were already suspecting it but it is clear now, there are some ***outliers*** in the TV feature: exactly the one we had chosen for our model! 😞

Well, let's try and train our model, maybe we'll get reasonable performances despite the outliers!

2. Train an ML model

Before we start: keep in mind we are training our model on **dirty data** which will probably affect the model's prediction.

2.1 Linear Regression

Linear regression is a simple method that looks for the line that best suits the data by minimizing the mean squared error.

In [ ]:
```
# do the usual train test split

sales_data_feat, sales_data_labels = sales_data.drop(columns='Sales'), sales
```

```
X_train, X_test, y_train, y_test = train_test_split(sales_data_feat, sales_c
```

We can use any of the three columns, but in this case we only keep the TV as we saw there is a strong linear correlation

In [ ]:
```
# this way we can pick which columns we should use
use_columns = ['TV']

# fit the model
lr = LinearRegression().fit(X=X_train[use_columns].values, y=y_train.values)
```

Let's see what is the mean error we are getting ...

In [ ]:
```
print('Score:', lr.score(X_test[use_columns].values, y_test))
y_pred = lr.predict(X_test[use_columns].values)
print("mean_absolute_error:    ", metrics.mean_absolute_error(y_test, y_prec
print("mean_squared_error:    ",metrics.mean_squared_error(y_test, y_pred))
print("squared mean_squared_error:   ",np.sqrt(metrics.mean_squared_error(y_
```

```
Score: 0.07856899406694329
mean_absolute_error:     4.084282010494524
mean_squared_error:     24.141894099364674
squared mean_squared_error:    4.913440149158701
```

Seems to be quite bad 😢 , let's try and visualize what's happening.

The result of linear regression is just a line in an N dimesional space, where N=number of features!

We can retrieve the line's equation and plot it!

In [ ]:
```
print('Coefficients: ', lr.coef_)
print('Intercept: ', lr.intercept_)

px.line(x=sales_data['TV'], y=sales_data['TV'] * lr.coef_ + lr.intercept_)
```

```
Coefficients:  [0.01470421]
Intercept:  11.51163455961833
```

Well, that's just a line... Now we can visualize how well the line fits the data.

```
In [ ]: trace1= go.Scatter(mode='markers', x=sales_data['TV'], y=sales_data['Sales']
        trace2 = go.Scatter(x=sales_data['TV'], y=sales_data['TV'] * lr.coef_ + lr.i

        fig = make_subplots(specs=[[{"secondary_y": True}]])
        fig.add_trace(trace1)
        fig.add_trace(trace2,secondary_y=True)
        fig['layout'].update(height = 600, width = 800,xaxis=dict(tickangle=-90))
        fig.show()
```

Exercise: 🏋️ Decision Tree

Perform the same task with a decision tree for regression (`sklearn.tree.DecisionTreeRegressor()`) and plot the results

In [ ]:
```
from sklearn import tree
tree_reg =
```

In [ ]:
```
#@title Peek solution
from sklearn import tree
# this way we can pick which features we should use
use_columns = ['TV']

# fit the model
tree_reg = tree.DecisionTreeRegressor().fit(X=X_train[use_columns].values, y
```

In [ ]:
```
print('Score:', tree_reg.score(X_test[use_columns].values, y_test))
y_pred = tree_reg.predict(X_test[use_columns].values)
print("mean_absolute_error:    ", metrics.mean_absolute_error(y_test, y_pred
print("mean_squared_error:    ",metrics.mean_squared_error(y_test, y_pred))
print("squared mean_squared_error:   ",np.sqrt(metrics.mean_squared_error(y_
```

```
Score: 0.12405083640592862
mean_absolute_error:     3.495
mean_squared_error:     22.950249999999997
squared mean_squared_error:    4.79064191940913
```

In [ ]:
```
y_pred = tree_reg.predict(X_test[use_columns].values)
df = pd.DataFrame({'x':X_test[use_columns].values.squeeze(), 'y': y_pred}).s
# px.line(df, x='x', y='y')
```

```
trace1= go.Scatter(mode='markers', x=sales_data['TV'], y=sales_data['Sales']
trace2 = go.Scatter(x=df['x'], y=df['y'])

fig = make_subplots(specs=[[{"secondary_y": True}]])
fig.add_trace(trace1)
fig.add_trace(trace2,secondary_y=True)
fig['layout'].update(height = 600, width = 800,xaxis=dict(tickangle=-90))
fig.show()
```

3. Treat outliers & retrain

We can assume outliers are the major responsible for our bad results.

In order to handle the outliers we should first:

1. Find how many outliers we have and where they have
2. Decide what to do with them

There are a few methods to handle outliers: z-score, Quartiles ...

Z-Score

Let's write a simple function to find out which samples are out of distribution.

We first compute the distribution's mean and standard deviation, and then we check how many 'standard deviations' each point is from the mean.

$\zeta(x) = \frac{(x-\mu)}{\sigma}$

where μ is the mean and σ is the std deviation.

We can then drop points which are too far from the mean, i.e.

```
if Z_score > threshold:
  drop sample
```

In [ ]:
```python
# a simple function to detect outliers
def Zscore_outlier(df: pd.DataFrame, threshold):
    out=[]
    m = np.mean(df) # mean
    sd = np.std(df) # std deviation

    # iterate over the dataset
    for idx, i in enumerate(df):
        z = (i-m)/sd
        if np.abs(z) > threshold:
            out.append(idx)
    print("Outliers:", out)
    return out

# apply function to TV column and get indexes of outliers
outliers_idx = Zscore_outlier(sales_data['TV'], threshold=2.5)
```

```
Outliers: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [ ]:
```python
# let's just drop the outliers and retrain

sales_data = sales_data.drop(index=outliers_idx)
```

In [ ]:
```python
sales_data_feat, sales_data_labels = sales_data.drop(columns='Sales'), sales

X_train, X_test, y_train, y_test = train_test_split(sales_data_feat, sales_d
```

In [ ]:
```python
use_columns = ['TV']

# fit the model
lr = LinearRegression().fit(X=X_train[use_columns].values, y=y_train.values)
```

In [ ]:
```python
print('Score:', lr.score(X_test[use_columns].values, y_test))
y_pred = lr.predict(X_test[use_columns].values)
```

```
print("mean_absolute_error:    ", metrics.mean_absolute_error(y_test, y_prec
print("mean_squared_error:    ",metrics.mean_squared_error(y_test, y_pred))
print("squared mean_squared_error:   ",np.sqrt(metrics.mean_squared_error(y_
```

```
Score: 0.589997273643232
mean_absolute_error:     2.841946224777112
mean_squared_error:     13.012417039604875
squared mean_squared_error:    3.6072727980574015
```

Looks like we got a quite good increase in score! 🍾

---

1. 🚫 Simply ***dropping the outliers*** is not a smart way to deal with them. In the vast majority of cases, you should conduct a deeper analysis of the outliers distribution, maybe ask a *domain expert* or replace the outliers with another value.!

2. 🚫 Some ML methods, like Linear Regression, are more sensitive to outliers. Other methods, like decision trees, are more robust. The choice of the ML algorithm you use will affect the robustness of your model!

In [ ]:
```
trace1= go.Scatter(mode='markers', x=sales_data['TV'], y=sales_data['Sales']
trace2 = go.Scatter(x=sales_data['TV'], y=sales_data['TV'] * lr.coef_ + lr.i

fig = make_subplots(specs=[[{"secondary_y": True}]])
fig.add_trace(trace1)
fig.add_trace(trace2,secondary_y=True)
fig['layout'].update(height = 600, width = 800,xaxis=dict(
      tickangle=-90
    ))
fig.show()
```

Final Exercises 🔥

We probably won't have time but you can do this at home just for fun 😀

1. Perform linear regression on the dirty dataset with all the three columns

In [ ]:
```
# your code here
```

In [ ]:
```
#@title Peek solution 👀

use_columns = ['TV', 'Radio', 'Newspaper']

# fit the model
lr = LinearRegression().fit(X=X_train[ue_columns].values, y=y_train.values)
```

```
print('Score:', lr.score(X_test[use_columns].values, y_test))
y_pred = lr.predict(X_test[use_columns].values)
print("mean_absolute_error:    ", metrics.mean_absolute_error(y_test, y_pred
print("mean_squared_error:    ",metrics.mean_squared_error(y_test, y_pred))
print("squared mean_squared_error:   ",np.sqrt(metrics.mean_squared_error(y_
```

```
Score: 0.843567025979476
mean_absolute_error:     1.1421037681213264
mean_squared_error:     3.3529669837143548
squared mean_squared_error:    1.831110860574628
```

2. Use SVC instead of linear regression

Hint: the model is called `sklearn.svm.SVR`.

In [ ]:
```
from sklearn import svm

# this way we can pick which features we should use
use_columns = ['TV']

# fit the model
svr = svm.SVR().fit(X=X_train[use_columns].values, y=y_train.values)
```

In [ ]:
```
print('Score:', svr.score(X_test[use_columns].values, y_test))
y_pred = svr.predict(X_test[use_columns].values)
print("mean_absolute_error:    ", metrics.mean_absolute_error(y_test, y_pred
print("mean_squared_error:    ",metrics.mean_squared_error(y_test, y_pred))
print("squared mean_squared_error:   ",np.sqrt(metrics.mean_squared_error(y_
```

```
Score: 0.5638541333971796
mean_absolute_error:     2.43187847158925
mean_squared_error:     9.34830204411301
squared mean_squared_error:    3.0574993122015597
```

In [ ]:
```
# let's plot and see what it looks like

y_svr = svr.predict(X_test[use_columns].values)
df = pd.DataFrame({'x':X_test[use_columns].values.squeeze(), 'y': y_svr}).sc


trace1= go.Scatter(mode='markers', x=sales_data['TV'], y=sales_data['Sales']
trace2 = go.Scatter(x=df['x'], y=df['y'])

fig = make_subplots(specs=[[{"secondary_y": True}]])
fig.add_trace(trace1)
fig.add_trace(trace2,secondary_y=True)
fig['layout'].update(height = 600, width = 800,xaxis=dict(tickangle=-90))
fig.show()
```