regression-from-scratch-using-ols

July 17, 2023

1 Problem Statement

Implement linear regression from scratch using Ordinary Least Squares (OLS) method and compare it with linear regression from sklearn

2 Importing libraries

[1]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

from sklearn.metrics import r2_score
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A
NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy
(detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

3 Dataset Description

[2]:
```
dataset = pd.read_csv("/kaggle/input/calcofi/bottle.csv")
```

```
/tmp/ipykernel_20/2368306870.py:1: DtypeWarning: Columns (47,73) have mixed
types. Specify dtype option on import or set low_memory=False.
  dataset = pd.read_csv("/kaggle/input/calcofi/bottle.csv")
```

[3]:
```
dataset.shape
```

[3]:
```
(864863, 74)
```

[4]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 864863 entries, 0 to 864862
Data columns (total 74 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   Cst_Cnt              864863 non-null  int64
 1   Btl_Cnt              864863 non-null  int64
 2   Sta_ID               864863 non-null  object
 3   Depth_ID             864863 non-null  object
 4   Depthm               864863 non-null  int64
 5   T_degC               853900 non-null  float64
 6   Salnty               817509 non-null  float64
 7   O2ml_L               696201 non-null  float64
 8   STheta               812174 non-null  float64
 9   O2Sat                661274 non-null  float64
 10  Oxy_µmol/Kg          661268 non-null  float64
 11  BtlNum               118667 non-null  float64
 12  RecInd               864863 non-null  int64
 13  T_prec               853900 non-null  float64
 14  T_qual               23127 non-null   float64
 15  S_prec               817509 non-null  float64
 16  S_qual               74914 non-null   float64
 17  P_qual               673755 non-null  float64
 18  O_qual               184676 non-null  float64
 19  SThtaq               65823 non-null   float64
 20  O2Satq               217797 non-null  float64
 21  ChlorA               225272 non-null  float64
 22  Chlqua               639166 non-null  float64
 23  Phaeop               225271 non-null  float64
 24  Phaqua               639170 non-null  float64
 25  PO4uM                413317 non-null  float64
 26  PO4q                 451786 non-null  float64
 27  SiO3uM               354091 non-null  float64
 28  SiO3qu               510866 non-null  float64
 29  NO2uM                337576 non-null  float64
 30  NO2q                 529474 non-null  float64
 31  NO3uM                337403 non-null  float64
 32  NO3q                 529933 non-null  float64
 33  NH3uM                64962 non-null   float64
 34  NH3q                 808299 non-null  float64
 35  C14As1               14432 non-null   float64
 36  C14A1p               12760 non-null   float64
 37  C14A1q               848605 non-null  float64
 38  C14As2               14414 non-null   float64
 39  C14A2p               12742 non-null   float64
 40  C14A2q               848623 non-null  float64
 41  DarkAs               22649 non-null   float64
 42  DarkAp               20457 non-null   float64
```

```
 43  DarkAq                840440 non-null  float64
 44  MeanAs                22650 non-null   float64
 45  MeanAp                20457 non-null   float64
 46  MeanAq                840439 non-null  float64
 47  IncTim                14437 non-null   object
 48  LightP                18651 non-null   float64
 49  R_Depth               864863 non-null  float64
 50  R_TEMP                853900 non-null  float64
 51  R_POTEMP              818816 non-null  float64
 52  R_SALINITY            817509 non-null  float64
 53  R_SIGMA               812007 non-null  float64
 54  R_SVA                 812092 non-null  float64
 55  R_DYNHT               818206 non-null  float64
 56  R_O2                  696201 non-null  float64
 57  R_O2Sat               666448 non-null  float64
 58  R_SIO3                354099 non-null  float64
 59  R_PO4                 413325 non-null  float64
 60  R_NO3                 337411 non-null  float64
 61  R_NO2                 337584 non-null  float64
 62  R_NH4                 64982 non-null   float64
 63  R_CHLA                225276 non-null  float64
 64  R_PHAEO               225275 non-null  float64
 65  R_PRES                864863 non-null  int64
 66  R_SAMP                122006 non-null  float64
 67  DIC1                  1999 non-null    float64
 68  DIC2                  224 non-null     float64
 69  TA1                   2084 non-null    float64
 70  TA2                   234 non-null     float64
 71  pH2                   10 non-null      float64
 72  pH1                   84 non-null      float64
 73  DIC Quality Comment   55 non-null      object
dtypes: float64(65), int64(5), object(4)
memory usage: 488.3+ MB
```

[5]:
```
#choosing only temperature and salinity cols
df = dataset[['T_degC', 'Salnty']]
```

[6]:
```
df.head()
```

[6]:

	T_degC	Salnty
0	10.50	33.440
1	10.46	33.440
2	10.46	33.437
3	10.45	33.420
4	10.45	33.421

3.0.1 Dropping null values

[7]:
```
df.isna().sum()
```

[7]:
```
T_degC    10963
Salnty    47354
dtype: int64
```

[8]:
```
df.dropna(axis = 0, inplace = True)
```

```
/tmp/ipykernel_20/165812464.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df.dropna(axis = 0, inplace = True)
```

[9]:
```
df.shape
```

[9]:
```
(814247, 2)
```

[10]:
```
#taking only first 1000 rows
df = df.sample(1000)
```

[11]:
```
df.shape
```

[11]:
```
(1000, 2)
```

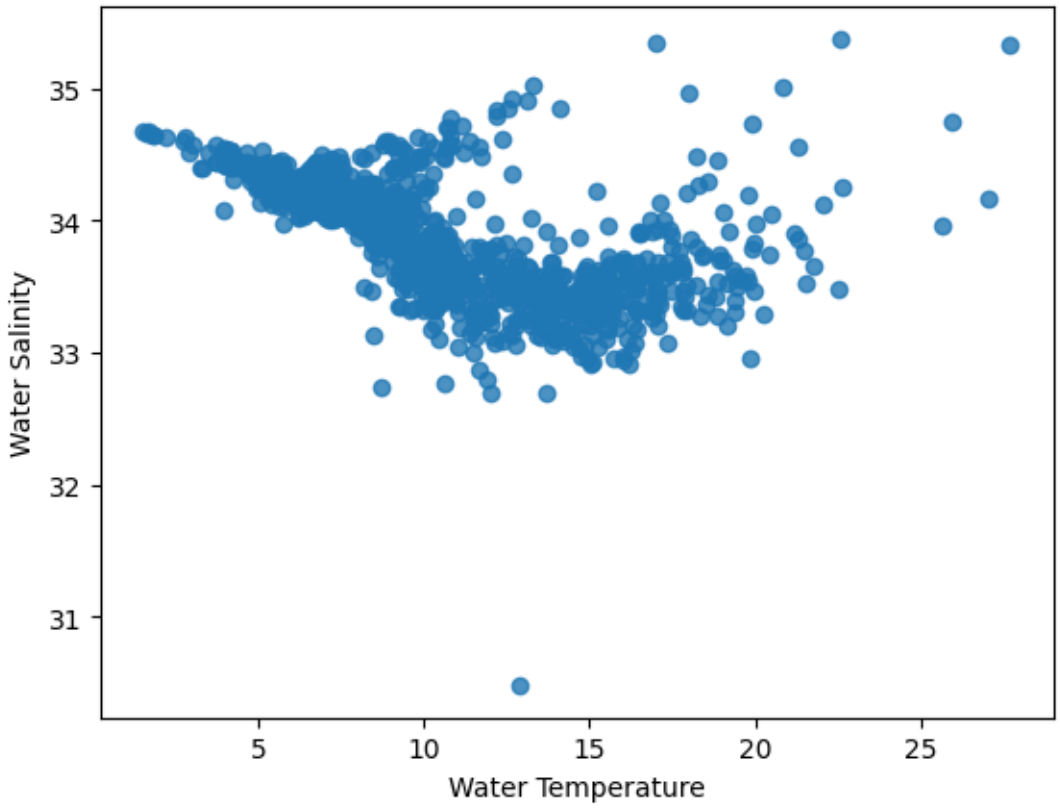
[12]:
```
df.describe()
```

[12]:

	T_degC	Salnty
count	1000.000000	1000.000000
mean	10.892001	33.853134
std	4.183828	0.454413
min	1.560000	30.480000
25%	8.000000	33.507425
50%	10.080000	33.856500
75%	13.830000	34.203950
max	27.680000	35.377000

3.0.2 Linear Relationship

[13]:
```
plt.scatter(x = df['T_degC'], y = df['Salnty'], alpha = 0.8)
plt.xlabel("Water Temperature")
plt.ylabel("Water Salinity")
plt.show()
#with temperature salinity decreases
```



4 Train-Test Split

[14]:
```
X = df['T_degC']
y = df['Salnty']
```

[15]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,␣
↪random_state = 10, shuffle = True)
```

[16]:
```
X_train.shape
```

[16]: (700,)

5 Linear Regression From Sklearn

[17]:
```
regressor = LinearRegression()
regressor.fit(np.array(X_train).reshape(-1, 1), np.array(y_train).reshape(-1,␣
↪1))
```

[17]: LinearRegression()

[18]:
```
m = regressor.coef_
m
```

[18]:
```
array([[-0.05802125]])
```

[19]:
```
b = regressor.intercept_
b
```

[19]:
```
array([34.4925945])
```

[20]:
```
y_pred = regressor.predict(np.array(X_test).reshape(-1,1))
```

5.0.1 R2 score

[21]:
```
r2_score(y_test, y_pred)
```

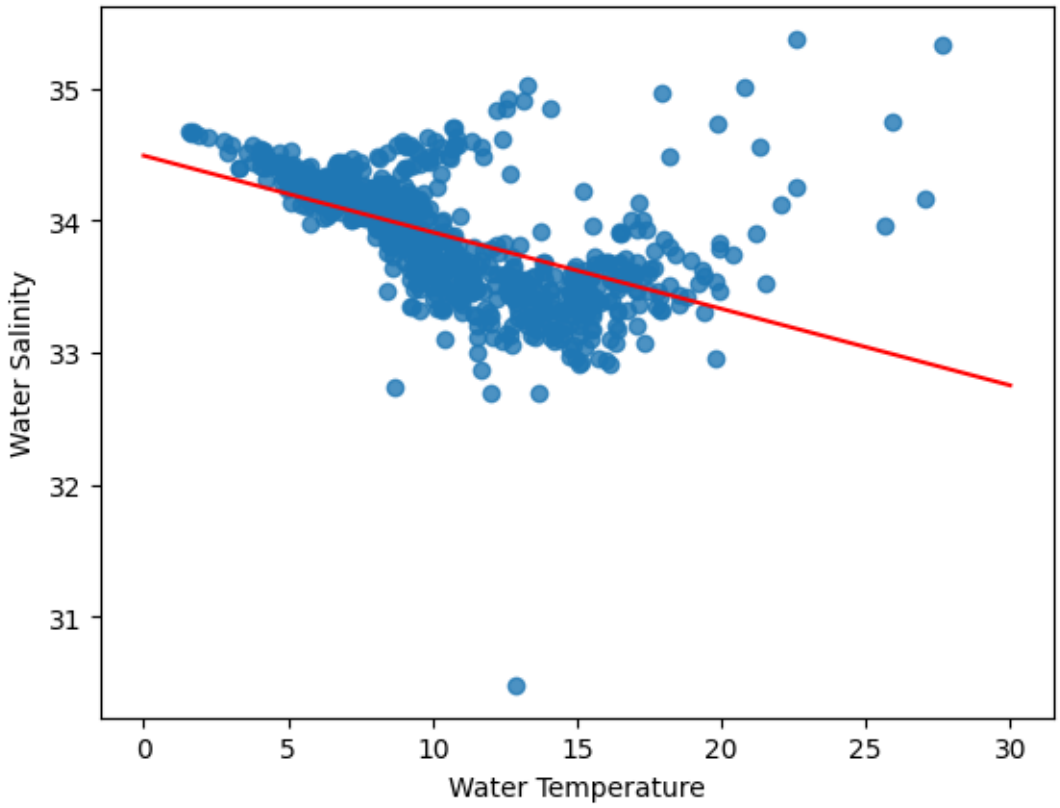
[21]:
```
0.2850305795165128
```

5.0.2 Best fit line

[22]:
```
x_input = np.linspace(0, 30, 500)
y_input = m * x_input + b
```

[23]:
```
y_input = y_input.reshape(-1, 1)
```

[24]:
```
plt.scatter(x = X_train, y = y_train, alpha = 0.8)
plt.plot(x_input, y_input, c = 'r')
plt.xlabel('Water Temperature')
plt.ylabel('Water Salinity')
plt.show()
```



6 Linear Regression From Scratch

[25]:
```
#m - slope, b = y intercept
class LinearRegressionFromScratch:

    def __init__(self):
        self.m = None
        self.b = None

    def fit(self, X_train, y_train):
        num = 0 #numerator
        den = 0 #denominator

        X_mean = X_train.mean()
        y_mean = y_train.mean()

        for i in range(X_train.shape[0]):
            num = num + ((X_train.iloc[i] - X_mean) * (y_train.iloc[i] -␣
 ↪y_mean))
```

```
            den = den + ((X_train.iloc[i] - X_mean) * (X_train.iloc[i] -␣
↪X_mean))

        self.m = num/den
        self.b = y_mean - (self.m * X_mean)
        return self.m, self.b

    def predict(self, X_test):
        y_pred = []
        for i in range(X_test.shape[0]):
            y_pred.append(self.m * X_test.iloc[i] + self.b)
        return y_pred
```

[26]:
```
regressor = LinearRegressionFromScratch()
```

[27]:
```
m, b = regressor.fit(X_train, y_train)
```

[28]:
```
print(m)
```

```
-0.05802125246595463
```

[29]:
```
print(b)
```

```
34.492594500868606
```

[30]:
```
y_pred = regressor.predict(X_test)
```

6.0.1 R2 score

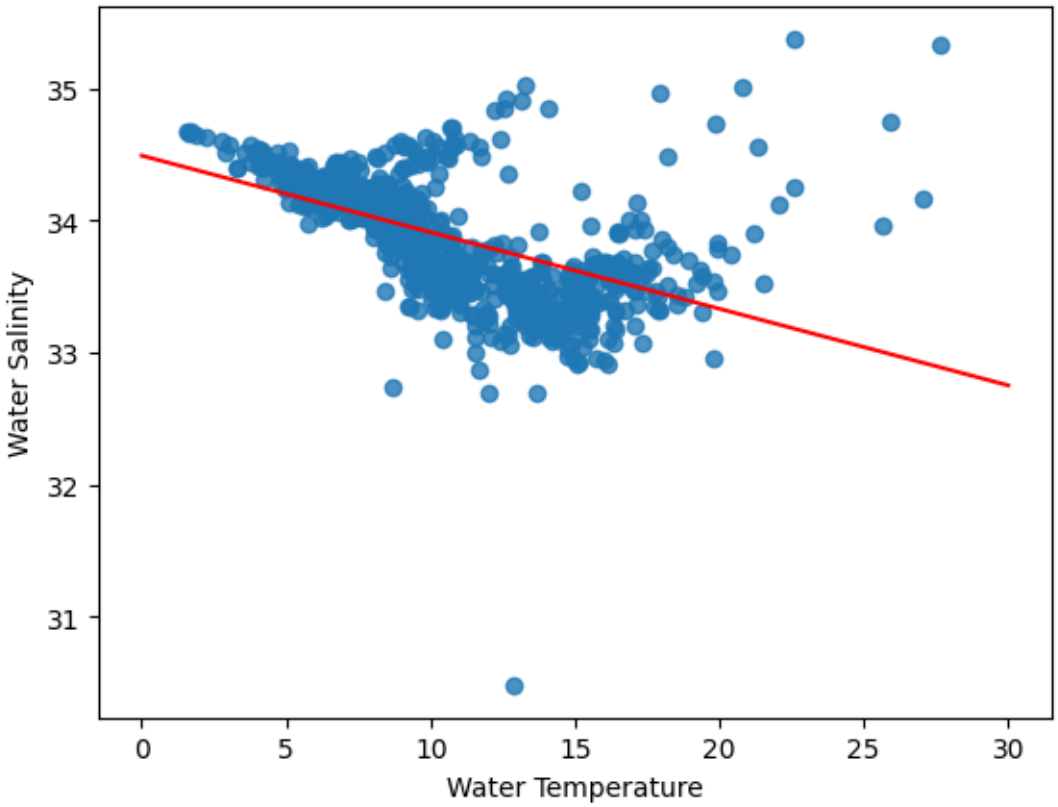
[31]:
```
r2_score(y_test, y_pred)
```

[31]:
```
0.28503057951651245
```

6.0.2 Best fit line

[32]:
```
x_input = np.linspace(0, 30, 500)
y_input = m * x_input + b
```

[33]:
```
plt.scatter(x = X_train, y = y_train, alpha = 0.8)
plt.plot(x_input, y_input, c = 'r')
plt.xlabel('Water Temperature')
plt.ylabel('Water Salinity')
plt.show()
```



7 Inference

Sklearn uses the exact OLS method which we implemented to perform linear regression.