Data Mining
Lab 1: Regression

Jonathan Aechtner

September 18, 2021

1 Assignment 1

1.1 (a) which answer is correct and why

from this selection option 3 is correct, *For a fixed value of IQ and GPA, males earn more on averagethan females provided that the GPA is high enough.* To prove this we can calculate 4 different cases.

$$\begin{aligned}\text{female and high GPA} &= -10 * 4 * 1 + 35 * 1 + 20 * 4 = 75\\ \text{male and high GPA} &= -10 * 4 * 0 + 35 * 0 + 20 * 4 = 80\\ \text{female and low GPA} &= -10 * 1 * 1 + 35 * 1 + 20 * 1 = 45\\ \text{male and low GPA} &= -10 * 1 * 0 + 35 * 0 + 20 * 1 = 20\end{aligned}$$

As we can see the case *male and high GPA* has a higher predicted outcome than *female and high GPA*. However this is only true for high GPAs as the case *male and low GPA* compared to *female and low GPA* shows. For this straight forward approach we ignore the terms $\beta_2 * X_2$ and $\beta_4 * X_1 * X_2$ as the β values are very small and the IQ is given as a fixed value. Hence the influence on the final result is supposedly very small.

From a more abstract point of view one could argue, that there are two main blocks to this equation. One decreases the outcome based on gender and GPA. The other increases the outcome based on Gender and GPA. If we think of these blocks as linear functions dependent on Gender and GPA there is an intersection of these two. Before that intersection females with the same GPA as males earn more. After this intersection the effect is reversed. Hence, *Males earn more for a fixed value for IQ and GPA, provided that the GPA is high enough.*

1.2 (b) Predict a specific salary

To predict the salary of a female with IQ 110 and GPA of 4 we simply insert all these values in the given formula and end up with the following equation.
y = predicted salary

$$\begin{aligned}y &= \beta_0 + \beta_1 * X_{GPA} + \beta_2 * X_{IQ} + \beta_3 * X_{Gender} + \beta_4 * X_{GPA} * X_{IQ} + \beta_5 * X_{GPA} * X_{Gender}\\ y &= 50 + 20 * 4 + 0.07 * 110 + 35 * 1 + 0.001 * 4 * 110 - 10 * 4 * 1\\ y &= 77.5\end{aligned}$$

2 Assignment 2

please find the code for assignment 2 in the following pages.

Lab1

September 18, 2021

1 Assignment 2

[ ]:
```
import numpy as np
```

(a) Using the random.normalfunction, create a vector, x, containing 100 observations drawn from a normal distribution with mean of 0.0 variance of 1. The vector xrepresents a feature X.

[ ]:
```
np.random.seed(42)
x = np.random.normal(loc=0, scale=1, size=(100,1))
x = np.sort(x, axis=0)
```

(b) Using the random.normalfunction, create a vector, eps, containing 100 observations drawn from a normal distribution with mean of 0.0 and variance of 0.25.

[ ]:
```
np.random.seed(42)
eps = np.random.normal(loc=0, scale=0.25, size=(100,1))
```

(c) Using xand eps, generate a vector y according to the model:Y= −1 + 0.5X+ . What is the length of the vector y? What are the values of 0 and 1 in this linear model?

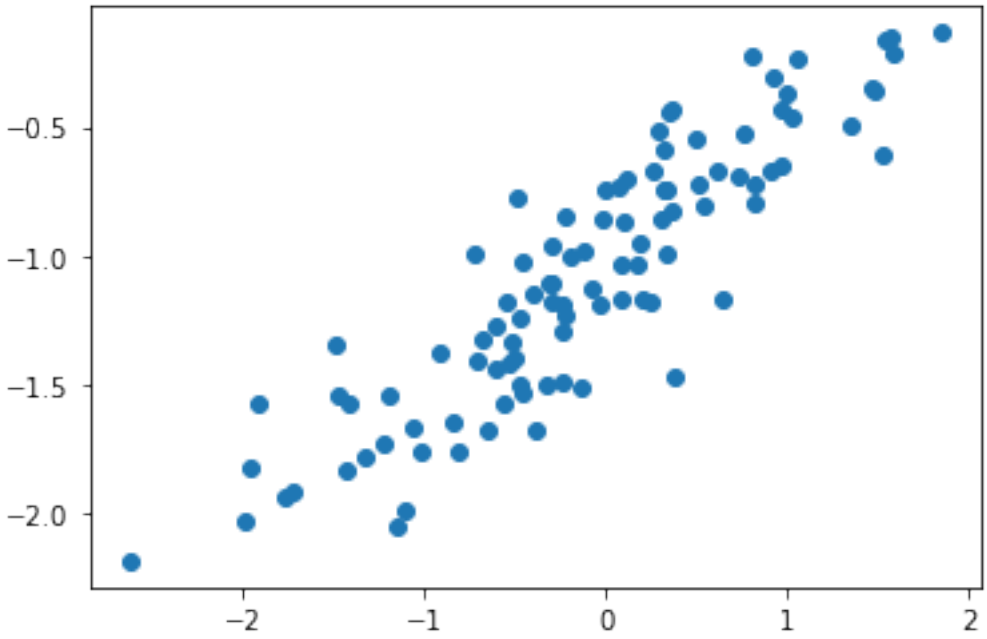
[ ]:
```
beta0 = -1
beta1 = 0.5
y = beta0 + beta1 * x + eps
print(f'Value for  0 = {beta0}')
print(f'Value for  1 = {beta1}')
print(f'Length of the vector y = {y.shape[0]}')
```

```
Value for  0 = -1
Value for  1 = 0.5
Length of the vector y = 100
```

(d) Create a scatterplot displaying the relationship between xand y. Comment on what you observe. For training regression models use the sklearnmodulelinear_model.

[ ]:
```
import matplotlib.pyplot as plt
plt.scatter(x,y)
```

[ ]:
```
<matplotlib.collections.PathCollection at 0x7fc58b3f1a00>
```



there is a very obvious correlation between x and y that seems to be linear.

(e) Fit a least squares linear model LinearRegression()from the module linear_model to predict y using x. Comment on the model obtained. How do the estimations of ˆ0 and ˆ1 compare to 0 and 1 themselves? Display the least squares line on the scatterplot obtained in (d).Compute R2statistics (using function r2_score from the sklearn.metricsmodule).

[ ]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

lin_model = LinearRegression()
lin_model.fit(x,y)
lin_model_coef = lin_model.coef_
lin_model_inter = lin_model.intercept_
print(f'the trained model has the following intercept: {lin_model_inter}')
print(f'the trained model has the following coefficients: {lin_model_coef}')

print(f'the difference between the original intercept and the model intercept␣
 ↪is {beta0-lin_model_inter}')
print(f'the difference between the original coeff beta1 and the model intercept␣
 ↪is {beta1-lin_model_coef}')
y_pred = lin_model.predict(x)
```

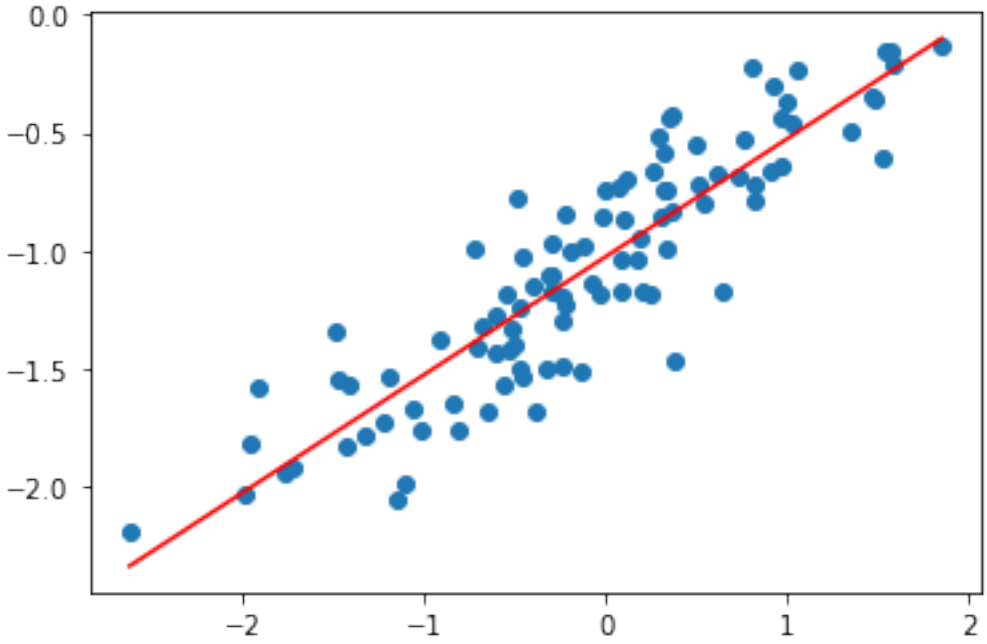
```
the trained model has the following intercept: [-1.02603934]
the trained model has the following coefficients: [[0.49925165]]
the difference between the original intercept and the model intercept is
[0.02603934]
```

```
the difference between the original coeff beta1 and the model intercept is
[[0.00074835]]
```

The model intercept (ˆ0) and the actual value for 0 are identical. However, there is a difference for the ˆ1 coefficient. T

[ ]:
```
plt.scatter(x,y)
plt.plot(x,y_pred, 'r-')
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x7fc58c7eee80>]
```



At least from a visual point of view the model seems to predict y very nicely for all the given x. This is hardly suprising, as we've seen the clear linear relationship between the original x and y. We compute the r2_score to see if the visual impression is supported by actual numbers.

[ ]:
```
print(f'The r2_score for the predictions obtained from the model is:␣
↪{r2_score(y, y_pred)}')
```

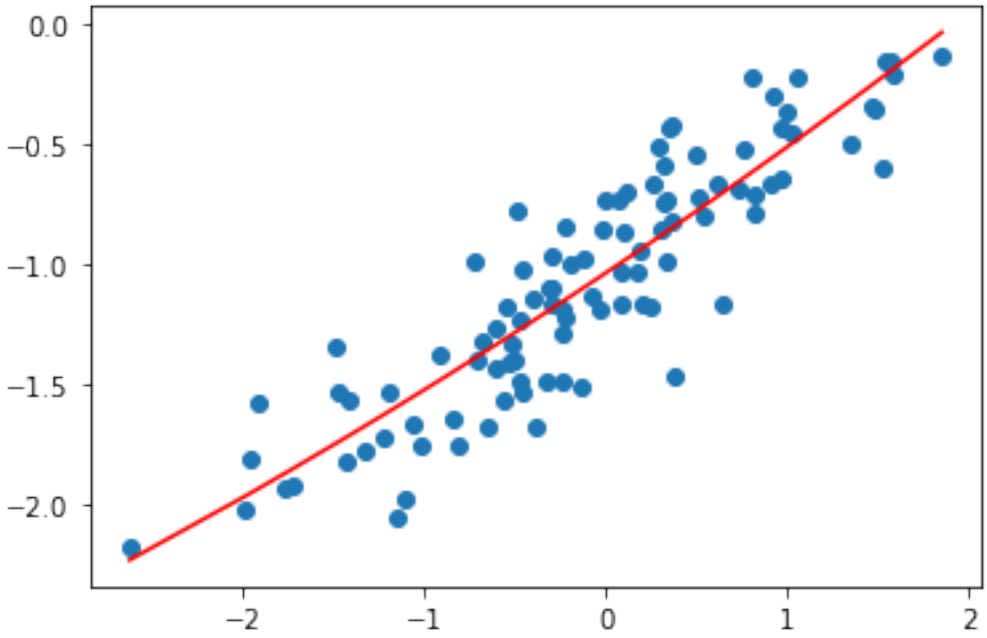
```
The r2_score for the predictions obtained from the model is: 0.7995217037124893
```

(f) Now fit a polynomial regression model that predicts yusing xand x2. Display the curve of polynomial regression model on thescatterplot obtained in (d). Is there evidence thatthe quadratic term improves themodel fit? Explain your answer.

[ ]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

x_poly = PolynomialFeatures().fit_transform(x)
```

```
poly_model = LinearRegression()
poly_model.fit(x_poly, y)
poly_mod_inter = poly_model.intercept_
poly_mod_coef = poly_model.coef_
print(f'the trained model has the following intercept: {poly_mod_inter}')
print(f'the trained model has the following coefficients: {poly_mod_coef}')

y_pred_poly = poly_model.predict(x_poly)

plt.scatter(x,y)
plt.plot(x, y_pred_poly, 'r-')
print(f'The r2_score for the predictions obtained from the model is:␣
 ↪{r2_score(y, y_pred_poly)}')
```

```
the trained model has the following intercept: [-1.04115401]
the trained model has the following coefficients: [[0.         0.50626173
0.01914967]]
The r2_score for the predictions obtained from the model is: 0.8012638693952027
```



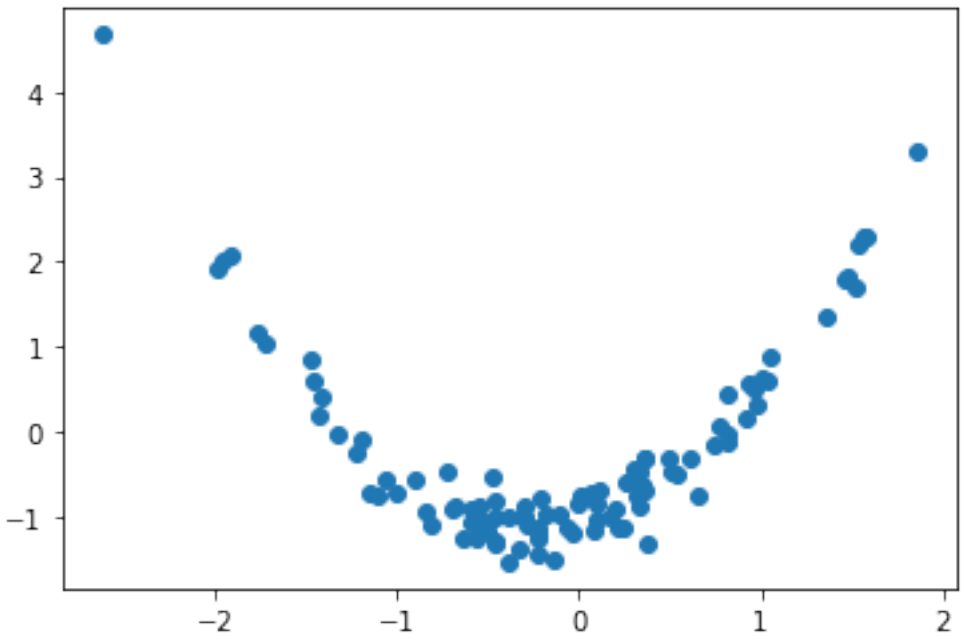
There is a slight positive change in the accuracy of the model. However, this difference is marginal, so I wouldn't say that this appproach improves the model whatsoever. Generally speaking it would be strange to see a big effect of a polynomial regression, given that the data was generated using a lineal model. Furthermore the increase we see here is most likely due to an decrease in variance that is associated with the introduction of further parameters. This on the other hand implies that the bias of the model increases as well, so it would fit less well to additional data points drawn from the same distribution.

(g) Repeat (a)–(f) after modifying the data generation process insuch a way that the vector y is according to the model: y= −1 + 0.5X+ X2 + .

[ ]:
```
beta0 = -1
beta1 = 0.5
y = beta0 + beta1 * x + np.square(x)+ eps
print(f'Value for  0 = {beta0}')
print(f'Value for  1 = {beta1}')
print(f'Length of the vector y = {y.shape[0]}')

import matplotlib.pyplot as plt
plt.scatter(x,y)
```

```
Value for  0 = -1
Value for  1 = 0.5
Length of the vector y = 100
```

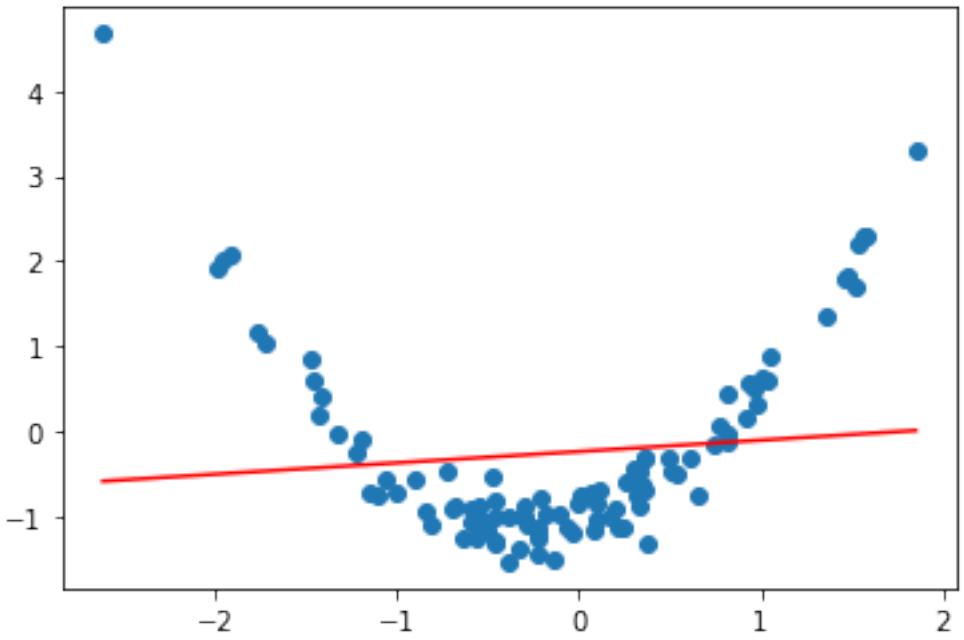
[ ]:
```
<matplotlib.collections.PathCollection at 0x7fc58c976c40>
```



[ ]:
```
lin_model = LinearRegression()
lin_model.fit(x,y)
lin_model_coef = lin_model.coef_
lin_model_inter = lin_model.intercept_
print(f'the trained model has the following intercept: {lin_model_inter}')
print(f'the trained model has the following coefficients: {lin_model_coef}')
```

```
print(f'the difference between the original intercept and the model intercept␣
 ↪is {beta0-lin_model_inter}')
print(f'the difference between the original coeff beta1 and the model intercept␣
 ↪is {beta1-lin_model_coef}')

y_pred = lin_model.predict(x)

plt.scatter(x,y)
plt.plot(x,y_pred, 'r-')
print(f'The r2_score for the predictions obtained from the model is:␣
 ↪{r2_score(y, y_pred)}')
```

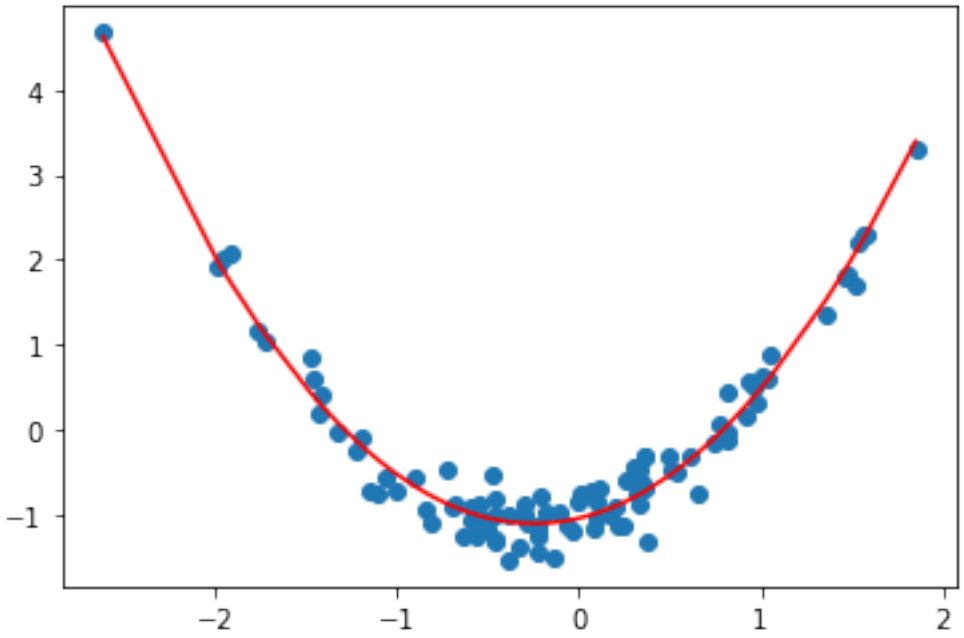
```
the trained model has the following intercept: [-0.23674791]
the trained model has the following coefficients: [[0.13318387]]
the difference between the original intercept and the model intercept is
[-0.76325209]
the difference between the original coeff beta1 and the model intercept is
[[0.36681613]]
The r2_score for the predictions obtained from the model is:
0.010962666331880144
```



[ ]:
```
x_poly = PolynomialFeatures().fit_transform(x)
poly_model = LinearRegression()
poly_model.fit(x_poly, y)
poly_mod_inter = poly_model.intercept_
poly_mod_coef = poly_model.coef_
```

```
print(f'the trained model has the following intercept: {poly_mod_inter}')
print(f'the trained model has the following coefficients: {poly_mod_coef}')

print(f'the difference between the original intercept and the model intercept␣
 ↪is {beta0-poly_mod_inter}')
print(f'the difference between the original coeff beta1 and the model intercept␣
 ↪is {beta1-poly_mod_coef}')

y_pred_poly = poly_model.predict(x_poly)

plt.scatter(x,y)
plt.plot(x, y_pred_poly, 'r-')
print(f'The r2_score for the predictions obtained from the model is:␣
 ↪{r2_score(y, y_pred_poly)}')
```

```
the trained model has the following intercept: [-1.04115401]
the trained model has the following coefficients: [[0.         0.50626173
1.01914967]]
the difference between the original intercept and the model intercept is
[0.04115401]
the difference between the original coeff beta1 and the model intercept is [[
0.5        -0.00626173 -0.51914967]]
The r2_score for the predictions obtained from the model is: 0.9617088393070675
```



In this case the polynomial regression perform performs significantly better than the linear regression. This is unsuprising as the model used to generate the data points is polynomial. A linear

model can't predict data of this type very well, because it has too few parameters to adequately capture the nature of the data distribution.