Assignment 2

Marion Rosec

Decembre 2023

Exercise 1

a)

Let $t = \begin{bmatrix} t_1 & \cdots & t_N \end{bmatrix}$

Let $W = \begin{bmatrix} w_0 \\ \cdots \\ w_N \end{bmatrix}$

Let $X = \begin{bmatrix} x_1 \\ \cdots \\ x_N \end{bmatrix}$ with $x_N = \begin{bmatrix} x_{1,N} & \cdots & x_{N,N} \end{bmatrix}$

Let $A = \begin{bmatrix} \alpha_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \alpha_N \end{bmatrix}$

The sum is then :

$$\begin{aligned} L &= \frac{1}{N}(WX - t)^T A(WX - t) \\ &= \frac{1}{N}(X^T W^T AXW - X^T W^T At - t^T AXW + t^T At) \\ &= \frac{1}{N}(X^T AXW^2 - 2X^T W^T At + t^T At) \end{aligned}$$

In order to compute the optimal parameters, we then need to solve this equation :

$$\frac{\partial L}{\partial W} = \frac{2}{N} X^T AXW - \frac{2}{N} X^T At + 0 = 0$$

$$X^T XW = X^T t \Leftrightarrow IW = (X^T AX)^{-1} X^T At$$

The optimal least square values are given by : $\bar{w} = (X^T AX)^{-1} X^T At$

b)

After modifying the linear regression class from assignment 1 to a weighted linear regression, we can use it on our testing set. We can then compare the plot observed with the one obtained in last assignment :

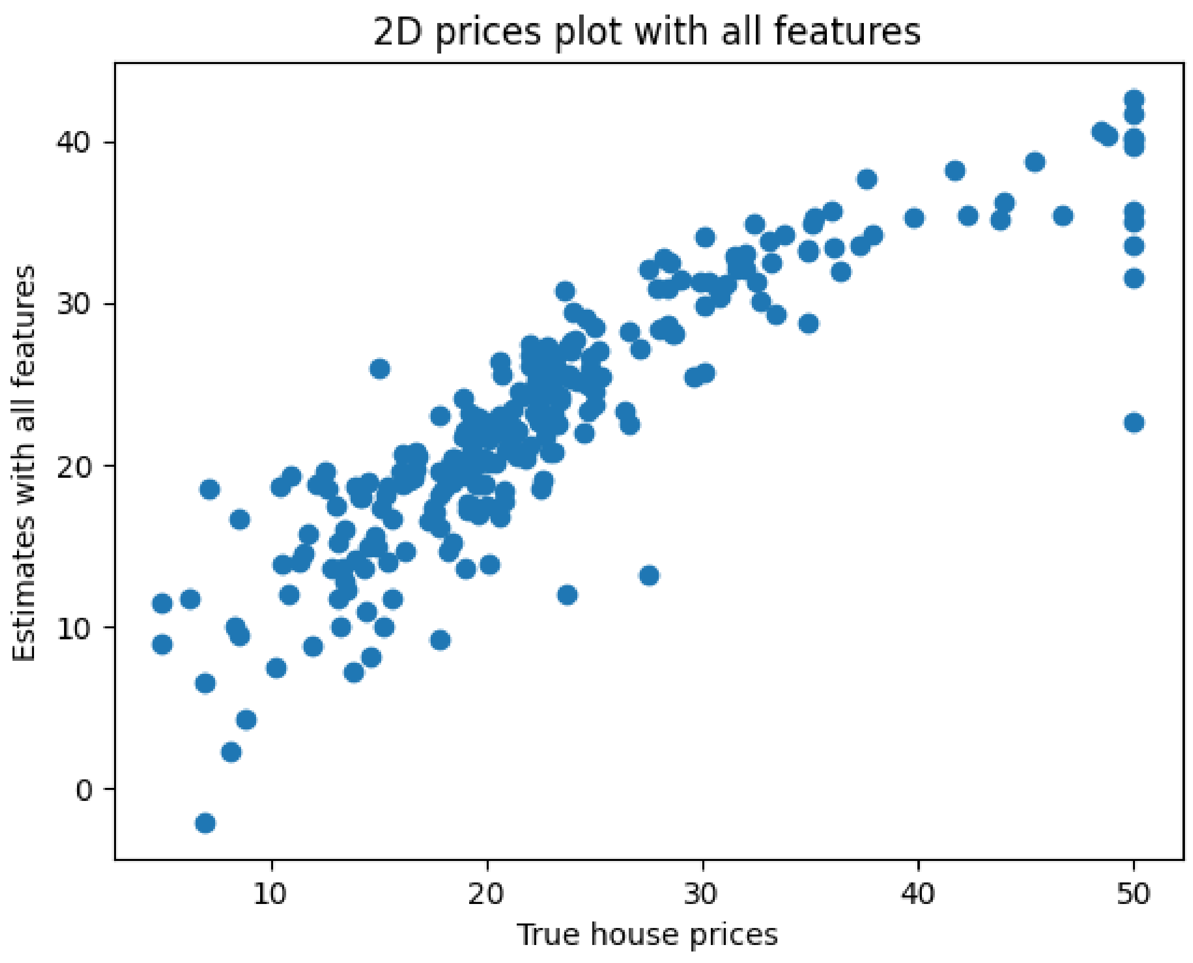

Figure 1 – Prices estimated on true house prices (unweighted linear regression)

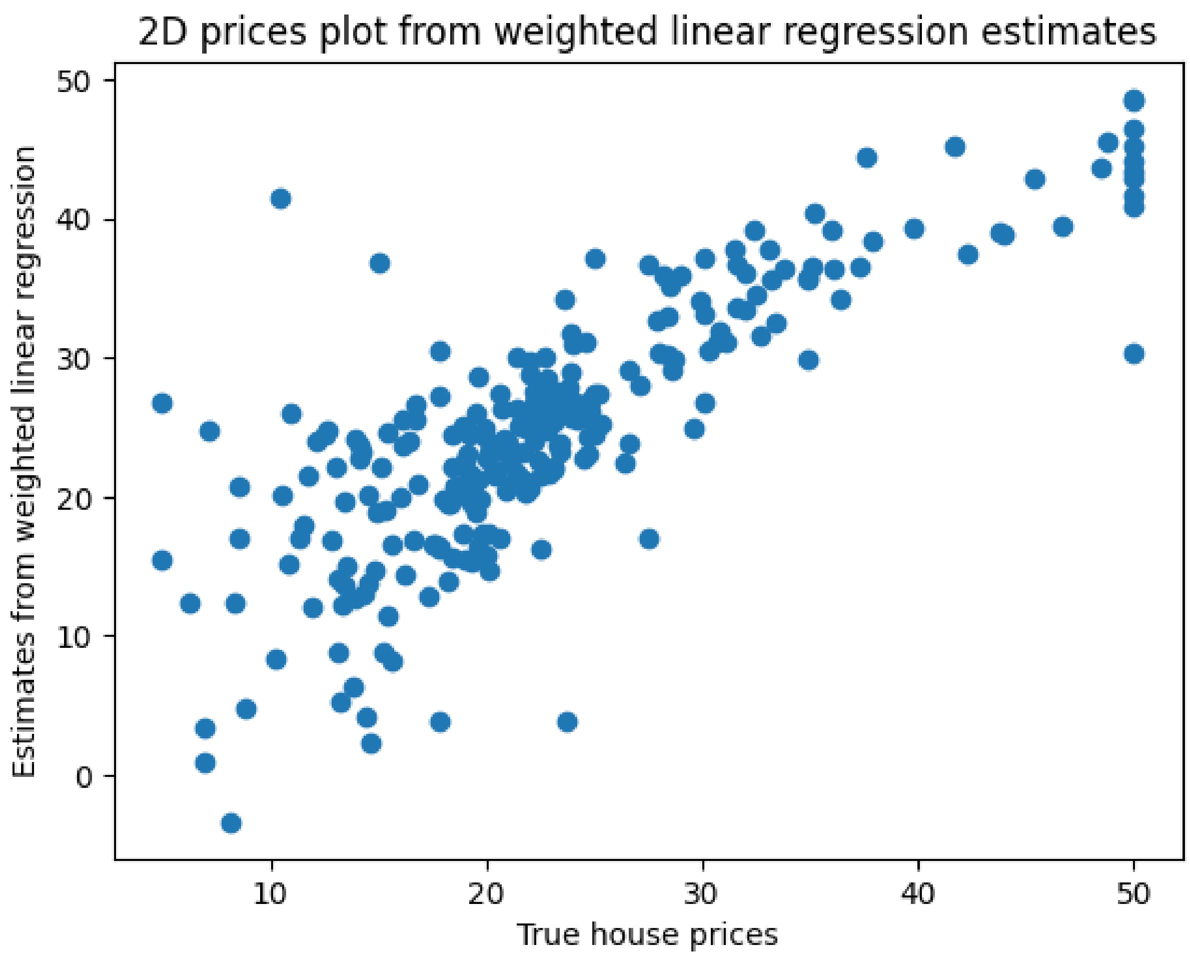

Figure 2 – Prices estimated on true house prices (weighted linear regression)

We can see that the weighted plot is more dispersed than the unweighted one, therefore, the additional have an impact on the regression.

Python

```python
import numpy as np

class LinearRegression:
    """
    Linear regression implementation.
    """

    def __init__(self):
        pass

    def fit(self, X, t):
        """
        Fits the linear regression model.

        Parameters
        ----------
        X : Array of shape [n_samples, n_features]
        t : Array of shape [n_samples, 1]
        """

        X = np.array(X).reshape((len(X), -1))
        t = np.array(t).reshape((len(t), 1))

        ones = np.ones((X.shape[0], 1))
        X = np.concatenate((ones, X), axis=1)

        A = np.diagflat(np.square(t))

        self.w = np.linalg.solve(X.T @ A @ X, X.T @ A @ t)

    def predict(self, X):
        """
        Computes predictions for a new set of points.

        Parameters
        ----------
        X : Array of shape [n_samples, n_features]

        Returns
        -------
        predictions : Array of shape [n_samples, 1]
        """
        X = np.array(X).reshape((len(X), -1))

        ones = np.ones((X.shape[0], 1))
        X = np.concatenate((ones, X), axis=1)

        t_new = np.dot(X, self.w)
        return t_new
```

Python

```python
import numpy as np
import pandas
import linweightreg
import matplotlib.pyplot as plt

# load data
train_data = np.loadtxt("boston_train.csv", delimiter=",")
test_data = np.loadtxt("boston_test.csv", delimiter=",")
X_train, t_train = train_data[:, :-1], train_data[:, -1]
X_test, t_test = test_data[:, :-1], test_data[:, -1]
# make sure that we have N-dimensional Numpy arrays (ndarray)
t_train = t_train.reshape((len(t_train), 1))
t_test = t_test.reshape((len(t_test), 1))
print("Number of training instances: %i" % X_train.shape[0])
print("Number of test instances: %i" % X_test.shape[0])
print("Number of features: %i" % X_train.shape[1])

#Exercise 1, b)
model_all = linweightreg.LinearRegression()
model_all.fit(X_train, t_train)
print(model_all.w)

t_predict = model_all.predict(X_test)
plt.scatter(t_test, t_predict)
plt.xlabel("True house prices")
plt.ylabel("Estimates from weighted linear regression")
plt.title("2D prices plot from weighted linear regression estimates")
plt.savefig("WeightedLinearReg.png", bbox_inches="tight")
plt.show()
```

Exercise 2

a)

First, we create a new class called polynomial with a custom method transform, to convert our data to a given polynomial form. Next we implements fit and predict methods as in the linear regression class but slightly modified to use our transform method and to take in consideration the regularization parameters lambda. Using this class, we then have to compute a t_predict array for each lambda value and compare it to the real value (t). To do that, we use leave-one-out cross-validation to predict each value in the t_predict array. We can then create a plot using the RMSE in function of the lambda values in order to better visualize which value of lambda provides the smallest error and, therefore, the best predictions.

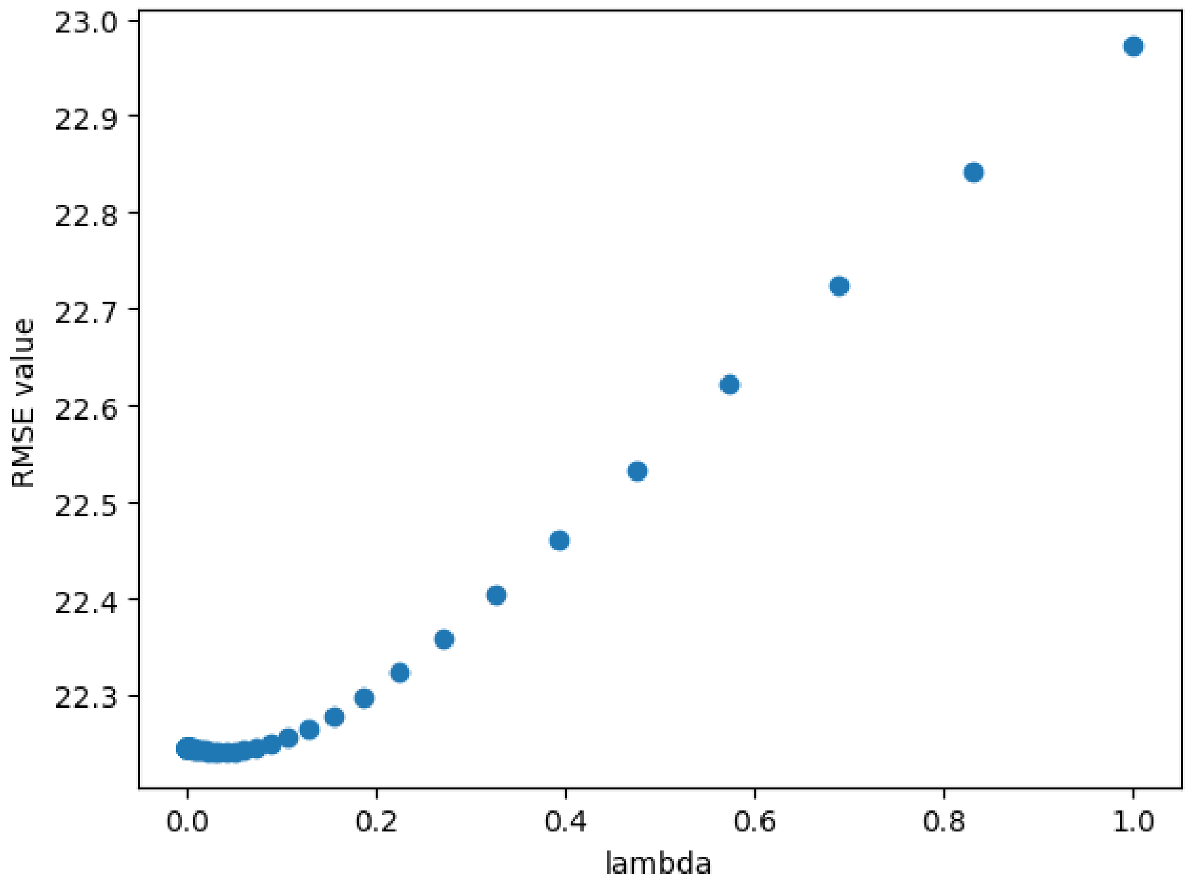

Figure 3 – Plot of the leave-one-out cross-validation error as a function of lambda for the first order polynomial

We can find the regression coefficient corresponding to the best lambda value as well as those corresponding to lambda = 0. We can see that the optimal value for lambda (the lowest rmse) is equal to 0.035.

Python

```
>>> lam=0.0351119173 and w0=12.4633935433 and w1=-0.5858196461
>>> lam=0.0000000000 and w0=12.9462989291 and w1=-0.6034315538
```

b)

We can use the exact same method for a fourth order polynomial by changing the order value to 4.

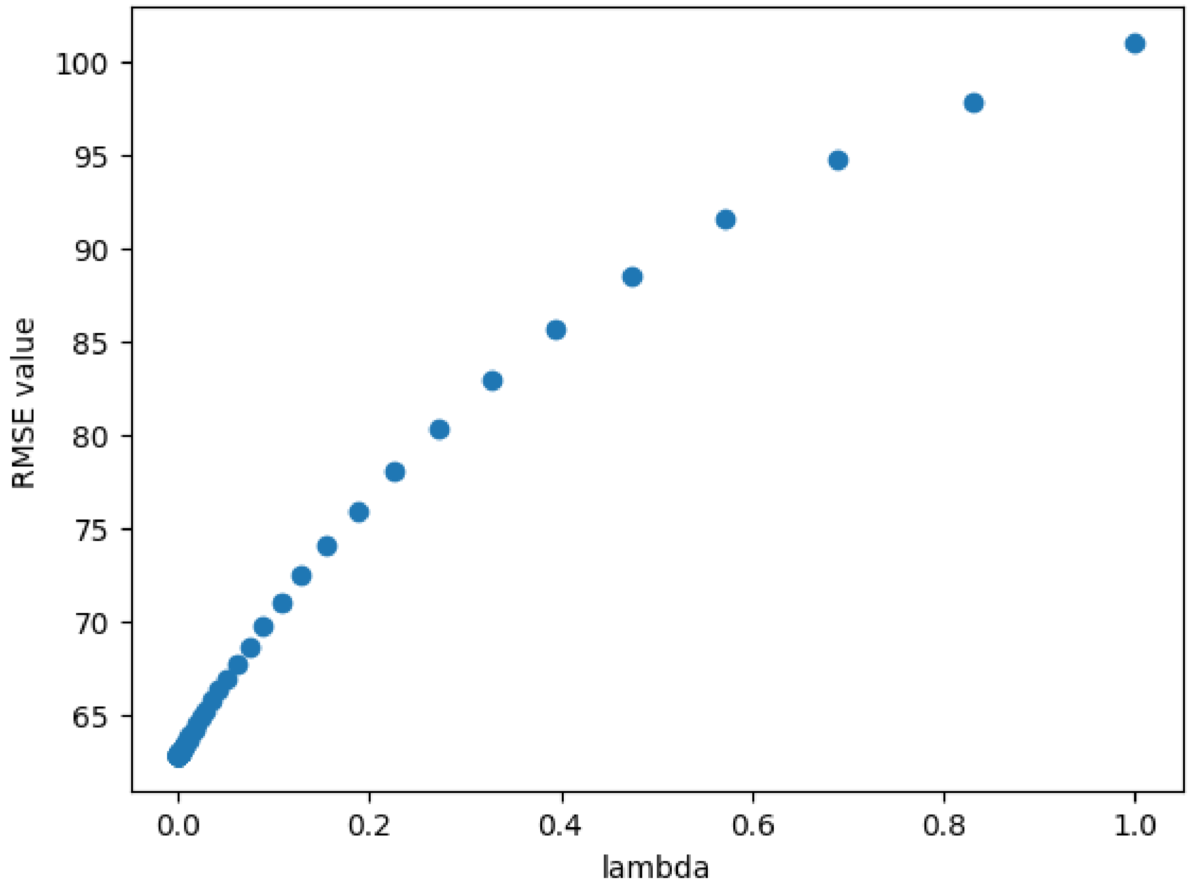

Figure 4 – Plot of the leave-one-out cross-validation error as a function of lambda for the fourth order polynomial

For the fourth order polynomial regression, we can see that the best lambda is equal 0 meaning that the model is optimal for the smallest value of lambda.

Python

```
>>> lam=0.0000000000 and w0=16.9306372038 and w1=-4.9101731170
    and w2=0.3427353380 and w3=-0.0079147499 and w4=0.0000560828
>>> lam=0.0000000000 and w0=16.9306372038 and w1=-4.9101731170
    and w2=0.3427353380 and w3=-0.0079147499 and w4=0.0000560828
```

Python

```python
import numpy as np


class Polynomial:
    """
    Linear regression implementation.
    """

    def __init__(self, order):
        self.order = order
        pass

    def transform(self, X):
        """create the X matrix"""

        ones = np.ones((X.shape[0], 1))
        X = np.concatenate([np.power(X, i) for i in range(1, self.order + 1)], axis=1)
        return np.concatenate((ones, X), axis=1)

    def fit(self, X, t, l):
        """
        Fits a n order polynomial to the data
        """

        X = self.transform(np.array(X).reshape((len(X), -1)))
        t = np.array(t).reshape((len(t), 1))

        # self.w = np.linalg.inv(X.T @ X + X.shape[0] @ l) @ X.T @ t
        self.w = np.linalg.solve(X.T @ X + X.shape[0] * l, X.T @ t)

    def predict(self, X):
        """
        Computes predictions for a new set of points.

        Parameters
        ----------
        X : Array of shape [n_samples, n_features]

        Returns
        -------
        predictions : Array of shape [n_samples, 1]
        """
        X = self.transform(np.array(X).reshape((len(X), -1)))

        t_new = np.dot(X, self.w)
        return t_new
```

Python

```python
import numpy as np
import pandas
import linweightreg
import Polynomial
import matplotlib.pyplot as plt


def rmse(t, tp):
    return np.sqrt(np.mean((t - tp) ** 2))


# load data
train_data = np.loadtxt("boston_train.csv", delimiter=",")
test_data = np.loadtxt("boston_test.csv", delimiter=",")
X_train, t_train = train_data[:, :-1], train_data[:, -1]
X_test, t_test = test_data[:, :-1], test_data[:, -1]
# make sure that we have N-dimensional Numpy arrays (ndarray)
t_train = t_train.reshape((len(t_train), 1))
t_test = t_test.reshape((len(t_test), 1))
```

```python
print("Number of training instances: %i" % X_train.shape[0])
print("Number of test instances: %i" % X_test.shape[0])
print("Number of features: %i" % X_train.shape[1])

# Exercise 1, b)
linearRegression = linweightreg.LinearRegression()
linearRegression.fit(X_train, t_train)
print(linearRegression.w)

t_predict = linearRegression.predict(X_test)
plt.scatter(t_test, t_predict)
plt.xlabel("True house prices")
plt.ylabel("Estimates from weighted linear regression")
plt.title("2D prices plot from weighted linear regression estimates")
plt.savefig("WeightedLinearReg.png", bbox_inches="tight")
plt.show()

# Exercise 2, a)

raw = np.genfromtxt("men-olympics-100.txt", delimiter=" ")

X, t = train_data[:, 0], train_data[:, 1]
# reshape both arrays to make sure that we deal with
# N-dimensional Numpy arrays
t = t.reshape((len(t), 1))
X = X.reshape((len(X), 1))
print("Shape of our data matrix: %s" % str(X.shape))
print("Shape of our target vector: %s" % str(t.shape))

lambdas = np.concatenate([np.array([0]), np.logspace(-8, 0, 100, base=10)])
computed_rmse = np.zeros(lambdas.shape)

polynomial = Polynomial.Polynomial(order=1)
for idx, lam in enumerate(lambdas):
    t_predict = np.zeros(t.shape)
    for i in range(0, X.shape[0]):
        polynomial.fit(np.delete(X, i), np.delete(t, i), lam)
        t_predict[i] = polynomial.predict(X[i])
    computed_rmse[idx] = rmse(t, t_predict)
    # print("lam=%.10f and rmse=%.10f" % (lam, computed_rmse[idx]))


plt.scatter(lambdas, computed_rmse)
plt.xlabel("lambda")
plt.ylabel("RMSE value")
plt.savefig("lambda_rmse_firstorder.png", bbox_inches="tight")
plt.show()

idx = np.argmin(computed_rmse)
polynomial.fit(X, t, lambdas[idx])
print(
    "lam=%.10f and w0=%.10f and w1=%.10f"
    % (float(lambdas[idx]), polynomial.w[0, 0], polynomial.w[1, 0])
)
polynomial.fit(X, t, lambdas[0])
print(
    "lam=%.10f and w0=%.10f and w1=%.10f"
    % (float(lambdas[0]), polynomial.w[0, 0], polynomial.w[1, 0])
)

# Exercice 2, b)

computed_rmse = np.zeros(lambdas.shape)

polynomial = Polynomial.Polynomial(order=4)
for idx, lam in enumerate(lambdas):
    t_predict = np.zeros(t.shape)
    for i in range(0, X.shape[0]):
        polynomial.fit(np.delete(X, i), np.delete(t, i), lam)
        t_predict[i] = polynomial.predict(X[i])
    computed_rmse[idx] = rmse(t, t_predict)
    # print("lam=%.10f and rmse=%.10f" % (lam, computed_rmse[idx]))


plt.scatter(lambdas, computed_rmse)
plt.xlabel("lambda")
plt.ylabel("RMSE value")
plt.savefig("lambda_rmse_fourthorder.png", bbox_inches="tight")
plt.show()

idx = np.argmin(computed_rmse)
polynomial.fit(X, t, lambdas[idx])
print(
    "lam=%.10f and w0=%.10f and w1=%.10f and w2=%.10f and w3=%.10f and w4=%.10f"
    % (
        float(lambdas[idx]),
        polynomial.w[0, 0],
        polynomial.w[1, 0],
        polynomial.w[2, 0],
        polynomial.w[3, 0],
        polynomial.w[4, 0],
    )
)
polynomial.fit(X, t, lambdas[0])
print(
    "lam=%.10f and w0=%.10f and w1=%.10f and w2=%.10f and w3=%.10f and w4=%.10f"
    % (
        float(lambdas[0]),
        polynomial.w[0, 0],
        polynomial.w[1, 0],
        polynomial.w[2, 0],
        polynomial.w[3, 0],
        polynomial.w[4, 0],
    )
)
```

Exercise 3

a)

To get the pdf from the cdf, we need to find the derivative. We only need to calculate it for the part where $x > 0$ because if we derive 0 (which the value F(x) takes for $x \leq 0$), we get 0 :

$$f(x) = -e^{-\beta x^\alpha}(-\beta x^\alpha)' = \alpha\beta x^{\alpha-1}e^{-\beta x^\alpha}$$

b)

To find the probability of the chip working longer than 4 years, we use the cdf function :

$$P(X > 4) = F(4) = 1 - e^{-\frac{4^2}{4}} = 1 - e^{-4} = 0.98$$

The probability of the chip working longer than 4 years is equal to 0,98. Therefore, there is a high probability that the chip will work longer than 4 years. To find the probability of the chip working between 5 and 10 years, we must use the pdf as this is an interval :

$$P(5 \leq X \leq 4) = \int_5^{10} f(x)\,dx = \int_5^{10} \frac{1}{2}xe^{-\frac{x^2}{4}}\,dx$$

Let $u = -\frac{x^2}{4}$, we then have $du = -\frac{x}{2}dx$ and $dx = -2\frac{du}{x}$. We can then rewrite our expression :

$$P(5 \leq X \leq 4) = -\int_{-\frac{25}{4}}^{-25} \frac{x}{2} e^u \frac{2du}{x} = \int_{-25}^{-\frac{25}{4}} e^u \, du = e^{-\frac{25}{4}} - e^{-25} = 0.0019$$

the probability that the chip will work between 5 and 10 years is equal to 0.0019. This probability is really low.

c)

To find the median of a chip life span, we must find the value at which the cdf is equal to 0.5 :

$$\begin{aligned} F(x) &= \frac{1}{2} \\ \Leftrightarrow 1 - e^{-\frac{x^2}{x}} &= \frac{1}{2} \\ \Leftrightarrow e^{-\frac{x^2}{x}} &= \frac{1}{2} \\ \Leftrightarrow e^{\frac{x^2}{x}} &= 2 \\ \Leftrightarrow \frac{x^2}{x} &= ln(2) \\ \Leftrightarrow x^2 &= 4ln(2) \\ \Leftrightarrow x &= \sqrt{4ln(2)} \text{ or } x = -\sqrt{4ln(2)} \\ \Leftrightarrow x &= 1.665 \text{ or } x = -1.665 \end{aligned}$$

It is impossible to have negative years (here negative life span) so the median life span value for the chip is 1.665 years.

Exercise 4

Firstly, to answer this question, we need to define some of our events (NC and C are given as no history of conviction and history of conviction.) :

- T : talks to the police.
- $\bar{T}$: Doesn't talk to the police.
- IN : end up in court.
- $I\bar{N}$: Doesn't go to court.
- CO : is convicted.
- $C\bar{O}$: isn't convicted.

a)

We are looking for the conditional probabilities of someone being convicted given that they have no history of conviction, they've been to court, and they didn't talk to the police ($P(CO|T, IN, NC)$) and the same probabilities given that the person have talked to the police ($P(CO|\bar{T}, IN, NC)$).

we are given the probability of someone not being convicted given that they have no history of conviction, that they didn't talk to the police and that they have been to court ($P(\bar{CO}|\bar{T}, IN, NC) = \frac{0.5}{4} = 0.125$).
We also know that $P(\bar{CO}|\bar{T}, IN, NC) + P(CO|\bar{T}, IN, NC) = 1$
Therefore : $P(CO|\bar{T}, IN, NC) = 1 - 0.125 = 0,875$ For the second part of the question, we can use the data from 2) and 4) : $P(CO|T, IN, NC) = P(IN|NC, T) * P(CO|NC, T) = 0.5 * 0.002 = 0,001$

We then multiply these probabilities of getting convicted by the time in jail the person will be sentenced to jail considering the conditions : if the person talked, it will be 5*0.75 = 3.75 and if the person didn't talk it will be 5 years.
people who didn't : $0,875 * 3.75 = 3.28$
people who talked : $0,001 * 5 = 0.00375$

b)

In the same way, we need to find the conditional probabilities of someone being convicted given that they have a history of conviction, they've been to court, and they didn't talk to the police ($P(CO|T, IN, C)$) and the same probabilities given that the person have talked to the police ($P(CO|\bar{T}, IN, C)$).

we are also given the probability of someone not being convicted given that they have a history of conviction, that they didn't talk to the police and that they have been to court ($P(\bar{CO}|\bar{T}, IN, C) = \frac{0.1}{4} = 0.025$).
We also know that $P(\bar{CO}|\bar{T}, IN, C) + P(CO|\bar{T}, IN, C) = 1$
Therefore : $P(CO|\bar{T}, IN, NC) = 1 - 0.025 = 0,975$ For the second part of the question, we can use the data from 2) and 5) : $P(CO|T, IN, NC) = P(IN|NC, T) * P(CO|NC, T) = 0.9 * 0.005 = 0,0045$

We then multiply these probabilities of getting convicted by the time in jail the person will be sentenced to jail considering the conditions : if the person talked, it will be 5*0.75 = 3.75 and if the person didn't talk it will be 5 years.
people who didn't : $0,975 * 3.75 = 3.66$
people who talked : $0,0045 * 5 = 0.00225$

If the results seem plausible for the people didn't talk, it is obvious that there is a mistake in computing the 2 others probabilities.