# Assignment 2 - MAD

Mikkel Willén

November 2021

## Exercise 1

### a)

Vi har

$$A = (\mathbf{a} \cdot 1^T) \cdot \mathbf{I}, \quad \mathbf{a} = \left[a_1, a_2, \ldots, a_n\right]^T, \quad 1 = \left[1, 1, \ldots, 1\right]^T$$

Vi kan derfor skrive funktionen som

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N} A(X\mathbf{w} - \mathbf{t})^T (X\mathbf{w} - \mathbf{t})$$

Vi simplificere funktionen

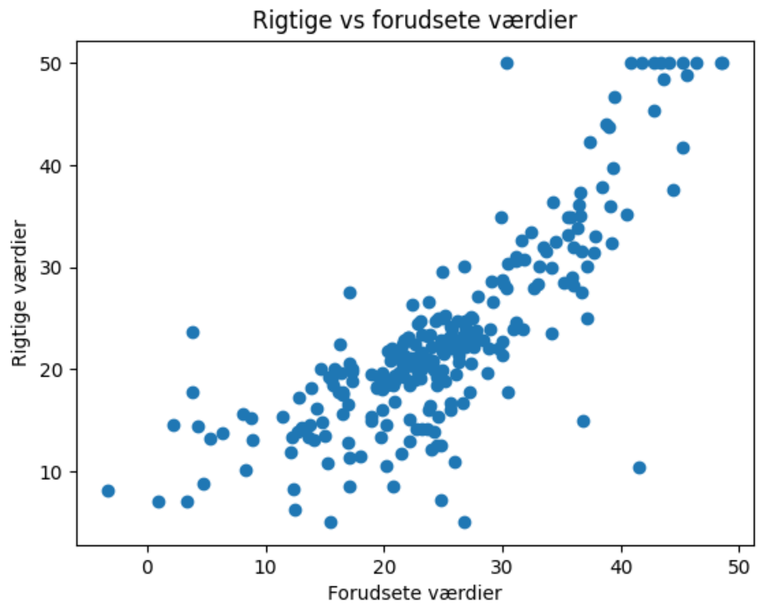
$$\begin{aligned}
\mathcal{L}(\mathbf{w}) &= \frac{1}{N} A(X\mathbf{w} - \mathbf{t})^T (X\mathbf{w} - \mathbf{t}) \\
&= \frac{1}{N} A((X\mathbf{w})^T - \mathbf{t}^T)(X\mathbf{w} - \mathbf{t} \\
&= \frac{1}{N} A(X\mathbf{w})^T X\mathbf{w} - \frac{1}{N}\mathbf{t}^T X\mathbf{w} - \frac{1}{N} A(X\mathbf{w})^T \mathbf{t} + \frac{1}{N}\mathbf{t}^T\mathbf{t} \\
&= \frac{1}{N} A\mathbf{w}^T X^T X\mathbf{w} - \frac{2}{N} A\mathbf{w}^T X^T \mathbf{t} + \frac{1}{N} A\mathbf{t}^T\mathbf{t}
\end{aligned}$$

Vi finder nu gradienten for denne funktion

$$\begin{aligned}
&& \nabla\mathcal{L}(\mathbf{w}) &= \frac{2}{N} AX^T X\mathbf{w} - \frac{2}{N} AX^T\mathbf{t} \\
&& \nabla\mathcal{L}(\mathbf{w}) &= 0 \\
\Leftrightarrow && \frac{2}{N} AX^T X\mathbf{w} - \frac{2}{N} AX^T\mathbf{t} &= 0 \\
\Leftrightarrow && AX^T X\mathbf{w} &= AX^T\mathbf{t} \\
\Leftrightarrow && \mathbf{I}\mathbf{w} &= (X^T AX)^{-1} X^T A\mathbf{t} \\
&& \hat{\mathbf{w}} &= (X^T AX)^{-1} X^T A\mathbf{t}
\end{aligned}$$

### b)

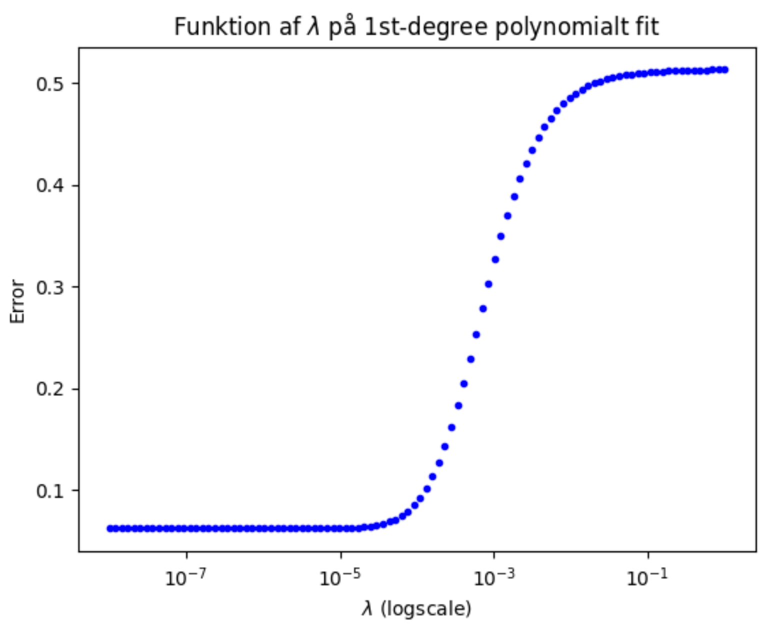
Vi forventede, at vi ville få en bedre løsning, end i den forrige opgave, men det har vi ikke fået. RMSE'en for den nye løsning er større end i den forrige. Disse vægte ændre markant på, hvad vi får ud af funktionerne.
Koden kan ses i appendix. Herunder er et billede, af vores scatterplot




# Exercise 2

Se kode i appendix.

## a)




Bedste lambdaværdi: 0.0000003430

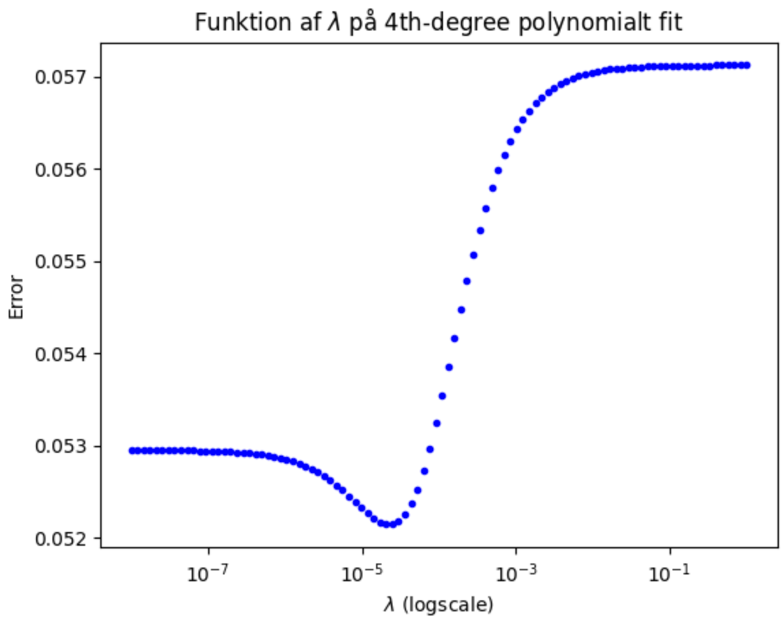
Værdier for $\lambda = 0$

$$\begin{bmatrix} 3.64164559 \cdot 10 \\ -1.33308857 \cdot 10^{-2} \end{bmatrix}$$

Værdier for $\lambda = 0.0000003430$

$$\begin{bmatrix} 3.63776282 \cdot 10 \\ -1.33110046e \cdot 10^{-2} \end{bmatrix}$$

**b)**




Bedste lambdaværdi: 0.0000205651

Værdier for $\lambda = 0$

$$\begin{bmatrix} 3.21358282 \cdot 10^{5} \\ -6.46374641 \cdot 10^{2} \\ 4.87449415 \cdot 10^{-1} \\ -1.63339180 \cdot 10^{-4} \\ 2.05197882 \cdot 10^{-8} \end{bmatrix}$$

Værdier for $\lambda = 0.0000205651$

$$\begin{bmatrix} 1.88300350 \cdot 10^{-2} \\ 9.11277821 \\ -1.38600662 \cdot 10^{-2} \\ 7.03376807e \cdot 10^{-6} \\ -1.19035006 \cdot 10^{-9} \end{bmatrix}$$

## Exercise 3

### a)

Vi differentiere funktionen

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

$$\begin{aligned}
\frac{d}{dx}\left(1 - e^{-\beta x^{\alpha}}\right) &= -\frac{d}{dx}\left(e^{-\beta x^{\alpha}}\right) && \text{konstant regl og sumregl} \\
&= \frac{d}{du}e^{u} \cdot u', \quad u = -\beta x^{\alpha} && \text{kædereglen} \\
&= e^{u} \cdot u', \quad u' = \left(-\frac{\beta x^{\alpha}\alpha}{x}\right) && \text{diff af begge funktioner} \\
&= e^{-\beta x^{\alpha}} \cdot \left(-\frac{\beta x^{\alpha}\alpha}{x}\right) && \text{sætter ind} \\
&= e^{-\beta x^{\alpha}} \beta\alpha x^{\alpha-1} && \text{reduceret}
\end{aligned}$$

Skriver pdf'en op

$$F(x) = \begin{cases} 0 & x \leq 0 \\ e^{-\beta x^{\alpha}} \beta\alpha x^{\alpha-1} & x > 0 \end{cases}$$

### b)

Vi sætter $\beta = \frac{1}{4}$ og $\alpha = 2$

$$F(x) = e^{-\frac{1}{4}x^{2}} \frac{1}{2}x$$

og udregner $F(4)$

$$F(4) = e^{-\frac{1}{4}4^{2}} \frac{1}{2}4 = 2 \cdot e^{-4} = 0.0366$$

herefter udregner vi $F(5) - F(10)$

$$F(5) - F(10) = e^{-\frac{1}{4}5^{2}} \frac{1}{2}5 - \left(e^{-\frac{1}{4}10^{2}} \frac{1}{2}10\right) = 0.00483$$

### c)

Vi finder ingegralet for pfd'en, som er cdf'en. Så løser vi nedenstående ligning for $m$

$$\begin{aligned}
& & 1 - e^{-\beta m^{\alpha}} - \left(1 - e^{-\beta 0^{\alpha}}\right) &= \frac{1}{2} \\
&\Leftrightarrow & 1 - e^{-\beta m^{\alpha}} &= \frac{1}{2} \\
&\Leftrightarrow & m &= e^{\frac{ln\left(\frac{ln(2)}{\beta}\right)}{\alpha}}
\end{aligned}$$

# Exercise 4

## a)

I denne opgave antager jeg, at der ikke skulle stå "not"i opgavebeskrivelsen da "not speak"og "silent"virker som det samme.
Jeg ganger de forskellige faktorer sammen og får:

$$X_{speak} = 0.002 \cdot 0.5 \cdot 0.75 = 0.00075$$

## b)

Jeg ganger de forskellige faktorer sammen og får:

$$X_{silent} = 0.001 \cdot (1 - 0.5 \cdot 0.25) = 0.000875$$

Det giver derfor mening at sige noget, da det giver en mindre potentiel straf.

## c)

jeg ganger de forskelige faktorer sammen og får:

$$Y_{speak} = 0.005 \cdot (1 - 0.1) \cdot 0.75 = 0.003375$$

$$Y_{silent} = 0.001 \cdot (1 - 0.1 \cdot 0.25) = 0.000975$$

Det giver derfor mening ikke at sige noget, da det giver en mindre potentiel straf.

# Appendix

## linweightreg.py

```
import numpy

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]
        """
        A = numpy.zeros((X.shape[0], X.shape[0]))
        for i in range(X.shape[0]):
            A[i][i] = t[i] ** 2
        print(A)
        X = numpy.reshape(X, (X.shape[0], -1))
        X = numpy.insert(X, 0, 1, axis=1)
        XTX = numpy.dot(numpy.dot(X.T, A), X)
        inverse = numpy.linalg.inv(XTX)
        XTXX = numpy.dot(inverse, X.T)
        self.w = numpy.dot(numpy.dot(XTXX, 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 = numpy.reshape(X, (X.shape[0], -1))
        X = numpy.insert(X, 0, 1, axis = 1)
        self.p = numpy.dot(X, self.w)
```

### housing_2.py

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

# load data
train_data = numpy.loadtxt("boston_train.csv", delimiter=",")
test_data = numpy.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])

# (b) fit linear regression using only the first feature
model_single = linweightreg.LinearRegression()
model_single.fit(X_train[:,0], t_train)
print("weights for a single variable: \n", model_single.w)
print("Husprisen starter på 23 tusinde dollars, når crimerate er 0, og falder med 0.4,
    når enheden for crimerate stiger med 1")

# (c) fit linear regression model using all features
model_multiple = linweightreg.LinearRegression()
model_multiple.fit(X_train, t_train)
print("weights for multiple variables: \n", model_multiple.w)
print("Husprisen starter på 31 tusiden dollars, når alle andre værdier er 0,
    og falder eller stiger, alt efter fortegnet på værdien, når de andre værdier stiger")

# (d) evaluation of results
model_single.predict(X_test[:,0])
model_multiple.predict(X_test)

def rmse(t, tp):
```

```
    N = t.shape[0]
    sum = 0.0
    for i in range (N):
        sum = sum + (t[i] - tp[i]) ** 2
    rmse = numpy.sqrt(sum/N)
    return rmse

print("RMSE for a single variable: ", rmse(t_test, model_single.p))
print("RMSE for multiple variables: ", rmse(t_test, model_multiple.p))

print("Plot of single variable")
plt.scatter(model_single.p, t_test)
print("Plot of multiple variables")
plt.scatter(model_multiple.p, t_test)
```

### linreg.py

```
import numpy

class LinearRegression():
    """
    Linear regression implementation with
    regularization.
    """

    def __init__(self, lam=0, solver="inverse"):
        self.lam = lam
        self.solver = solver

        assert self.solver in ["inverse", "solve"]

    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 = numpy.reshape(X, (X.shape[0], -1))
        X = numpy.insert(X, 0, 1, axis=1)
        t = numpy.reshape(t, (len(t),1))

        diagonal = self.lam * len(X) * numpy.identity(X.shape[1])
```

```
        km = numpy.dot(X.T, X) + diagonal

        if self.solver == "solve":
            self.w = numpy.linalg.solve(km, numpy.dot(X.T, t))

        elif self.solver == "inverse":
            self.w = numpy.linalg.inv(km)
            self.w = numpy.dot(self.w, X.T)
            self.w = numpy.dot(self.w, t)

        else:
            raise Exception("Unknown solver!")

    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 = numpy.reshape(X, (X.shape[0], -1))
        X = numpy.insert(X, 0, 1, axis = 1)
        self.p = numpy.dot(X, self.w)
```

### RegularizedLinReg.py

```
import matplotlib.pyplot as plt
import numpy as np
import linreg

solver = "solve"

def lossFunction(X, t, lam, verbose=0):

    loss = 0

    for i in range(len(X)):
        X_train = np.delete(X, i, 0)
        t_train = np.delete(t, i, 0)

        model = linreg.LinearRegression(lam=lam, solver=solver)
```

```
        model.fit(X_train, t_train)

        X_val = X[i].reshape((1, X_train.shape[1]))
        t_val = t[i].reshape((1, 1))
        model.predict(X_val)

        loss += (t_val[0,0] - model.p[0,0]) ** 2.0

    loss = loss / len(X)

    if verbose > 0:
        print("lam=%.10f and loss=%.10f" % (lam, loss))

    return loss

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

t = raw[:,1].reshape((len(raw),1))

lambdaValues = np.logspace(-8, 0, 100, base=10)
print("1st Order Polynomial: ")
X = raw[:,0].reshape((len(raw),1))

resultA = np.array([lossFunction(X, t, lam) for lam in lambdaValues])
bestLambdaA = lambdaValues[np.argmin(resultA)]
print("Best lambda value: %.10f" % bestLambdaA)

modelZero = linreg.LinearRegression(lam=0.0, solver=solver)
modelZero.fit(X, t)
print("Optimal coefficients for lam=%.10f: \n%s" % (0.0, str(modelZero.w)))

modelBest = linreg.LinearRegression(lam=bestLambdaA, solver=solver)
modelBest.fit(X, t)
print("Optimal coefficients for lam=%.10f: \n%s" % (bestLambdaA, str(modelBest.w)))

plt.figure()
plt.plot(lambdaValues, resultA, "bo", markersize=3)
plt.title("Funktion af $\lambda$ på 1st-degree polynomialt fit")
plt.ylabel("Error")
plt.xlabel("$\lambda$ (logscale)")
plt.xscale("log")
plt.show()

print("4th Degree Polynomial")
X4 = np.empty((len(raw[:,0]),4))
```

```python
X4[:,0] = raw[:,0]
X4[:,1] = raw[:,0] ** 2
X4[:,2] = raw[:,0] ** 3
X4[:,3] = raw[:,0] ** 4

resultB = np.array([lossFunction(X4, t, lam) for lam in lambdaValues])
bestLambdab = lambdaValues[np.argmin(resultB)]
print("Best lambda value: %.10f" % bestLambdab)

modelZero = linreg.LinearRegression(lam=0.0, solver=solver)
modelZero.fit(X4, t)
print("Optimal coefficients for lam=%.10f: \n%s" % (0.0, str(modelZero.w)))

modelBest = linreg.LinearRegression(lam=bestLambdab, solver=solver)
modelBest.fit(X4, t)
print("Optimal coefficients for lam=%.10f: \n%s" % (bestLambdab, str(modelBest.w)))

plt.figure()
plt.plot(lambdaValues, resultB, "bo", markersize=3)
plt.title("Funktion af $\lambda$ på 4th-degree polynomialt fit")
plt.ylabel("Error")
plt.xlabel("$\lambda$ (logscale)" )
plt.xscale("log")
plt.show()
```