1.1 – Linjär regression

1.1.1 - Introduktion

- I traditionell programmering används indata/insignaler i kombination med regler för att generera utdata/utsignaler. Exempel kan en temperatur läsas in som indata, där regler är satta för vad som ska ske vid olika temperaturer; vid en viss temperatur kanske en fläkt aktiveras och vid en lägre temperatur så stängs den av.

- I maskininlärning används indata/insignaler i kombination med utdata/utsignaler för att maskinen ska skapa egna regler för vad som gäller. Exempelvis kan maskinen lära sig utefter in- och utdata att vid en viss temperatur (indata), så aktiveras alltid fläkten (för att sänka temperaturen).

- Därmed hittar maskinen själv reglerna, exempelvis att över en viss temperatur aktiveras fläkten. När maskinen på detta sätt upptäcker reglerna själv sägs den bli tränad och datan den tränas på kallas träningsdata. Träningsdatan består av en eller flera träningsuppsättningar, där en given träningsuppsättning innehar en insignal x samt motsvarande utsignal y.

1.1.2 – Varför maskininlärning?

- Innan maskininlärning blev utbrett var symbolisk AI den dominanta paradigmen inom artificiell intelligens. Symbolisk AI påminner om traditionell programmering, med skillnaden att en maskin förses med en mycket stor mängd med regler för enormt olika scenarion, vilket troddes kunde resultera i människoliknande intelligens.

- Symbolisk AI, som var dominerande fram till sena 1980-talet, medförde att maskiner kunde lösa komplexa, men väldefinierade uppgifter, exempelvis spela schack. Dock visade sig symbolisk AI vara inadekvat för mindre väldefinierade uppgifter, såsom datorseende, taligenkänning och språköversättning, där indata (bilder, ord, fraser) kan variera stort gällande olika ljus, vinklar, vem som säger något, dialekt och dylikt.

- Därmed uppstod i stället maskininlärning, där maskinen själv får hitta reglerna genom att bli tränad via stora mängder indata samt motsvarande förväntad utdata. Ifall ett nytt scenario dyker upp får maskinen helt enkelt prediktera vad som ska göras utifrån vad den redan vet (har blivit tränad till), vilket påminner om hur vi människor fungerar. Detta är en stor fördel jämfört med en traditionellt programmerad maskin, som inte genomförs något ifall en regel för aktuellt scenario inte existerar.

1.1.3 - Vad krävs för en välfungerande maskininlärningsalgoritm?

- För att realisera en välfungerande maskininlärningsalgoritm krävs tre faktorer:

  1. Indata, exempelvis bilder på objekt, data från sensorer eller dylikt.
  2. Förväntad utdata, så att maskinen kan lära sig vilken utdata som förväntas för given indata.
  3. En metod för att mäta ifall algoritmen fungerar väl, alltså ifall aktuell utdata matchar förväntad utdata. Vid behov måste algoritmen justeras, vilket kallas lärande.

1.1.4 – En första maskininlärningsalgoritm - linjär regression

- En av de vanligaste maskininlärningsalgoritmerna inom maskininlärning är linjär regression, som i praktiken innebär att maskinen tränas till att detektera ett linjärt mönster mellan indata x samt utdata y. Ur detta mönster kan en rät linje erhållas enligt nedanstående förstagradsekvation:

$$y = kx + m,$$

  där k är linjens lutning, och m är linjens vilovärde (värdet på utsignal y då insignal x är lika med noll).

- Inom maskininlärning används ofta begreppen vikt samt biasvärde i stället för k- och m-värde:

  1. Insignal x sägs multipliceras med en vikt k.
  2. Modellens biasvärde m utgör modellens vilovärde, vilket utgör utsignal y då insignal x är lika med noll.

- Särskilt för neurala nätverk, där en så kallad nod sägs inneha ett biasvärde (m-värde) samt en eller flera vikter (k-värden). I praktiken är det dock samma; nodens insignaler x multipliceras med vikter k och summan k * x av dessa tillsammans med biasvärdet m utgör nodens utsignal y.

- Noder i neurala nätverk är dock något mer komplicerade än så, då utsignal y vanligtvis måste överstiga ett tröskelvärde, vanligtvis noll, annars blir utsignalen noll och noden är ”inaktiverad”, likt noder i en hjärna, som kräver en viss stimulans från dess insignaler för att noden ska aktiveras. Neurala nätverk beskrivs i nästa kapitel, *1.2 – Neurala nätverk.*

- För att träna modellen krävs träningsdata i form av indata x samt motsvarande utdata y. En uppsättning bestående av en insignal x samt en utsignal y utgör en träningsuppsättning. I tabell 1 nedan demonstreras ett exempel på fem träningsuppsättningar, där förhållandet mellan insignal x samt utsignal y kan beskrivas med formeln y = 2x + 1:

x	y
0	1
1	3
2	5
3	7
4	9

Tabell 1 – Träningsdata i form av fem träningsuppsättningar.

- Träningsdatan bör inneha ett flertal olika träningsuppsättningar, alltså ett flertal uppsättningar av indata x samt utdata y. Vid träning kan samtliga träningsuppsättningar används för att träna modellen flera omgångar, föredragsvis i slumpvis ordning för att modellen inte ska blir för bekant med träningsdatan. Antalet omgångar som träningsdatan används för att träna modellen kallas epoker eller *epochs.*

- Vid träning bör sedan predikterad utdata y_p jämföras mot motsvarande referensvärde från träningsdatan y_{ref}. Differensen mellan dessa utgör avvikelsen δ:

$$\delta = y_{ref} - y_p,$$

  där δ är avvikelsen mellan referensvärde y_{ref} samt predikterad utdata y_p.

- Målet är att avvikelsen δ ska hamna så nära noll som möjligt, då modellen fungerar så bra som möjligt:

$$\delta = 0 => modellen\ fungerar\ utmärkt$$

- Om avvikelsen δ är positiv, så är predikterat värde y_p för lågt, vilket innebär att modellens k- och m-värde bör ökas:

$$\delta > 0 => k\ och\ m\ bör\ ökas$$

- På samma sätt gäller att om avvikelsen δ är negativ, så är är predikterat värde y_p för högt, vilket innebär att modellens k- och m-värde bör minskas:

$$\delta < 0 => k\ och\ m\ bör\ minskas$$

- Vid avvikelse bör modellens k- och m-värde justeras en viss justeringsmängd Δe, som utgör en faktor av avvikelsen δ enligt nedan:

$$\Delta e = \delta * LR,$$

  där LR utgör lärhastigheten, även kallat *learning rate,* som bör justeras efter hur väl modellen fungerar. Ju högre lärhastighet, desto kraftigare justeras k- och m-värdet vid avvikelse. För hög lärhastighet kan dock medföra för justering per k- och m-värde. Som ett startvärde kan L sättas till omkring 1 %, vilket motsvarar 0.01 vid beräkningarna. Detta värde bör sedan justerar tillsammans med antalet epoker (antalet träningsomgångar av aktuellt antal träningsuppsättningar).

- Modellens m-värde bör ökas med justeringsmängden Δe. Om avvikelsen δ överstiger noll, så reduceras då m. Annars om δ understiger noll, så ökas m:

$$m = m + \Delta e$$

- Modellens k-värde bör ökas med justeringsmängden Δe multiplicerat med aktuell indata x. Därmed gäller att då x är lika med noll, då enbart m-värdet avgör utsignal y, så justeras inte k-värdet vid avvikelse, utan enbart m-värdet. Samtidigt gäller att ju högre x-värde, desto mer justeras k-värdet för given indata x, då eventuell avvikelse till större del då utgörs av lutningen k * x i stället för m-värdet:

$$k = k + \Delta e * x$$

1.1.5 – Träning av regressionsmodell för hand

- En regressionsmodell ska tränas via de fem träningsuppsättningarna definierade enligt formeln y = 2x + 1 i tabell 1:

x	y
0	1
1	3
2	5
3	7
4	9

Tabell 1 – Fem träningsuppsättningar.

- Anta att modellens bias (m-värde) samt vikt (k-värde) är noll vid start:

$$\begin{cases} k = 0 \\ m = 0 \end{cases}$$

- Genomför träning under en epok med en lärhastighet LR på 10 %:

$$LR = 0.1$$

- Genomför sedan prediktion för indata bestående av alla heltal inom intervallet [-5, 5].

Lösning

- Vi genomför träning för varje träningsuppsättning en efter en.

Träningsuppsättning 1

- Från den första träningsuppsättningen erhålls indata x = 0 samt referensvärde y_{ref} = 1:

$$\begin{cases} x = 0 \\ y_{ref} = 1 \end{cases}$$

- Eftersom modellens parametrar är lika med noll vid start blir predikterad utdata y_p lika med noll, då

$$y_p = k * x + m = 0 * 0 + 0 = 0$$

- Avvikelsen δ blir därmed lika med ett, då

$$\delta = y_{ref} - y_p = 1 - 0 = 1$$

- För en lärhastighet LR på 10% blir därmed justeringsmängden Δe lika med 0.1, då

$$\Delta e = \delta * LR = 1 * 0.1 = 0.1$$

- Modellens m-värde ökas direkt med justeringsmängden Δe, vilket medför en ökning till 0.1, då

$$m = m + \Delta e = 0 + 0.1 = 0.1$$

- Modellens k-värde ökas med justeringsmängden Δe multiplicerat med aktuell indata x, vilket när x = 0 medför ingen förändring, då aktuell avvikelse enbart beror på m-värdet:

$$k = k + \Delta e * x = 0 + 0.1 * 0 = 0$$

- Efter den första träningsrundan har därmed regressionsmodellens parametrar justerats till följande:

$$\begin{cases} k = 0 \\ m = 0.1 \end{cases}$$

Träningsuppsättning 2

- Från den andra träningsuppsättningen erhålls indata x = 1 samt referensvärde y_{ref} = 3:

$$\begin{cases} x = 1 \\ y_{ref} = 3 \end{cases}$$

- Predikterad utdata y_p blir nu lika med 0.1, då

$$y_p = k * x + m = 0 * 1 + 0.1 = 0.1$$

- Avvikelsen/aktuellt fel δ blir därmed lika med 2.9, då

$$\delta = y_{ref} - y_p = 3 - 0.1 = 2.9$$

- För en lärhastighet på 10% blir därmed justeringsmängden Δe lika med 0.29, då

$$\Delta e = \delta * LR = 2.9 * 0.1 = 0.29$$

- Modellens m-värde ökas direkt med justeringsmängden Δe, vilket medför en ökning till 0.39:

$$m = m + \Delta e = 0.1 + 0.29 = 0.39$$

- Modellens k-värde ökas med justeringsmängden Δe multiplicerat med aktuell indata x, vilket medför en ökning till 0.29:

$$k = k + \Delta e * x = 0 + 0.29 * 1 = 0.29$$

- Efter den andra träningsrundan har därmed regressionsmodellens parametrar justerats till följande:

$$\begin{cases} k = 0.29 \\ m = 0.39 \end{cases}$$

Träningsuppsättning 3

- Från den tredje träningsuppsättningen erhålls indata x = 2 samt referensvärde y_{ref} = 5:

$$\begin{cases} x = 2 \\ y_{ref} = 5 \end{cases}$$

- Predikterad utdata y_p blir lika med 0.97, då

$$y_p = k * x + m = 0.29 * 2 + 0.39 = 0.97$$

- Avvikelsen δ blir därmed lika med 2.9, då

$$\delta = y_{ref} - y_p = 5 - 0.97 = 4.03$$

- För en lärhastighet på 10% blir därmed justeringsmängden Δe lika med 0.403, då

$$\Delta e = \delta * LR = 4.03 * 0.1 = 0.403$$

- Modellens m-värde ökas direkt med justeringsmängden Δe, vilket medför en ökning till 0.793, då

$$m = m + \Delta e = 0.39 + 0.403 = 0.793$$

- Modellens k-värde ökas med justeringsmängden Δe multiplicerat med aktuell indata x, vilket medför en ökning till 1.096:

$$k = k + \Delta e * x = 0.29 + 0.403 * 2 = 1.096$$

- Efter den tredje träningsrundan har därmed regressionsmodellens parametrar justerats till följande:

$$\begin{cases} k = 1.096 \\ m = 0.793 \end{cases}$$

- Notera att parametrarna börjar närma sig önskade värden (k = 2, m = 1).

Träningsuppsättning 4

- Från den fjärde träningsuppsättningen erhålls indata x = 3 samt referensvärde y_{ref} = 7:

$$\begin{cases} x = 3 \\ y_{ref} = 7 \end{cases}$$

- Predikterad utdata y_p blir lika med 4.381, då

$$y_p = k * x + m = 1.096 * 3 + 0.793 = 4.081$$

- Avvikelsen δ blir därmed lika med 2.9, då

$$\delta = y_{ref} - y_p = 7 - 4.081 = 2.919$$

- Notera att avvikelsen δ nu för första gången har börjat minska, vilket också medför att justering av regressionsmodellens parametrar börjar minska.

- För en lärhastighet på 10% blir därmed justeringsmängden Δe lika med 0.2919, då

$$\Delta e = \delta * LR = 2.919 * 0.1 = 0.2919$$

- Modellens m-värde ökas med justeringsmängden Δe, vilket medför en ökning till 1.0849, då

$$m = m + \Delta e = 0.793 + 0.2919 = 1.0849$$

- Modellens k-värde ökas med justeringsmängden Δe multiplicerat med aktuell indata x, vilket medför en ökning till 1.9717:

$$k = k + \Delta e * x = 1.096 + 0.2919 * 3 = 1.9717$$

- Efter den fjärde träningsrundan har därmed regressionsmodellens parametrar justerats till följande:

$$\begin{cases} k = 1.9717 \\ m = 1.0849 \end{cases}$$

- Notera att parametrarna är mycket nära önskade värden (k = 2, m = 1).

Träningsuppsättning 5

- Från den femte träningsuppsättningen erhålls indata x = 4 samt referensvärde y_{ref} = 9:

$$\begin{cases} x = 4 \\ y_{ref} = 9 \end{cases}$$

- Predikterad utdata y_p blir nu lika med 8.9717, då

$$y_p = k * x + m = 1.9717 * 4 + 1.0849 = 8.9717$$

- Avvikelsen δ blir därmed lika med 0.0283, då

$$\delta = y_{ref} - y_p = 9 - 8.9717 = 0.0283$$

- För en lärhastighet på 10% blir därmed justeringsmängden Δe lika med 0.00283 då

$$\Delta e = \delta * LR = 0.0283 = 0.1 = 0.00283$$

- Modellens m-värde ökas med justeringsmängden Δe, vilket medför en ökning till 1.08773, då

$$m = m + \Delta e = 1.0849 + 0.00283 = 1.08773$$

- Notera att m-värdet nu justerades från önskat värde m = 1. Detta kommer ske så länge predikterat värde y_p understiger referensvärdet y_{ref}. Men så länge avvikelsen δ är nära noll blir förändringen minimal.

- Modellens k-värde ökas med justeringsmängden Δe multiplicerat med aktuell indata x, vilket medför en ökning till 1.97453:

$$k = k + \Delta e * x = 1.9717 + 0.00283 * 4 = 1.97453$$

- Notera att k-värdet nu hamnade närmare önskat värde k = 2. Förändringen blev dock relativt liten, eftersom avvikelsen är låg. Ifall fler epoker hade genomförts hade k- och m-värdet mycket långsamt hamnat mycket när önskade värden.

- Efter den femte träningsrundan har därmed regressionsmodellens parametrar justerats till följande:

$$\begin{cases} k = 1.97453 \\ m = 1.08773 \end{cases}$$

- Notera att enbart efter en epok har regressionsmodellens parametrar hamnat mycket nära önskade värden (k = 2, m = 1). Normalt genomförs mycket fler epoker än så, exempelvis 1000 – 10 000 epoker. Samtidigt brukar lärhastigheten ofta vara lägre, vilket medför mindre justering av parametrarna per epok.

- Efter genomförd träning under en epok predikterar därmed regressionsmodellens enligt följande formel:

$$y_p = 1.97453 * x + 1.08773,$$

där y_p utgör predikterad utdata och x utgör indata.

Verifiering

- I tabell 2 nedan visas predikterad utdata y_p samt referensvärden (önskad utdata) för indata x i intervallet [-5, 5] i enlighet med formeln y = 1.97453 * x + 1.08773. Predikterad utdata har avrundats till två decimaler.

x	y_p	y_{ref}
-5	-8.79	-9
-4	-6.81	-7
-3	-4.83	-5
-2	-2.86	-3
-1	-0.89	-1
0	1.09	1
1	3.06	3
2	5.04	5
3	7.01	7
4	8.99	9
5	10.96	11

Tabell 2 – Indata x samt motsvarande predikterad utdata y_p och önskad utdata y_{ref}.

- Notera att predikterad utdata y_p i samtliga fall hamnar nära önskad utdata y_{ref} efter genomförd träning under en enda epok!

1.1.6 – Exempel 1 a) – Enkel implementering av linjär regression i C++

- I detta första exempel demonstreras en mycket enkel implementering av en modell för linjär regression via en strukt döpt *lin_reg*, där fem träningsuppsättningar definierade enligt formeln y = 10x + 2 lagras via två vektorer. Träningsuppsättningarna visas i tabell 4 nedan.

x	y
0	2
1	12
2	22
3	32
4	42

Tabell 4 – Träningsuppsättningar i Exempel 1 a).

- Träning sker som default under 10 000 epoker med en lärhastighet på 1 %, men det är möjligt att välja dessa parametrar via ingående argument när programmet körs. Efter att träning har genomförts sker prediktion för insignaler bestående av alla heltal inom intervallet [-10, 10]. Samtliga insignaler x samt predikterade utsignaler y skrivs sedan ut i terminalen. Träningen har lyckats väl och modellen predikterar med 100 % precision.

- *Se bilaga A - Exempel 1 b) för motsvarande C-program!*

Exempel 1 a) - Filen *main.cpp:*

```
/*****************************************************************************
* main.cpp: Implementering av en enkel maskininlärningsmodell baserad på
*           linjär regression, med träningsdata definierat direkt i funktionen
*           main och lagrat via två vektorer. Träningsdatan kan ändras utefter
*           behov, både via fler uppsättningar eller via helt ny data.
*
*           I Windows, kompilera programkoden och skapa en körbar fil döpt
*           main.exe via följande kommando:
*           $ g++ main.cpp lin_reg.cpp -o main.exe -Wall
*
*           Programmet kan sedan köras under 10 000 epoker med en lärhastighet
*           på 1 % via följande kommando:
*           $ main.exe
*
*           För att mata in antalet epoker samt lärhastighet som ska användas
*           vid träning kan följande kommando användas:
*           $ main.exe <num_epochs> <learning_rate>
*
*           Som exempel, för att genomföra träning under 5000 epoker med en
*           lärhastighet på 2 % kan följande kommando användas:
*           $ main.exe 5000 0.02
*****************************************************************************/
#include "lin_reg.hpp"

/*****************************************************************************
* main: Tränar en maskininlärningsmodell baserad på linjär regression via
*       träningsdata bestående av fem träningsuppsättningar, lagrade via var
*       sin vektor. Modellen tränas som default under 10 000 epoker med en
*       lärhastighet på 1 %. Dessa parametrar kan dock väljas av användaren
*       via inmatning i samband med körning av programmet, vilket läses in
*       via ingående argument argc samt argv.
*
*       Efter träningen är slutförd sker prediktion för samtliga insignaler
*       mellan -10 och 10 med en stegringshastighet på 1.0. Varje insignal
*       i detta intervall skrivs ut i terminalen tillsammans med predikterad
*       utsignal.
*
*       - argc: Antalet argument som har matats in vid körning av programmet
*               (default = 1, vilket är kommandot för att köra programmet).
*       - argc: Pekare till array innehållande samtliga inlästa argument i
*               form av text (default = exekveringskommandot, exempelvis main).
*****************************************************************************/
int main(const int argc,
         const char** argv)
{
   lin_reg l1;
   const std::vector<double> train_in = { 0, 1, 2, 3, 4 };
   const std::vector<double> train_out = { 2, 12, 22, 32, 42 };

   std::size_t num_epochs = 10000;
   double learning_rate = 0.2;

   if (argc == 3)
   {
      num_epochs = std::atoi(argv[1]);
      learning_rate = std::atof(argv[2]);
   }

   l1.set_training_data(train_in, train_out);
   l1.train(num_epochs, learning_rate);
   l1.predict();
   return 0;
}
```

Exempel 1 a) - Filen *lin_reg.hpp:*

```
/*****************************************************************************
* lin_reg.hpp: Innehåller funktionalitet för enkel implementering av
*              maskininlärningsmodeller baserade på linjär regression via
*              strukten lin_reg.
*****************************************************************************/
#ifndef LIN_REG_HPP_
#define LIN_REG_HPP_

/* Inkluderingsdirektiv: */
#include <iostream>
#include <vector>

/*****************************************************************************
* lin_reg: Strukt för implementering av maskininlärningsmodeller baserade på
*          linjär regression. Träningsdata passeras via referenser till vektorer
*          innehållande träningsuppsättningarnas in- och utdata. Träning
*          genomförs under angivet antal epoker med angiven lärhastighet.
*****************************************************************************/
struct lin_reg
{
   /* Medlemmar: */
   std::vector<double> train_in;         /* Indata för träningsuppsättningar. */
   std::vector<double> train_out;        /* Utdata för träningsuppsättningar. */
   std::vector<std::size_t> train_order; /* Lagrar ordningsföljd vid träning. */
   double bias = get_random();           /* Vilovärde (m-värde). */
   double weight = get_random();         /* Vikt (k-värde). */

   /* Medlemsfunktioner: */
   std::size_t num_sets(void) { return this->train_order.size(); }
   void set_training_data(const std::vector<double>& train_in,
                          const std::vector<double>& train_out);
   void train(const std::size_t num_epochs,
              const double learning_rate);
   double predict(const double input) { return this->weight * input + this->bias; }
   void predict(std::ostream& ostream = std::cout);
   void predict_range(const double min,
                      const double max,
                      const double step = 1.0,
                      std::ostream& ostream = std::cout);
private:
   double get_random(void) { return std::rand() / static_cast<double>(RAND_MAX); }
   void shuffle(void);
   void optimize(const double input,
                 const double reference,
                 const double learning_rate);
};

#endif /* LIN_REG_HPP_ */
```

Exempel 1 a) - Filen *lin_reg.cpp:*

```
/*****************************************************************************
* lin_reg.cpp: Definition av funktionsmedlemmar tillhörande strukten lin_reg,
*              som används för implementering av enkla maskininlärningsmodeller
*              som baseras på linjär regression.
*****************************************************************************/
#include "lin_reg.hpp"

/*****************************************************************************
* set_training_data: Läser in träningsdata för angiven regressionsmodell via
*                    passerad in- och utdata, tillsammans med att index
*                    för respektive träningsuppsättning lagras.
*
*                    - train_in : Innehåller indata för träningsuppsättningar.
*                    - train_out: Innehåller utdata för träningsuppsättningar.
*****************************************************************************/
void lin_reg::set_training_data(const std::vector<double>& train_in,
                                const std::vector<double>& train_out)
{
   const auto num_sets = train_in.size() <= train_out.size() ? train_in.size() : train_out.size();
   this->train_in.resize(num_sets);
   this->train_out.resize(num_sets);
   this->train_order.resize(num_sets);

   for (std::size_t i = 0; i < num_sets; ++i)
   {
      this->train_in[i] = train_in[i];
      this->train_out[i] = train_out[i];
      this->train_order[i] = i;
   }

   return;
}

/*****************************************************************************
* train: Tränar angiven regressionsmodell med befintlig träningsdata under
*        angivet antal epoker samt angiven lärhastighet. I början av varje epok
*        randomiseras ordningen på träningsuppsättningarna för att undvika att
*        modellen vänjer sig för mycket vid träningsdatan.
*
*        - num_epochs   : Antalet epoker/omgångar som träning ska genomföras.
*        - learning_rate: Lärhastigheten, som avgör hur stor andel av uppmätt
*                         avvikelse som modellens parametrar justeras med.
*****************************************************************************/
void lin_reg::train(const std::size_t num_epochs,
                    const double learning_rate)
{
   if (!this->num_sets())
   {
      std::cerr << "Training data missing!\n\n";
      return;
   }

   for (std::size_t i = 0; i < num_epochs; ++i)
   {
      this->shuffle();

      for (auto& j : this->train_order)
      {
         this->optimize(this->train_in[j], this->train_out[j], learning_rate);
      }
   }
   return;
}
```

```
/*****************************************************************************
* predict: Genomför prediktion med angiven regressionsmodell via indata från
*          samtliga befintliga träningsuppsättningar och skriver ut varje
*          insignal samt motsvarande predikterat värde via angiven utström
*          där standardutenheten std::cout används som default för utskrift
*          i terminalen.
*
*          - ostream: Angiven utström (default = std::cout).
*****************************************************************************/
void lin_reg::predict(std::ostream& ostream)
{
   if (!this->num_sets())
   {
      std::cerr << "Training data missing!\n\n";
      return;
   }

   constexpr auto threshold = 0.01;
   const auto* end = &this->train_in[this->train_in.size() - 1];

   ostream << "-------------------------------------------------------------------------------\n";

   for (auto& i : this->train_in)
   {
      const auto prediction = this->predict(i);

      ostream << "Input: " << i << "\n";

      if (prediction > -threshold && prediction < threshold)
      {
         ostream << "Predicted output: " << 0.0 << "\n";
      }
      else
      {
         ostream << "Predicted output: " << prediction << "\n";
      }

      if (&i < end) ostream << "\n";
   }

   ostream << "-------------------------------------------------------------------------------\n\n";
   return;
}
```

```
/*****************************************************************************
* predict_range: Genomför prediktion med angiven regressionsmodell för
*                datapunkter inom intervallet mellan angivet min- och maxvärde
*                [min, max] med angiven stegringshastighet step, som sätts till
*                1.0 som default.
*
*                Varje insignal skrivs ut tillsammans med motsvarande
*                predikterat värde via angiven utström, där standardutenheten
*                std::cout används som default för utskrift i terminalen.
*
*                - min    : Lägsta värde för datatpunkter som ska testas.
*                - max    : Högsta värde för datatpunkter som ska testas.
*                - step   : Stegringshastigheten, dvs. differensen mellan
*                           varje datapunkt som ska testas (default = 1.0).
*                - ostream: Angiven utström (default = std::cout).
*****************************************************************************/
void lin_reg::predict_range(const double min,
                            const double max,
                            const double step,
                            std::ostream& ostream)
{
   if (min >= max)
   {
      std::cerr << "Error: Minimum input value cannot be higher or equal to maximum input value!\n\n";
      return;
   }

   constexpr auto threshold = 0.01;
   ostream << "-------------------------------------------------------------------------------\n";

   for (auto i = min; i <= max; i = i + step)
   {
      const auto prediction = this->predict(i);
      ostream << "Input: " << i << "\n";

      if (prediction > -threshold && prediction < threshold)
      {
         ostream << "Predicted output: " << 0.0 << "\n";
      }
      else
      {
         ostream << "Predicted output: " << prediction << "\n";
      }

      if (i < max) ostream << "\n";
   }

   ostream << "-------------------------------------------------------------------------------\n\n";
   return;
}
```

```
/*****************************************************************************
* shuffle: Randomiserar den inbördes ordningen på träningsuppsättningarna för
*          angiven regressionsmodell, vilket genomförs i syfte att minska risken
*          för att eventuella icke avsedda mönster i träningsdatan ska
*          påverka träningen.
*****************************************************************************/
void lin_reg::shuffle(void)
{
   for (std::size_t i = 0; i < this->num_sets(); ++i)
   {
      const auto r = std::rand() % this->num_sets();
      const auto temp = this->train_order[i];
      this->train_order[i] = this->train_order[r];
      this->train_order[r] = temp;
   }

   return;
}

/*****************************************************************************
* optimize: Beräknar aktuell avvikelse för angiven regressionsmodell och
*           justerar modellens parametrar därefter.
*
*           input        : Insignal som prediktion ska genomföras med.
*           reference    : Referensvärde från träningsdatan, vilket utgör det
*                          värde som modellen önskas prediktera.
*           learning_rate: Modellens lärhastighet, avgör hur mycket modellens
*                          parametrar justeras vid avvikelse.
*****************************************************************************/
void lin_reg::optimize(const double input,
                       const double reference,
                       const double learning_rate)
{
   const auto prediction = this->predict(input);
   const auto error = reference - prediction;
   const auto change_rate = error * learning_rate;

   this->bias += change_rate;
   this->weight += change_rate * input;
   return;
}
```

1.1.7 – Exempel 2 a) - Linjär regression med inläsning av träningsdata i C++

- I detta andra exempel demonstreras en något mer avancerad implementering av en modell för linjär regression via en klass döpt *lin_reg*. I detta fall används tio träningsuppsättningar definierade enligt formeln y = -2.5x + 10, som läses in från en textfil döpt *data.txt.* Träningsuppsättningarna visas i tabell 5 nedan.

x	y
-5	-22.5
-4	-20
-3	-17.5
-2	-15
-1	-12.5
0	-10
1	-7.5
2	-5
3	-2.5
4	0

Tabell 5 – Träningsuppsättningar i exempel 2 a).

- Träning sker återigen under 1000 epoker med en lärhastighet på 1 %, följt av att modellen testas, i detta fall för samtliga flyttal mellan -10 och 10 i intervall om 0.5, där resultatet skrivs ut i terminalen. Träningen har återigen lyckats väl och modellen predikterar med 100 % precision.

- *Se bilaga B - Exempel 2 b) för motsvarande C-program!*

Exempel 2 a) - Filen *main.cpp:*

```
/*****************************************************************************************
* main.cpp: Implementerar en modell som bygger på linjär regression via ett objekt av klassen
*           lin_reg. Träningsdata läses in från en textfil. Efter träningen har
*           slutförts så genomförs prediktion av alla indata inom ett angivet intervall,
*           vilket skrivs ut i terminalen.
*
*           I Windows, kompilera koden och skapa en körbar fil main.exe med följande kommando:
*           $ g++ main.cpp lin_reg.cpp -o main.exe -Wall
*
*           Kör sedan programmet med följande kommando:
*           $ main.exe
*****************************************************************************************/
#include "lin_reg.hpp"

/*****************************************************************************************
* main: Implementerar en maskininlärningsmodell som baseras på linjär regression, där träningsdata
*       läses in från en fil döpt data.txt. Regressionsmodellen tränas under 1000 epoker med en
*       lärhastighet på 1 %. Modellen testas sedan för indata inom intervallet [-10, 10] med
*       en stegringshastighet på 0.5. Indata samt motsvarande predikterad utdata skrivs ut i
*       terminalen. Resultatet indikerar att modellen efter träning predikterar med en precision
*       på 100 %, vilket innebär att träningen var lyckad.
*****************************************************************************************/
int main(void)
{
   lin_reg l1(1000, 0.01);
   l1.load_training_data("data.txt");
   l1.train();
   l1.predict_range(-10, 10, 0.5);
   return 0;
}
```

Exempel 2 a) - Filen *lin_reg.hpp:*

```
/*****************************************************************************************
* lin_reg.hpp: Innehåller funktionalitet för implementering av maskininlärningsmodeller som
*              baseras på linjär regression via klassen lin_reg.
*****************************************************************************************/
#ifndef LIN_REG_HPP_
#define LIN_REG_HPP_

/* Inkluderingsdirektiv: */
#include <iostream>
#include <vector>
#include <string>
#include <fstream>

/*****************************************************************************************
* lin_reg: Klass för implementering av maskininlärningsmodeller som baseras på linjär regression.
*          Träningsdata med valfritt antal träningsuppsättningar kan läsas in från en fil eller
*          passeras via referenser till vektorer.
*
*          Klassens kopieringskonstruktor samt tilldelningsoperator är raderade, vilket medför att
*          minnet för ett givet objekt ej kan kopieras till/från ett annat objekt. Klassens
*          förflyttningskonstruktor är dock implementerad, vilket medför att minnet för ett
*          givet objekt kan förflyttas till ett annat objekt via funktionen std::move.
*****************************************************************************************/
class lin_reg
{
protected:
   /* Medlemmar: */
   std::vector<double> m_train_in;          /* Indata för träningsuppsättningar */
   std::vector<double> m_train_out;         /* Utdata för träningsuppsättningarna. */
   std::vector<std::size_t> m_train_order;  /* Ordningsföljd för träningsuppsättningarna. */
   double m_weight = 0;                     /* Lutning (k-värde). */
   double m_bias = 0;                       /* Vilovärde (m-värde). */
   double m_learning_rate = 0;              /* Lärhastighet (avgör justeringsgrad vid fel). */
   std::size_t m_num_epochs = 0;            /* Antalet träningsomgångar. */

   /* Medlemsfunktioner: */
   void extract(const std::string& s);
   void shuffle(void);
   void optimize(const double input,
                 const double output);
public:
   lin_reg(void) { }
   lin_reg(const std::size_t num_epochs,
                    const double learning_rate);
   ~lin_reg(void) { }
   lin_reg(lin_reg&) = delete;
   lin_reg& operator = (lin_reg&) = delete;
   lin_reg(lin_reg&& source) noexcept;

   double weight(void) { return m_weight; }
   double bias(void) { return m_bias; }
   double learning_rate(void) { return m_learning_rate; }
   std::size_t epochs(void) { return m_num_epochs; }

   void set_epochs(const std::size_t num_epochs);
   void set_learning_rate(const double learning_rate);
   void load_training_data(const std::string& filepath);
   void set_training_data(const std::vector<double>& train_in,
                          const std::vector<double>& train_out);
```

```
    void train(void);
    double predict(const double input);
    void predict_all(const double threshold = 0.001,
                     std::ostream& ostream = std::cout);
    void predict_range(const double start_val,
                       const double end_val,
                       const double step = 1,
                       const double threshold = 0.001,
                       std::ostream& ostream = std::cout);
};

#endif /* LIN_REG_HPP_ */
```

Exempel 2 - Filen *lin_reg.cpp:*

```
/*****************************************************************************************************
* lin_reg.cpp: Innehåller medlemsfunktioner tillhörande klassen lin_reg, vilket används för
*              implementering av maskininlärningsmodeller som baseras på linjär regression.
*****************************************************************************************************/
#include "lin_reg.hpp"

/* Statiska funktioner: */
static bool char_is_digit(const char c);
static void retrieve_double(std::vector<double>& data,
                            std::string& s);

/*****************************************************************************************************
* extract: Extraherar träningsdata i form av flyttal ur angiven sträng och lagrar som träningsdata
*          för angiven regressionsmodell. Ifall två flyttal lyckas extraheras lagras dessa som en
*          träningsuppsättning via vektorer m_train_in samt m_train_out. Index för varje lagrad
*          träningsuppsättning lagras också via vektorn m_train_order för att enkelt kunna
*          randomisera ordningsföljden för träningsuppsättningarna vid träning utan att förflytta
*          träningsdatan, vilket genomförs för att minska risken att eventuella icke avsedda
*          mönster som förekommer i träningsdatan ska påverka träningen av regressionsmodellen.
*
*          - s: Sträng innehållande de flyttal som ska extraheras.
*****************************************************************************************************/
void lin_reg::extract(const std::string& s)
{
   std::string num_str;
   std::vector<double> data;

   for (auto& i : s)
   {
      if (char_is_digit(i))
      {
         num_str += i;
      }
      else
      {
         retrieve_double(data, num_str);
      }
   }

   retrieve_double(data, num_str);

   if (data.size() == 2)
   {
      m_train_in.push_back(data[0]);
      m_train_out.push_back(data[1]);
      m_train_order.push_back(m_train_order.size());
   }
   return;
}
```

```
/*****************************************************************************************
* shuffle: Randomiserar den inbördes ordningsföljden för angiven regressionsmodells
*          träningsuppsättningar genom att förflytta innehållet i vektorn m_train_order, som
*          lagrar index för respektive träningsuppsättning.
*****************************************************************************************/
void lin_reg::shuffle(void)
{
   for (std::size_t i = 0; i < m_train_order.size(); ++i)
   {
      const auto r = rand() % m_train_order.size();
      const auto temp = m_train_order[i];
      m_train_order[i] = m_train_order[r];
      m_train_order[r] = temp;
   }
   return;
}

/*****************************************************************************************
* optimize: Justerar parametrar för angiven regressionsmodell i syfte att minska aktuellt fel.
*           Prediktion genomförs via given insignal, där predikterat värde jämförs mot givet
*           referensvärde för att beräkna aktuellt fel. Modellens parametrar justeras sedan med en
*           bråkdel av felet, vilket avgörs av lärhastigheten.
*
*           För vikten (k-värdet) tas aktuell insignal i åtanke gällande graden av justering,
*           då viktens betydelse för aktuell fel står i direkt proportion med aktuell insignal
*           (ju högre insignal, desto mer påverkan har vikten på predikterad utsignal och därmed
*           eventuellt fel).
*
*           - input    : Insignal från träningsdata som används för att genomföra prediktion.
*           - reference: Referensvärde från träningsdatan, som används för att beräkna aktuellt
*                        fel via jämförelse med predikterat värde.
*****************************************************************************************/
void lin_reg::optimize(const double input,
                       const double reference)
{
   const auto prediction = m_weight * input + m_bias;
   const auto error = reference - prediction;
   const auto change_rate = error * m_learning_rate;
   m_bias += change_rate;
   m_weight += change_rate * input;
   return;
}

/*****************************************************************************************
* lin_reg: Konstruktor för klassen lin_reg, vilket används för att initiera
*                     en ny regressionsmodell som baseras på linjär regression. Angivet antal
*                     epoker samt lärhastighet lagras inför träning. Träningsdata måste dock
*                     tillföras i efterhand via någon av medlemsfunktioner load_training_data
*                     (för att läsa in träningsuppsättingarna från en fil) eller set_training_data
*                     (för att passera träningsdata via referenser till vektorer).
*
*                     - num_epochs   : Antalet epoker/omgångar som ska genomföras vid träning.
*                     - learning_rate: Lärhastighet, avgör med hur stor andel av aktuellt fel som
*                                      modellens parametrar (bias och vikt) ska justeras.
*****************************************************************************************/
lin_reg::lin_reg(const std::size_t num_epochs,
                 const double learning_rate)
{
   set_epochs(num_epochs);
   set_learning_rate(learning_rate);
   return;
}
```

```
/*****************************************************************************************
* lin_reg: Förflyttningskonstruktor, som medför förflyttning av minne från en
*                    regressionsmodell till en annan, i detta fall från source till angivet
*                    objekt this via anrop av funktionen std::move.
*
*                    Innehållet lagrat av regressionsmodellen source kopieras till angiven modell
*                    this, följt av att source nollställs. Efter förflyttningen har därmed enbart
*                    angiven modell this tillgång till minnet i fråga.
*
*                    - source: Den regressionsmodell som minnet ska förflyttas från.
*****************************************************************************************/
lin_reg::lin_reg(lin_reg&& source) noexcept
{
   this->m_train_in = source.m_train_in;
   this->m_train_out = source.m_train_out;
   this->m_train_order = source.m_train_order;
   this->m_weight = source.m_weight;
   this->m_bias = source.m_bias;
   this->m_learning_rate = source.m_learning_rate;
   this->m_num_epochs = source.m_num_epochs;

   source.m_train_in.clear();
   source.m_train_out.clear();
   source.m_train_order.clear();
   source.m_weight = 0;
   source.m_bias = 0;
   source.m_learning_rate = 0;
   source.m_num_epochs = 0;
   return;
}

/*****************************************************************************************
* set_epochs: Uppdaterar antalet epoker som sker vid träning för angiven regressionsmodell ifall
*             angivet nytt antal överstiger noll.
*
*             - num_epochs: Det nya antalet epoker som ska genomföras vid träning.
*****************************************************************************************/
void lin_reg::set_epochs(const std::size_t num_epochs)
{
   if (num_epochs > 0)
   {
      m_num_epochs = num_epochs;
   }
   return;
}

/*****************************************************************************************
* set_learning_rate: Sätter ny lärhastighet för att justera parametrarna för angiven
*                    regressionsmodell ifall angivet nytt värde överstiger noll.
*
*                    - learning_rate: Den nya lärhastighet som ska användas för att justera
*                                     modellens parametrar (bias och vikt) vid fel.
*****************************************************************************************/
void lin_reg::set_learning_rate(const double learning_rate)
{
   if (learning_rate > 0)
   {
      m_learning_rate = learning_rate;
   }
   return;
}
```

```
/*****************************************************************************************
* load_training_data: Läser in träningsdata från en fil via angiven filsökväg, extraherar denna
*                     data i form av flyttal och lagrar som träningsuppsättningar för angiven
*                     regressionsmodell.
*
*                     - filepath: Filsökvägen som träningsdatan ska läsas från.
*****************************************************************************************/
void lin_reg::load_training_data(const std::string& filepath)
{
   std::ifstream fstream(filepath, std::ios::in);

   if (!fstream)
   {
      std::cerr << "Could not open file at path " << filepath << "!\n\n";
   }
   else
   {
      std::string s;
      while (std::getline(fstream, s))
      {
         extract(s);
      }
   }
   return;
}


/*****************************************************************************************
* set_training_data: Kopierar träningsdata till angiven regressionsmodell från refererade vektorer
*                    samt lagrar index för respektive träningsuppsättning. Enbart fullständiga
*                    träningsuppsättningar där både in- och utsignal förekommer lagras.
*
*                    - train_in : Innehåller insignaler för samtliga träningsuppsättningar.
*                    - train_out: Innehåller utsignaler för samtliga träningsuppsättningar.
*****************************************************************************************/
void lin_reg::set_training_data(const std::vector<double>& train_in,
                                const std::vector<double>& train_out)
{
   auto num_sets = train_in.size();

   if (train_in.size() > train_out.size())
   {
      num_sets = train_out.size();
   }

   m_train_in.resize(num_sets);
   m_train_out.resize(num_sets);
   m_train_order.resize(num_sets);

   for (std::size_t i = 0; i < num_sets; ++i)
   {
      m_train_in[i] = train_in[i];
      m_train_out[i] = train_out[i];
      m_train_order[i] = i;
   }
   return;
}
```

```
/*****************************************************************************************
* train: Tränar angiven regressionsmodell under angivet antal epoker. Inför varje ny epok
*        randomiseras ordningsföljden på träningsuppsättningarna för att undvika att eventuella
*        mönster som uppträder i träningsdatan ska påverka träningen av modellen.
*
*        Varje varv optimeras modellens parametrar genom att en prediktion genomförs via en
*        insignal från träningsdatan, där det predikterade värdet jämförs med referensvärdet
*        från träningsdatan. Differensen mellan dessa värden utgör aktuellt fel och parametrarna
*        justeras med en bråkdel av detta värde, beroende på aktuell lärhastighet.
*****************************************************************************************/
void lin_reg::train(void)
{
   for (std::size_t i = 0; i < m_num_epochs; ++i)
   {
      shuffle();

      for (auto& j : m_train_order)
      {
         optimize(m_train_in[j], m_train_out[j]);
      }
   }
   return;
}

/*****************************************************************************************
* predict: Genomför prediktion med angiven regressionsmodell via angiven insignal och returnerar
*          motsvarande predikterad utsignal i form av ett flytta.
*
*          - input: Den insignal som prediktion ska genomföras på.
*****************************************************************************************/
double lin_reg::predict(const double input)
{
   return m_weight * input + m_bias;
}
```

```
/*****************************************************************************************
* predict_all: Genomför prediktion med angiven regressionsmodell för samtliga insignaler som
*              förekommer i träningsdatan och skriver ut motsvarande predikterade utsignaler via
*              angiven utström, där standardutenhet std::cout används som default för utskrift i
*              i terminalen. Värden mycket nära noll [-threshold, threshold] avrundas till noll
*              för att undvika utskrift med ett flertal decimaler i onödan, vilket annars sker
*              just runt noll.
*
*              - threshold: Tröskelvärde runt nollpunkten [-threshold, threshold], där
*                           predikterad värde ska avrundas till noll (default = 0.001).
*              - ostream  : Angiven utström (default = std::cout).
*****************************************************************************************/
void lin_reg::predict_all(const double threshold,
                          std::ostream& ostream)
{
   const auto* last = &m_train_in[m_train_in.size() - 1];
   ostream << "-------------------------------------------------------------------------\n";

   for (auto& i : m_train_in)
   {
      const auto prediction = predict(i);
      ostream << "Input: " << i << "\n";

      if (prediction > -threshold && prediction < threshold)
      {
         ostream << "Output: " << 0 << "\n";
      }
      else
      {
         ostream << "Output: " << predict(i) << "\n";
      }

      if (&i < last) ostream << "\n";
   }

   ostream << "-------------------------------------------------------------------------\n\n";
   return;
}
```

```
/*****************************************************************************************
* predict_range: Genomför prediktion med angiven regressionsmodell för insignaler inom intervallet
*                mellan angivet start- och slutvärde [start_val, end_val], där inkrementering av
*                insignalen sker inom detta intervall med angivet stegvärde step.
*
*                Varje insignal samt motsvarande predikterat värde skrivs ut via angiven utström,
*                där standardutenheten std::cout används som default för utskrift i terminalen.
*                Värden mycket nära noll [-threshold, threshold] avrundas till noll för att
*                undvika utskrift med ett flertal decimaler i onödan, vilket annars sker just
*                runt noll.
*
*                - start_val: Minvärde för det intervall av insignaler som ska testas.
*                - end_val  : Maxvärde för det intervall av insignaler som ska testas.
*                - step     : Stegvärde/inkrementeringsvärde för insignaler (default = 1.0).
*                - threshold: Tröskelvärde, där samtliga predikterade värden som ligger inom
*                             intervallet [-threshold, threshold] avrundas till noll för att
*                             undvika utskrift med onödigt antal decimaler (default = 0.001).
*                - ostream:   Angiven utström (default = std::cout).
*****************************************************************************************/
void lin_reg::predict_range(const double start_val,
                            const double end_val,
                            const double step,
                            const double threshold,
                            std::ostream& ostream)
{
   ostream << "-------------------------------------------------------------------------\n";

   for (double i = start_val; i <= end_val; i += step)
   {
      const auto prediction = predict(i);
      ostream << "Input: " << i << "\n";

      if (prediction > -threshold && prediction < threshold)
      {
         ostream << "Output: " << 0 << "\n";
      }
      else
      {
         ostream << "Output: " << predict(i) << "\n";
      }

      if (i < end_val) ostream << "\n";
   }

   ostream << "-------------------------------------------------------------------------\n\n";
   return;
}

/*****************************************************************************************
* char_is_digit: Indikerar ifall givet tecken utgör en siffra eller ett relaterat tecken, såsom
*                ett minustecken eller en punkt. Eftersom flyttal ibland matas in både med
*                punkt samt kommatecken så utgör båda giltiga tecken.
*
*                - c: Det tecken som ska kontrolleras.
*****************************************************************************************/
static bool char_is_digit(const char c)
{
   const auto s = "0123456789-.,";
   for (auto i = s; *i; ++i)
   {
      if (c == *i)
      {
         return true;
      }
   }
   return false;
}
```

```
/*****************************************************************************************
* retrieve_double: Typomvandlar innehåll lagrat som text till ett flyttal och lagrar detta i
*                  angiven vektor. Innan typomvandlingen äger rum ersätts eventuella kommatecken
*                  med punkt, vilket möjliggör att flyttal kan läsas in både med punkt eller
*                  kommatecken som decimaltecken.
*
*                  - data: Den vektor som typomvandlat flyttal ska lagras i.
*                  - s   : Data i form av text som ska typomvandlas till ett flyttal.
*****************************************************************************************/
static void retrieve_double(std::vector<double>& data,
                            std::string& s)
{
   for (auto& i : s)
   {
      if (i == ',') i = '.';
   }

   try
   {
      const auto number = std::stod(s);
      data.push_back(number);
   }
   catch (std::invalid_argument&)
   {
      std::cerr << "Failed to convert " << s << " to double\n";
   }

   s.clear();
   return;
}
```

Bilaga A

Exempel 1 b) Enkel implementering av linjär regression i C

- Detta exempel utgör en C-version av den enkla regressionsmodell som demonstrerades i exempel 1 a). Även i detta fall används en strukt döpt *lin_reg.* Eftersom struktar i C inte kan innehålla medlemsfunktioner så används i stället ett flertal externa funktioner för att realisera modellen. I stället för dynamiska vektorer av klassen *std::vector* används statiska och dynamiska arrayer.

- Som i föregående fall används nedanstående fem träningsuppsättningar definierade enligt formeln y = 10x + 2 för att träna modellen, i detta fall lagrade via två statiska arrayer. Träningsuppsättningarna visas i tabell 6 nedan.

x	y
0	2
1	12
2	22
3	32
4	42

Tabell 6 – Träningsuppsättningar i Exempel 1 b).

- Även här sker träning som default under 10 000 epoker med en lärhastighet på 1 %, men det är möjligt att välja dessa parametrar via ingående argument när programmet körs. Efter att träning har genomförts sker prediktion för insignaler bestående av alla heltal inom intervallet [-10, 10]. Samtliga insignaler x samt predikterade utsignaler y skrivs sedan ut i terminalen. Träningen har lyckats väl och modellen predikterar med 100 % precision.

Exempel 1 b) - Filen *main.c:*

```
/*****************************************************************************
* main.c: Implementering av en enkel maskininlärningsmodell baserad på linjär
*         regression, med träningsdata definierat direkt i funktionen main.
*
*         I Windows, kompilera programkoden och skapa en körbar fil döpt
*         main.exe via följande kommando:
*         $ gcc main.c lin_reg.c -o main.exe -Wall
*
*         Programmet kan sedan köras under 10 000 epoker med en lärhastighet
*         på 1 % via följande kommando:
*         $ main.exe
*
*         För att mata in antalet epoker samt lärhastighet som ska användas
*         vid träning kan följande kommando användas:
*         $ main.exe <num_epochs> <learning_rate>
*
*         Som exempel, för att genomföra träning under 5000 epoker med en
*         lärhastighet på 2 % kan följande kommando användas:
*         $ main.exe 5000 0.02
*****************************************************************************/
#include "lin_reg.h"

/*****************************************************************************
* main: Tränar en maskininlärningsmodell baserad på linjär regression via
*       träningsdata bestående av fem träningsuppsättningar, lagrade via var
*       sin vektor. Modellen tränas som default under 10 000 epoker med en
*       lärhastighet på 1 %. Dessa parametrar kan dock väljas av användaren
*       via inmatning i samband med körning av programmet, vilket läses in
*       via ingående argument argc samt argv.
*
*       Efter träningen är slutförd sker prediktion för samtliga insignaler
*       mellan -10 och 10 med en stegringshastighet på 1.0. Varje insignal
*       i detta intervall skrivs ut i terminalen tillsammans med predikterad
*       utsignal.
*
*       - argc: Antalet argument som har matats in vid körning av programmet
*               (default = 1, vilket är kommandot för att köra programmet).
*       - argc: Pekare till array innehållande samtliga inlästa argument i
*               form av text (default = exekveringskommandot, exempelvis main).
*****************************************************************************/
int main(const int argc,
         const char** argv)
{
   struct lin_reg l1;

   const double train_in[] = { 0, 1, 2, 3, 4 };
   const double train_out[] = { 2, 12, 22, 32, 42 };

   size_t num_epochs = 10000;
   double learning_rate = 0.01;

   if (argc == 3)
   {
      num_epochs = (size_t)atoi(argv[2]);
      learning_rate = atof(argv[3]);
   }

   lin_reg_new(&l1);
   lin_reg_set_training_data(&l1, train_in, train_out, 5);
   lin_reg_train(&l1, num_epochs, learning_rate);

   lin_reg_predict_range(&l1, -10, 10, 1, stdout);
   return 0;
}
```

Exempel 1 b) - Filen *lin_reg.h:*

```
/*****************************************************************************
* lin_reg.h: Innehåller funktionalitet för enkel implementering av
*            maskininlärningsmodeller baserade på linjär regression via
*            strukten lin_reg samt tillhörande externa funktioner:
*****************************************************************************/
#ifndef LIN_REG_H_
#define LIN_REG_H_

/* Inkluderingsdirektiv: */
#include <stdio.h>
#include <stdlib.h>

/*****************************************************************************
* lin_reg: Strukt för implementering av maskininlärningsmodeller baserade på
*          linjär regression. Träningsdata passeras via pekare till arrayer
*          innehållande träningsuppsättningarnas in- och utdata.
*****************************************************************************/
struct lin_reg
{
   const double* train_in;  /* Indata för träningsuppsättningar. */
   const double* train_out; /* Utdata för träningsuppsättningar. */
   size_t* train_order;     /* Lagrar ordningsföljden vid träning. */
   size_t num_sets;         /* Antalet befintliga träningsuppsättningar. */
   double bias;             /* Vilovärde (m-värde). */
   double weight;           /* Lutning (k-värde). */
};

/* Externa funktioner: */
void lin_reg_new(struct lin_reg* self);
void lin_reg_delete(struct lin_reg* self);
struct lin_reg* lin_reg_ptr_new(void);
void lin_reg_ptr_delete(struct lin_reg** self);
int lin_reg_set_training_data(struct lin_reg* self,
                              const double* train_in,
                              const double* train_out,
                              const size_t num_sets);
void lin_reg_train(struct lin_reg* self,
                   const size_t num_epochs,
                   const double learning_rate);
double lin_reg_predict(const struct lin_reg* self,
                       const double input);
void lin_reg_predict_train_in(const struct lin_reg* self,
                              FILE* ostream);
void lin_reg_predict_range(const struct lin_reg* self,
                           const double min,
                           const double max,
                           const double step,
                           FILE* ostream);

#endif /* LIN_REG_H_ */
```

Exempel 1 c) - Filen *lin_reg.c:*

```
/*****************************************************************************
* lin_reg.c: Definition av externa funktioner för ämnade för strukten lin_reg,
*            som används för implementering av enkla maskininlärningsmodeller
*            som baseras på linjär regression.
*****************************************************************************/
#include "lin_reg.h"

/* Statiska funktioner: */
static void lin_reg_shuffle(struct lin_reg* self);
static void lin_reg_optimize(struct lin_reg* self,
                             const double input,
                             const double reference,
                             const double learning_rate);
static double get_random(void);
static inline size_t* uint_ptr_new(size_t size);

/*****************************************************************************
* lin_reg_new: Initierar angiven regressionsmodell. Modellens bias och vikt
*              tilldelas randomiserade startvärden mellan 0.0 - 1.0.
*
*              - self: Pekare till regressionsmodellen.
*****************************************************************************/
void lin_reg_new(struct lin_reg* self)
{
   self->train_in = 0;
   self->train_out = 0;
   self->train_order = 0;
   self->num_sets = 0;
   self->bias = get_random();
   self->weight = get_random();
   return;
}

/*****************************************************************************
* lin_reg_delete: Nollställer angiven regressionsmodell.
*
*                 - self: Pekare till regressionsmodellen.
*****************************************************************************/
void lin_reg_delete(struct lin_reg* self)
{
   self->train_in = 0;
   self->train_out = 0;

   free(self->train_order);
   self->train_order = 0;

   self->num_sets = 0;
   self->bias = 0;
   self->weight = 0;
   return;
}

/*****************************************************************************
* lin_reg_ptr_new: Returnerar pekare till en ny heapallokerad regressionsmodell.
*                  Modellens bias och vikt tilldelas randomiserade startvärden
*                  mellan 0.0 - 1.0.
*****************************************************************************/
struct lin_reg* lin_reg_ptr_new(void)
{
   struct lin_reg* self = (struct lin_reg*)malloc(sizeof(struct lin_reg));
   if (!self) return 0;
   lin_reg_new(self);
   return self;
}
```

```
/*****************************************************************************
* lin_reg_ptr_delete: Raderar heapallokerad regressionsmodell och sätter
*                     motsvarande pekare till null.
*
*                     - self: Adressen till pekaren som pekar på den
*                             heapallokerade regressionsmodellen.
*****************************************************************************/
void lin_reg_ptr_delete(struct lin_reg** self)
{
   lin_reg_delete(*self);
   free(*self);
   *self = 0;
   return;
}

/*****************************************************************************
* lin_reg_set_training_data: Läser in träningsdata för angiven regressionsmodell
*                            via passerad in- och utdata, tillsammans med att
*                            index för respektive träningsuppsättning lagras.
*
*                            - self     : Pekare till regressionsmodellen.
*                            - train_in : Pekare till array innehållande indata.
*                            - train_out: Pekare till array innehållande utdata.
*                            - num_sets : Antalet träningsuppsättningar som
*                                         förekommer i passerad träningsdata.
*****************************************************************************/
int lin_reg_set_training_data(struct lin_reg* self,
                              const double* train_in,
                              const double* train_out,
                              const size_t num_sets)
{
   self->train_in = train_in;
   self->train_out = train_out;
   self->train_order = uint_ptr_new(num_sets);

   if (!self->train_order)
   {
      self->num_sets = 0;
      return 1;
   }
   else
   {
      self->num_sets = num_sets;

      for (size_t i = 0; i < self->num_sets; ++i)
      {
         self->train_order[i] = i;
      }

      return 0;
   }
}
```

```
/*****************************************************************************
* lin_reg_train: Tränar angiven regressionsmodell med befintlig träningsdata
*                under angivet antal epoker samt angiven lärhastighet. I början
*                av varje epok randomiseras ordningen på träningsuppsättningarna
*                för att undvika modellen blir för bekant med träningsdatan.
*
*                För varje träningsuppsättning sker en prediktion via aktuell
*                indata. Det predikterade värdet jämförs mot aktuellt
*                referensvärde för att beräkna aktuell avvikelse. Modellens
*                parametrar justeras därefter.
*
*                - self         : Pekare till regressionsmodellen.
*                - num_epochs   : Antalet omgångar träning som ska genomföras.
*                - learning_rate: Lärhastigheten, som avgör hur stor andel av
*                                 uppmätt avvikelse som modellens parametrar
*                                 justeras med.
*****************************************************************************/
void lin_reg_train(struct lin_reg* self,
                   const size_t num_epochs,
                   const double learning_rate)
{
   for (size_t i = 0; i < num_epochs; ++i)
   {
      lin_reg_shuffle(self);

      for (size_t j = 0; j < self->num_sets; ++j)
      {
         const size_t k = self->train_order[j];
         lin_reg_optimize(self, self->train_in[k], self->train_out[k], learning_rate);
      }
   }

   return;
}


/*****************************************************************************
* lin_reg_predict: Genomför prediktion med angiven regressionsmodell för
*                  angiven insignal och returnerar resultatet.
*
*                  - self : Pekare till regressionsmodellen.
*                  - input: Insignal som prediktion ska genomföras utefter.
*****************************************************************************/
double lin_reg_predict(const struct lin_reg* self,
                       const double input)
{
   return self->weight * input + self->bias;
}
```

```c
/*****************************************************************************
* lin_reg_predict_train_in: Genomför prediktion med angiven regressionsmodell
*                           via indata från befintliga träningsuppsättningar.
*                           Varje insignal samt motsvarande predikterad utsignal
*                           skrivs ut via angiven utström, där standardutenheten
*                           stdout används som default för utskrift i terminalen.
*
*                           - self   : Pekare till regressionsmodellen.
*                           - ostream: Pekare till utström för uskrift
*                                      (default = stdout).
*****************************************************************************/
void lin_reg_predict_train_in(const struct lin_reg* self,
                              FILE* ostream)
{
   const double threshold = 0.01;
   const double* end = 0;

   if (!self->num_sets)
   {
      fprintf(stderr, "Training data missing!\n\n");
      return;
   }

   end = self->train_in + self->num_sets - 1;
   if (!ostream) ostream = stdout;

   fprintf(ostream, "-------------------------------------------------------------------------------\n");

   for (const double* i = self->train_in; i < self->train_in + self->num_sets; ++i)
   {
      const double prediction = self->weight * (*i) + self->bias;

      fprintf(ostream, "Input: %g\n", *i);

      if (prediction > -threshold && prediction < threshold)
      {
         fprintf(ostream, "Predicted output: %g\n", 0.0);
      }
      else
      {
         fprintf(ostream, "Predicted output: %g\n", prediction);
      }

      if (i < end) fprintf(ostream, "\n");
   }

   fprintf(ostream, "-------------------------------------------------------------------------------\n\n");
   return;
}
```

```c
/*****************************************************************************
* lin_reg_predict_range: Genomför prediktion med angiven regressionsmodell för
*                        datapunkter inom intervallet mellan angivet min- och
*                        maxvärde [min, max] med angiven stegringshastighet.
*
*                        Varje insignal inom intervallet skrivs ut tillsammans
*                        med motsvarande predikterat värde via angiven utström,
*                        där standardutenheten stdout används som default för
*                        utskrift i terminalen.
*
*                        - self   : Pekare till regressionsmodellen.
*                        - min    : Minvärde för datatpunkter som ska testas.
*                        - max    : Maxvärde för datatpunkter som ska testas.
*                        - step   : Stegringshastigheten, dvs. differensen mellan
*                                   varje datapunkt som ska testas.
*                        - ostream: Pekare till angiven utström för utskrift
*                                   (default = stdout).
*****************************************************************************/
void lin_reg_predict_range(const struct lin_reg* self,
                           const double min,
                           const double max,
                           const double step,
                           FILE* ostream)
{
   const double threshold = 0.01;

   if (min >= max)
   {
      fprintf(stderr, "Error: Minimum input value cannot be higher or equal to maximum input
value!\n\n");
      return;
   }

   if (!ostream) ostream = stdout;

   fprintf(ostream, "-------------------------------------------------------------------------\n");

   for (double i = min; i <= max; ++i)
   {
      const double prediction = self->weight * i + self->bias;

      fprintf(ostream, "Input: %g\n", i);

      if (prediction > -threshold && prediction < threshold)
      {
         fprintf(ostream, "Predicted output: %g\n", 0.0);
      }
      else
      {
         fprintf(ostream, "Predicted output: %g\n", prediction);
      }

      if (i < max) fprintf(ostream, "\n");
   }

   fprintf(ostream, "-------------------------------------------------------------------------\n\n");
   return;
}
```

```
/*****************************************************************************
* lin_reg_shuffle: Randomiserar den inbördes ordningsföljden på befintliga
*                  träningsuppsättningar för angiven regressionsmodell, vilket
*                  genomförs för att modellen inte ska bli för bekant med
*                  träningsdatan.
*
*                  - self: Pekare till regressionsmodellen.
*****************************************************************************/
static void lin_reg_shuffle(struct lin_reg* self)
{
   for (size_t i = 0; i < self->num_sets; ++i)
   {
      const size_t r = rand() % self->num_sets;
      const size_t temp = self->train_order[i];
      self->train_order[i] = self->train_order[r];
      self->train_order[r] = temp;
   }

   return;
}

/*****************************************************************************
* lin_reg_optimize: Beräknar aktuell avvikelse för angiven regressionsmodell
*                   och justerar modellens parametrar därefter.
*
*                   - self   : Pekare till regressionsmodellen.
*                   - input        : Insignal som prediktion ska genomföras med.
*                   - reference    : Referensvärde från träningsdatan, som utgör
*                                    det värde som modellen önskas prediktera.
*                   - learning_rate: Modellens lärhastighet, som avgör hur mycket
*                                    modellens parametrar justeras vid avvikelse.
*****************************************************************************/
static void lin_reg_optimize(struct lin_reg* self,
                             const double input,
                             const double reference,
                             const double learning_rate)
{
   const double prediction = self->weight * input + self->bias;
   const double deviation = reference - prediction;
   const double change_rate = deviation * learning_rate;

   self->bias += change_rate;
   self->weight += change_rate * input;
   return;
}

/*****************************************************************************
* get_random: Returnerar ett randomiserat flyttal mellan 0.0 - 1.0.
*****************************************************************************/
static double get_random(void)
{
   return rand() / (double)RAND_MAX;
}

/*****************************************************************************
* uint_ptr_new: Returnerar en pekare till ett heapallokerat fält som rymmer
*               angivet antal osignerade heltal.
*
*               - size: Storleken på det heapallokerade fältet, dvs. antalet
*                       osignerade heltal det ska rymma.
*****************************************************************************/
static inline size_t* uint_ptr_new(size_t size)
{
   return (size_t*)malloc(sizeof(size_t) * size);
}
```

Bilaga B

Exempel 2 b) Linjär regression med inläsning av träningsdata i C

- Detta exempel utgör en C-version av den mer avancerade regressionsmodell som demonstrerades i exempel 2 a). I detta fall används en strukt döpt *lin_reg* samt ett flertal externa funktioner för att realisera modellen.

- Dynamiska vektorer för flyttal samt osignerade tal implementeras via struktar *double_vector* samt *uint_vector,* tillsammans med ett flertal externa funktioner. I detta exempel används tio träningsuppsättningar definierade enligt formeln y = -5x + 0.5, se tabell 7 nedan.

x	y
-5	-24.5
-4	-19.5
-3	-14.5
-2	-9.5
-1	-4.5
0	0.5
1	5.5
2	10.5
3	15.5
4	20.5

Tabell 7 – Träningsuppsättningar för exempel i exempel 2 b).

- Träningsuppsättningarna läses in från en textfil döpt *data.txt,* följt av att träning sker under 1000 epoker med en lärhastighet på 1 %. Slutligen testat modellen för samtliga flyttal mellan -10 och 10 i intervall om 1, där resultatet skrivs ut i terminalen. Träningen har återigen lyckats väl och modellen predikterar med 100 % precision.

Exempel 2 b) - Filen *main.c:*

```
/*****************************************************************************************
* main.c: Implementerar en maskininlärningsmodell som baseras på linjär regression. Träningsdata
*         läses in från en textfil. Modellen tränas följt av att prediktion genomförs med 100 %
*         precision, vilket indikerar lyckad träning.
*
*         I Windows, kompilera koden och skapa en körbar fil main.exe med följande kommando:
*         $ gcc main.c lin_reg.c double_vector.c uint_vector.c -o main.exe -Wall
*
*         Kör sedan programmet med följande kommando:
*         $ main.exe
*****************************************************************************************/
#include "lin_reg.h"

/*****************************************************************************************
* main: Implementerar en regressionsmodell och läser in träningsdata från en fil döpt data.txt.
*       Modellen tränas under 1000 epoker med en lärhastighet på 1 %. Modellen testas sedan för
*       insignaler inom intervallet [-10, 10] med en stegringshastighet på 1, där indata samt
*       motsvarande predikterad utdata skrivs ut i terminalen. Resultatet indikerar prediktion
*       med 100 % precision.
*****************************************************************************************/
int main(void)
{
   struct lin_reg l1;
   lin_reg_new(&l1);
   lin_reg_load_training_data(&l1, "data.txt");
   lin_reg_train(&l1, 1000, 0.01);
   lin_reg_predict_range(&l1, -10, 10, 1, 0.0001, stdout);
   return 0;
}
```

Exempel 2 b) - Filen *lin_reg.h:*

```
/*****************************************************************************************
* lin_reg.h: Innehåller funktionalitet för implementering av maskininlärningsmodeller baserade på
*            linjär regression via strukten lin_reg samt externa funktioner.
*****************************************************************************************/
#ifndef LIN_REG_H_
#define LIN_REG_H_

/* Inkluderingsdirektiv: */
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "double_vector.h"
#include "uint_vector.h"

/*****************************************************************************************
* lin_reg: Strukt för implementering av maskininlärningsmodeller baserade på linjär regression.
*          Träningsdata bestående av valfritt antal träningsuppsättningar kan läsas in från en
*          fil eller passeras via pekare till arrayer.
*****************************************************************************************/
struct lin_reg
{
   struct double_vector train_in;  /* Träningsuppsättningarnas insignaler. */
   struct double_vector train_out; /* Träningsuppsättningarnas utsignaler. */
   struct uint_vector train_order; /* Lagrar träningsuppsättningarnas ordningsföljd. */
   double bias;                    /* Vilovärde (m-värde). */
   double weight;                  /* Lutning (k-värde). */
};

/* Externa funktioner: */
void lin_reg_new(struct lin_reg* self);
void lin_reg_delete(struct lin_reg* self);
struct lin_reg* lin_reg_ptr_new(void);
void lin_reg_ptr_delete(struct lin_reg** self);
void lin_reg_load_training_data(struct lin_reg* self,
                                const char* filepath);
void lin_reg_set_training_data(struct lin_reg* self,
                               const double* train_in,
                               const double* train_out,
                               const size_t num_sets);
void lin_reg_train(struct lin_reg* self,
                   const size_t num_epochs,
                   const double learning_rate);
double lin_reg_predict(const struct lin_reg* self,
                       const double input);
void lin_reg_predict_all(const struct lin_reg* self,
                         const double threshold,
                         FILE* ostream);
void lin_reg_predict_range(const struct lin_reg* self,
                           const double start_val,
                           const double end_val,
                           const double step,
                           const double threshold,
                           FILE* ostream);

#endif /* LIN_REG_H_ */
```

Exempel 2 b) - Filen *lin_reg.c:*

```
/*************************************************************************************************
* lin_reg.c: Innehåller externa funktioner avsedda för strukten lin_reg, som används för
*            implementering av maskininlärningsmodeller som baseras på linjär regression.
*************************************************************************************************/
#include "lin_reg.h"

// Statiska funktioner:
static void lin_reg_shuffle(struct lin_reg* self);
static void lin_reg_optimize(struct lin_reg* self,
                             const double input,
                             const double reference,
                             const double learning_rate);
static void lin_reg_extract(struct lin_reg* self,
                            const char* s);
static bool char_is_digit(const char c);
static void retrieve_double(struct double_vector* data,
                            char* s);

/*************************************************************************************************
* lin_reg_new: Initierar angiven regressionsmodell. Träningsdata måste tillföras i efterhand via
*              någon av funktioner lin_reg_load_training_data (för inläsning av träningsdata
*              från en textfil) eller lin_reg_set_training_data (för att passera pekare till
*              arrayer innehållande träningsdata).
*
*              - self         : Pekare till regressionsmodellen.
*************************************************************************************************/
void lin_reg_new(struct lin_reg* self)
{
   double_vector_new(&self->train_in);
   double_vector_new(&self->train_out);
   uint_vector_new(&self->train_order);
   self->bias = 0;
   self->weight = 0;
   return;
}

/*************************************************************************************************
* lin_reg_delete: Nollställer angiven regressionsmodell. Minnet för modellen frigörs dock inte,
*                 så denna kan återanvändas vid behov.
*
*                 - self: Pekare till regressionsmodellen.
*************************************************************************************************/
void lin_reg_delete(struct lin_reg* self)
{
   double_vector_delete(&self->train_in);
   double_vector_delete(&self->train_out);
   uint_vector_delete(&self->train_order);
   self->bias = 0;
   self->weight = 0;
   return;
}

/*************************************************************************************************
* lin_reg_ptr_new: Returnerar en pekare till en ny heapallokerad regressionsmodell. Träningsdata
*                  måste tillföras i efterhand via någon av funktioner lin_reg_load_training_data
*                  (för inläsning av träningsdata från en textfil) eller lin_reg_set_training_data
*                  (för att passera pekare till arrayer innehållande träningsdata).
*************************************************************************************************/
struct lin_reg* lin_reg_ptr_new(void)
{
   struct lin_reg* self = (struct lin_reg*)malloc(sizeof(struct lin_reg));
   if (!self) return 0;
   lin_reg_new(self);
   return self;
}
```

```c
/*****************************************************************************************
* lin_reg_delete: Nollställer och frigör minne för angiven heapallokerad regressionsmodell.
*                 Pekaren till regressionsmodellen sätts till null efter att minnet har frigjorts.
*
*                 - self: Adressen till regressionsmodellpekaren.
*****************************************************************************************/
void lin_reg_ptr_delete(struct lin_reg** self)
{
   lin_reg_delete(*self);
   free(*self);
   *self = 0;
   return;
}

/*****************************************************************************************
* lin_reg_load_training_data: Läser in träningsdata till angiven regressionsmodell från en fil
*                             via angiven filsökväg.
*
*                             - self    : Pekare till regressionsmodellen.
*                             - filepath: Pekare till filsökvägen.
*****************************************************************************************/
void lin_reg_load_training_data(struct lin_reg* self,
                                const char* filepath)
{
   FILE* fstream = fopen(filepath, "r");

   if (!fstream)
   {
      fprintf(stderr, "Could not open file at path %s!\n\n", filepath);
   }
   else
   {
      char s[100] = { '\0' };
      while (fgets(s, (int)sizeof(s), fstream))
      {
         lin_reg_extract(self, s);
      }
      fclose(fstream);
   }

   return;
}
```

```
/*****************************************************************************************
* lin_reg_set_training_data: Kopierar träningsdata till angiven regressionsmodell från refererade
*                            arrayer samt lagrar index för respektive träningsuppsättning.
*
*                            - self     : Pekare till regressionsmodellen.
*                            - train_in : Pekare till array innehållande insignaler.
*                            - train_out: Pekare till array innehållande referensvärden.
*                            - num_sets : Antalet passerade träningsuppsättningar.
*****************************************************************************************/
void lin_reg_set_training_data(struct lin_reg* self,
                               const double* train_in,
                               const double* train_out,
                               const size_t num_sets)
{
   const size_t new_size = self->train_in.size + num_sets;
   const size_t offset = self->train_in.size;

   double_vector_resize(&self->train_in, new_size);
   double_vector_resize(&self->train_out, new_size);
   uint_vector_resize(&self->train_order, new_size);

   for (size_t i = 0; i < num_sets; ++i)
   {
      self->train_in.data[offset + i] = train_in[i];
      self->train_out.data[offset + i] = train_out[i];
      self->train_order.data[offset + i] = offset + i;
   }

   return;
}

/*****************************************************************************************
* lin_reg_train: Tränar angiven regressionsmodell med givet antal epoker samt given lärhastighet.
*                I början av varje epok randomiseras ordningsföljden på träningsuppsättningarna
*                för att undvika att eventuella mönster som förekommer i träningsdatan ska
*                påverka träningen.
*
*                - self         : Pekare till regressionsmodellen.
*                - num_epochs   : Antalet epoker som ska genomföras vid träning.
*                - learning_rate: Den lärhastighet som ska användas vid träning för att
*                                 justera modellens parametrar vid avvikelse.
*****************************************************************************************/
void lin_reg_train(struct lin_reg* self,
                   const size_t num_epochs,
                   const double learning_rate)
{
   for (size_t i = 0; i < num_epochs; ++i)
   {
      lin_reg_shuffle(self);

      for (size_t j = 0; j < self->train_order.size; ++j)
      {
         const size_t k = self->train_order.data[j];
         lin_reg_optimize(self, self->train_in.data[k], self->train_out.data[k], learning_rate);
      }
   }

   return;
}
```

```c
/*****************************************************************************************
* lin_reg_predict: Genomför prediktion med angiven regressionsmodell via angiven insignal och
*                  returnerar det predikterade resultatet.
*
*                  - self : Pekare till regressionsmodellen.
*                  - input: Insignal som ska användas för prediktion.
*****************************************************************************************/
double lin_reg_predict(const struct lin_reg* self,
                       const double input)
{
   return self->weight * input + self->bias;
}


/*****************************************************************************************
* lin_reg_predict_all: Genomför prediktion med angiven regressionsmodell för samtliga insignaler
*                      från träningsdatan och skriver ut motsvarande predikterade utsignaler via
*                      angiven utström, där standardutenheten stdout används som default för
*                      utskrift i terminalen. Värden mycket nära noll avrundas för att undvika
*                      utskrift med ett flertal decimaler.
*
*                      - self     : Pekare till regressionsmodellen.
*                      - threshold: Tröskelvärde, där samtliga predikterade värden som ligger
*                                   inom intervallet [-threshold, threshold] avrundas till noll.
*                      - ostream  : Pekare till angiven utström (default = stdout).
*****************************************************************************************/
void lin_reg_predict_all(const struct lin_reg* self,
                         const double threshold,
                         FILE* ostream)
{
   if (!ostream) ostream = stdout;
   if (!self->train_in.size) return;

   const size_t last = self->train_in.size - 1;
   fprintf(ostream, "-------------------------------------------------------------------------------\n");

   for (size_t i = 0; i < self->train_in.size; ++i)
   {
      const double prediction = self->weight * self->train_in.data[i] + self->bias;
      fprintf(ostream, "Input: %g", self->train_in.data[i]);

      if (prediction < threshold && prediction > -threshold)
      {
         fprintf(ostream, "Output: %g", 0.0);
      }
      else
      {
         fprintf(ostream, "Output: %g", prediction);
      }

      if (i < last) fprintf(ostream, "\n");
   }

   fprintf(ostream, "-------------------------------------------------------------------------------\n\n");
   return;
}
```

```c
/*****************************************************************************************
* lin_reg_predict_range: Genomför prediktion med angiven regressionsmodell för insignaler mellan
*                        angivet start- och slutvärde i steg om angiven stegvärde. Motsvarande
*                        predikterad utsignal skrivs ut via angiven utström. Värden mycket nära
*                        noll avrundas för att undvika utskrift med ett flertal decimaler.
*
*                        - self     : Pekare till regressionsmodellen.
*                        - start_val: Minvärde för insignaler som ska testas.
*                        - end_val  : Maxvärde för insignaler som ska testas.
*                        - step     : Stegvärde/inkrementeringsvärde för insignaler.
*                        - threshold: Tröskelvärde, där samtliga predikterade värden inom
*                                     intervallet [-threshold, threshold] avrundas till noll.
*                         - ostream : Pekare till angiven utström (default = stdout).
*****************************************************************************************/
void lin_reg_predict_range(const struct lin_reg* self,
                           const double start_val,
                           const double end_val,
                           const double step,
                           const double threshold,
                           FILE* ostream)
{
   if (!self->train_in.size) return;
   if (!ostream) ostream = stdout;
   fprintf(ostream, "-------------------------------------------------------------------------\n");

   for (double i = start_val; i <= end_val; i += step)
   {
      const double prediction = self->weight * i + self->bias;
      fprintf(ostream, "Input: %g\n", i);

      if (prediction < threshold && prediction > -threshold)
      {
         fprintf(ostream, "Output: %g\n", 0.0);
      }
      else
      {
         fprintf(ostream, "Output: %g\n", prediction);
      }

      if (i < end_val) fprintf(ostream, "\n");
   }

   fprintf(ostream, "-------------------------------------------------------------------------\n\n");
   return;
}

/*****************************************************************************************
* lin_reg_shuffle: Randomiserar den inbördes ordningsföljden för angiven regressionsmodells
*                  träningsuppsättningar.
*
*                  - self: Pekare till regressionsmodellen.
*****************************************************************************************/
static void lin_reg_shuffle(struct lin_reg* self)
{
   for (size_t i = 0; i < self->train_order.size; ++i)
   {
      const size_t r = (size_t)rand() % self->train_order.size;
      const size_t temp = self->train_order.data[i];
      self->train_order.data[i] = self->train_order.data[r];
      self->train_order.data[r] = temp;
   }
   return;
}
```

```
/*****************************************************************************************
* lin_reg_optimize: Justerar parametrar för angiven regressionsmodell med målsättningen att minska
*                   aktuell avvikelse. Prediktion genomförs via angiven insignal, där predikterad
*                   utdat jämförs mot givet referensvärde för att beräkna aktuell avvikelse, som
*                   tillsammans med lärhastigheten avgör graden av justering.
*
*                   - self         : Pekare till regressionsmodellen.
*                   - input        : Insignal från träningsdata, som används för prediktion.
*                   - reference    : Referensvärde från träningsdata, som jämförs mot predikterad
*                                    utsignal för att beräkna aktuellt fel.
*                   - learning_rate: Den lärhastighet som ska användas vid träning för att
*                                    justera modellens parametrar vid avvikelse.
*****************************************************************************************/
static void lin_reg_optimize(struct lin_reg* self,
                             const double input,
                             const double reference,
                             const double learning_rate)
{
   const double prediction = self->weight * input + self->bias;
   const double error = reference - prediction;
   const double change_rate = error * learning_rate;
   self->bias += change_rate;
   self->weight += change_rate * input;
   return;
}

/*****************************************************************************************
* lin_reg_extract: Extraherar träningsdata i form av flyttal ur angivet textstycke. Ifall två
*                  flyttal lyckas extraheras så lagras dessa som en träningsuppsättning. Index
*                  för träningsuppsättningen lagras också för att enkelt kunna randomisera
*                  uppsättningarnas ordningsföljd vid träning utan att förflytta träningsdatan.
*
*                  - self: Pekare till regressionsmodellen.
*                  - s   : Pekare till textstycket som flyttal extraheras ur.
*****************************************************************************************/
static void lin_reg_extract(struct lin_reg* self,
                            const char* s)
{
   char num_str[20] = { '\0' };
   size_t index = 0;
   struct double_vector numbers = { .data = 0, .size = 0 };

   for (const char* i = s; *i; ++i)
   {
      if (char_is_digit(*i))
      {
         num_str[index++] = *i;
      }
      else
      {
         retrieve_double(&numbers, num_str);
         index = 0;
      }
   }

   if (index)
   {
      retrieve_double(&numbers, num_str);
   }
   if (numbers.size == 2)
   {
      double_vector_push(&self->train_in, numbers.data[0]);
      double_vector_push(&self->train_out, numbers.data[1]);
      uint_vector_push(&self->train_order, self->train_order.size);
   }
   double_vector_delete(&numbers);
   return;
}
```

```
/*****************************************************************************************
* char_is_digit: Indikerar ifall givet tecken utgör en siffra eller ett relaterat tecken, såsom
*                ett minustecken eller en punkt. Eftersom flyttal ibland matas in både med
*                punkt samt kommatecken så utgör båda giltiga tecken.
*
*                - c: Det tecken som ska kontrolleras.
*****************************************************************************************/
static bool char_is_digit(const char c)
{
   const char* s = "0123456789-.,";

   for (const char* i = s; *i; ++i)
   {
      if (*i == c) return true;
   }

   return false;
}

/*****************************************************************************************
* retrieve_double: Typomvandlar innehåll lagrat som text till ett flyttal och lagrar resultatet
*                  i en vektor. Innan typomvandlingen äger rum ersätts eventuella kommatecken
*                  med punkt, vilket möjliggör att flyttal kan läsas in både med punkt eller
*                  kommatecken som decimaltecken.
*
*                  - data: Pekare till den vektor som typomvandlat flyttal ska lagras i.
*                  - s   : Pekare till det textstycke som ska typomvandlas till ett flyttal.
*****************************************************************************************/
static void retrieve_double(struct double_vector* data,
                            char* s)
{
   for (char* i = s; *i; ++i)
   {
      if (*i == ',') *i = '.';
   }

   const double num = atof(s);
   double_vector_push(data, num);
   s[0] = '\0';
   return;
}
```

Exempel 2 b) - Filen *double_vector.h:*

```
/*****************************************************************************************
* double_vector.h: Implementering av dynamiska vektorer för lagring av flyttal via strukten
*                  double_vector samt motsvarande externa funktioner.
*****************************************************************************************/
#ifndef DOUBLE_VECTOR_H_
#define DOUBLE_VECTOR_H_

/* Inkluderingsdirektiv: */
#include <stdio.h>
#include <stdlib.h>

/*****************************************************************************************
* double_vector: Vektor innehållande ett dynamiskt fält för lagring av flyttal. Antalet element
*                som lagras i fältet räknas upp och uttrycks i form av vektorns storlek.
*****************************************************************************************/
struct double_vector
{
   double* data; /* Pekare till dynamiskt fält för lagring av flyttal. */
   size_t size;  /* Vektorns storlek (antalet element i fältet). */
};

/* Externa funktioner: */
void double_vector_new(struct double_vector* self);
void double_vector_delete(struct double_vector* self);
struct double_vector* double_vector_ptr_new(const size_t size);
void double_vector_ptr_delete(struct double_vector** self);
int double_vector_resize(struct double_vector* self,
                         const size_t new_size);
int double_vector_push(struct double_vector* self,
                       const double new_element);
int double_vector_pop(struct double_vector* self);
void double_vector_print(const struct double_vector* self,
                         FILE* ostream);
double* double_vector_begin(const struct double_vector* self);
double* double_vector_end(const struct double_vector* self);

/* Funktionspekare: */
extern void (*double_vector_clear)(struct double_vector* self);

#endif /* DOUBLE_VECTOR_H_ */
```

Exempel 2 b) - Filen *double_vector.c:*

```
/*****************************************************************************************
* double_vector.c: Innehåller funktionsdefinitioner för implementering av dynamiska vektorer
*                  för lagring av flyttal via strukten double_vector.
*****************************************************************************************/
#include "double_vector.h"

/*****************************************************************************************
* double_vector_new: Initierar angiven vektor.
*
*                    - self: Pekare till vektorn.
*****************************************************************************************/
void double_vector_new(struct double_vector* self)
{
   self->data = 0;
   self->size = 0;
   return;
}

/*****************************************************************************************
* double_vector_delete: Tömmer innehållet i angiven vektor.
*
*                       - self: Pekare till vektorn.
*****************************************************************************************/
void double_vector_delete(struct double_vector* self)
{
   free(self->data);
   self->data = 0;
   self->size = 0;
   return;
}

/*****************************************************************************************
* double_vector_ptr_new: Returnerar en pekare till en ny heapallokerad vektor av angiven storlek.
*
*                        - size: Fältets storlek (antalet element det rymmer) vid start.
*****************************************************************************************/
struct double_vector* double_vector_ptr_new(const size_t size)
{
   struct double_vector* self = (struct double_vector*)malloc(sizeof(struct double_vector));
   if (!self) return 0;
   self->data = 0;
   self->size = 0;
   double_vector_resize(self, size);
   return self;
}

/*****************************************************************************************
* double_vector_ptr_delete: Frigör minne för angiven heapallokerad vektor. Vektorpekarens adress
*                           passeras för att både frigöra minnet för det dynamiska fält denna pekar
*                           på, minnet för själva vektorn samt att vektorpekaren sätts till null.
*
*                           - self: Adressen till vektorpekaren.
*****************************************************************************************/
void double_vector_ptr_delete(struct double_vector** self)
{
   double_vector_delete(*self);
   free(*self);
   *self = 0;
   return;
}
```

```c
/*****************************************************************************************
* double_vector_resize: Ändrar storleken / kapaciteten på angiven vektor via omallokering.
*
*                       - self    : Pekare till vektorn.
*                       - new_size: Vektorns nya storlek efter omallokeringen.
*****************************************************************************************/
int double_vector_resize(struct double_vector* self,
                         const size_t new_size)
{
   double* copy = (double*)realloc(self->data, sizeof(double) * new_size);
   if (!copy) return 1;
   self->data = copy;
   self->size = new_size;
   return 0;
}

/*****************************************************************************************
* double_vector_push: Lägger till ett nytt element längst bak i angiven vektor.
*
*                     - self       : Pekare till vektorn.
*                     - new_element: Det nya element som ska läggas till.
*****************************************************************************************/
int double_vector_push(struct double_vector* self,
                       const double new_element)
{
   double* copy = (double*)realloc(self->data, sizeof(double) * (self->size + 1));
   if (!copy) return 1;
   copy[self->size++] = new_element;
   self->data = copy;
   return 0;
}

/*****************************************************************************************
* double_vector_pop: Tar bort ett element längst bak i angiven vektor, om ett sådant finns.
*
*                    - self: Pekare till vektorn.
*****************************************************************************************/
int double_vector_pop(struct double_vector* self)
{
   if (self->size <= 1)
   {
      double_vector_delete(self);
      return 1;
   }
   else
   {
      double* copy = (double*)realloc(self->data, sizeof(double) * (self->size - 1));
      if (!copy) return 1;
      self->data = copy;
      self->size--;
      return 0;
   }
}
```

```c
/*****************************************************************************************
* double_vector_print: Skriver ut innehåll lagrat i angiven vektor via angiven utström, där
*                      standardutenheten stdout används som default för utskrift i terminalen.
*
*                      - self   : Pekare till vektorn.
*                      - ostream: Pekare till angiven utström (default = stdout).
*****************************************************************************************/
void double_vector_print(const struct double_vector* self,
                         FILE* ostream)
{
   if (!self->size) return;
   if (!ostream) ostream = stdout;
   fprintf(ostream, "--------------------------------------------------------------------------\n");

   for (const double* i = self->data; i < self->data + self->size; ++i)
   {
      fprintf(ostream, "%g\n", *i);
   }

   fprintf(ostream, "--------------------------------------------------------------------------\n\n");
   return;
}

/*****************************************************************************************
* double_vector_begin: Returnerar adressen till det första elementet i angiven vektor.
*
*                      - self: Pekare till vektorn.
*****************************************************************************************/
double* double_vector_begin(const struct double_vector* self)
{
   return self->data;
}

/*****************************************************************************************
* double_vector_end: Returnerar adressen direkt efter det sista elementet i angiven vektor.
*
*                    - self: Pekare till vektorn.
*****************************************************************************************/
double* double_vector_end(const struct double_vector* self)
{
   return self->data + self->size;
}

/*****************************************************************************************
* double_vector_clear: Tömmer innehållet i angiven vektor.
*
*                      - self: Pekare till vektorn.
*****************************************************************************************/
void (*double_vector_clear)(struct double_vector* self) = &double_vector_delete;
```

Exempel 2 b) - Filen *uint_vector.h:*

```
/*****************************************************************************************
* uint_vector.h: Implementering av dynamiska vektorer för lagring av osignerade heltal via
*                strukten uint_vector samt motsvarande externa funktioner.
*****************************************************************************************/
#ifndef UINT_VECTOR_H_
#define UINT_VECTOR_H_

/* Inkluderingsdirektiv: */
#include <stdio.h>
#include <stdlib.h>

/*****************************************************************************************
* uint_vector: Vektor innehållande ett dynamiskt fält för lagring av osignerade heltal. Antalet
*              element som lagras i fältet räknas upp och uttrycks i form av vektorns storlek.
*****************************************************************************************/
struct uint_vector
{
   size_t* data; /* Pekare till dynamiskt fält för lagring av osignerade heltal. */
   size_t size;  /* Vektorns storlek (antalet element i fältet). */
};

/* Externa funktioner: */
void uint_vector_new(struct uint_vector* self);
void uint_vector_delete(struct uint_vector* self);
struct uint_vector* uint_vector_ptr_new(const size_t size);
void uint_vector_ptr_delete(struct uint_vector** self);
int uint_vector_resize(struct uint_vector* self,
                       const size_t new_size);
int  uint_vector_push(struct uint_vector* self,
                      const size_t new_element);
int uint_vector_pop(struct uint_vector* self);
void uint_vector_print(const struct uint_vector* self,
                       FILE* ostream);
size_t* uint_vector_begin(const struct uint_vector* self);
size_t* uint_vector_end(const struct uint_vector* self);

/* Funktionspekare: */
extern void (*uint_vector_clear)(struct uint_vector* self);

#endif /* UINT_VECTOR_H_ */
```

Exempel 2 b) - Filen *uint_vector.c:*

```
/*****************************************************************************************
* uint_vector.c: Innehåller funktionsdefinitioner för implementering av dynamiska vektorer
*                  för lagring av osignerade heltal via strukten uint_vector.
*****************************************************************************************/
#include "uint_vector.h"

/*****************************************************************************************
* uint_vector_new: Initierar angiven vektor.
*
*                  - self: Pekare till vektorn.
*****************************************************************************************/
void uint_vector_new(struct uint_vector* self)
{
   self->data = 0;
   self->size = 0;
   return;
}

/*****************************************************************************************
* uint_vector_delete: Tömmer innehållet i angiven vektor.
*
*                     - self: Pekare till vektorn.
*****************************************************************************************/
void uint_vector_delete(struct uint_vector* self)
{
   free(self->data);
   self->data = 0;
   self->size = 0;
   return;
}

/*****************************************************************************************
* uint_vector_ptr_new: Returnerar en pekare till en ny heapallokerad vektor av angiven storlek.
*
*                      - size: Fältets storlek (antalet element det rymmer) vid start.
*****************************************************************************************/
struct uint_vector* uint_vector_ptr_new(const size_t size)
{
   struct uint_vector* self = (struct uint_vector*)malloc(sizeof(struct uint_vector));
   if (!self) return 0;
   self->data = 0;
   self->size = 0;
   uint_vector_resize(self, size);
   return self;
}

/*****************************************************************************************
* uint_vector_ptr_delete: Frigör minne för angiven heapallokerad vektor. Vektorpekarens adress
*                         passeras för att både frigöra minnet för det dynamiska fält denna pekar
*                         på, minnet för själva vektorn samt att vektorpekaren sätts till null.
*
*                         - self: Adressen till vektorpekaren.
*****************************************************************************************/
void uint_vector_ptr_delete(struct uint_vector** self)
{
   uint_vector_delete(*self);
   free(*self);
   *self = 0;
   return;
}
```

```c
/*****************************************************************************************
* uint_vector_resize: Ändrar storleken / kapaciteten på angiven vektor via omallokering.
*
*                     - self    : Pekare till vektorn.
*                     - new_size: Vektorns nya storlek efter omallokeringen.
*****************************************************************************************/
int uint_vector_resize(struct uint_vector* self,
                       const size_t new_size)
{
   size_t* copy = (size_t*)realloc(self->data, sizeof(size_t) * new_size);
   if (!copy) return 1;
   self->data = copy;
   self->size = new_size;
   return 0;
}

/*****************************************************************************************
* uint_vector_push: Lägger till ett nytt element längst bak i angiven vektor.
*
*                   - self       : Pekare till vektorn.
*                   - new_element: Det nya element som ska läggas till.
*****************************************************************************************/
int uint_vector_push(struct uint_vector* self,
                     const size_t new_element)
{
   size_t* copy = (size_t*)realloc(self->data, sizeof(size_t) * (self->size + 1));
   if (!copy) return 1;
   copy[self->size++] = new_element;
   self->data = copy;
   return 0;
}

/*****************************************************************************************
* uint_vector_pop: Tar bort ett element längst bak i angiven vektor, om ett sådant finns.
*
*                  - self: Pekare till vektorn.
*****************************************************************************************/
int uint_vector_pop(struct uint_vector* self)
{
   if (self->size <= 1)
   {
      uint_vector_delete(self);
      return 1;
   }
   else
   {
      size_t* copy = (size_t*)realloc(self->data, sizeof(size_t) * (self->size - 1));
      if (!copy) return 1;
      self->data = copy;
      self->size--;
      return 0;
   }
}
```

```c
/*****************************************************************************************
* uint_vector_print: Skriver ut innehåll lagrat i angiven vektor via angiven utström, där
*                    standardutenheten stdout används som default för utskrift i terminalen.
*
*                    - self   : Pekare till vektorn.
*                    - ostream: Pekare till angiven utström (default = stdout).
*****************************************************************************************/
void uint_vector_print(const struct uint_vector* self,
                       FILE* ostream)
{
   if (!self->size) return;
   if (!ostream) ostream = stdout;
   fprintf(ostream, "------------------------------------------------------------------------\n");

   for (const size_t* i = self->data; i < self->data + self->size; ++i)
   {
      fprintf(ostream, "%zu\n", *i);
   }

   fprintf(ostream, "------------------------------------------------------------------------\n\n");
   return;
}

/*****************************************************************************************
* uint_vector_begin: Returnerar adressen till det första elementet i angiven vektor.
*
*                    - self: Pekare till vektorn.
*****************************************************************************************/
size_t* uint_vector_begin(const struct uint_vector* self)
{
   return self->data;
}

/*****************************************************************************************
* uint_vector_end: Returnerar adressen direkt efter det sista elementet i angiven vektor.
*
*                  - self: Pekare till vektorn.
*****************************************************************************************/
size_t* uint_vector_end(const struct uint_vector* self)
{
   return self->data + self->size;
}

/*****************************************************************************************
* uint_vector_clear: Tömmer innehållet i angiven vektor.
*
*                    - self: Pekare till vektorn.
*****************************************************************************************/
void (*uint_vector_clear)(struct uint_vector* self) = &uint_vector_delete;
```