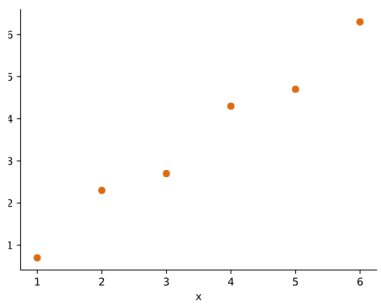
1 Linear Regression

A linear regression model is a linear function that maps the feature(s) to the target value. In the case of one feature, this model is a line. And the best model is the line which best fits the data.

1.1 Basic example

x	t
1	0.7
2	2.3
3	2.7
4	4.3
5	4.7
6	6.3



features

We define the set of all instaces as a matrix $\mathbf{X}$, where each row is an instance $\mathbf{x}_i \in \mathbf{X}$ and the features are columns of $\mathbf{X}$ where a feature $x_i \in \mathbf{x}$.

model with one feature

Assuming we have a single feature x_i the standard linear regression model can be defined as:

$$f_{\mathbf{w},\mathbf{b}}(\mathbf{x}) = \mathbf{w}x_1 + b \tag{1}$$

Where

- $\mathbf{w}$ is the weight vector, which you can also just see as the slope of the line
- $\mathbf{b}$ is the bias, which you can also see as the y-intercept
- x_1 is the feature we are using
- $\mathbf{x}$ is the instance

model with multiple features

For each new feature x_i we add a new weight w_i to the model. The model then becomes:

$$\begin{aligned}
\text{one feature} &: f_{\mathbf{w},\mathbf{b}}(\mathbf{x}) = \mathbf{w}x_1 + b \\
\text{two features} &: f_{\mathbf{w},\mathbf{b}}(\mathbf{x}) = w_1x_1 + w_2x_2 + b \\
&\vdots \\
n \text{ features} &: f_{\mathbf{w},\mathbf{b}}(\mathbf{x}) = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b
\end{aligned}$$

Expressed in terms of the dot product of the weight vector and the feature vector, the model becomes:

$$
\begin{aligned}
f_{\mathbf{w},b}(\mathbf{x}) &= \mathbf{w}^\intercal \mathbf{x} + b && \text{dot product notation} \\
&= \sum_{i=1}^{n} w_i x_i + b && \text{expanded notation} \\
&= ||\mathbf{w}||\ ||\mathbf{x}||\ \cos(\theta) + b && \text{geometric notation}
\end{aligned}
$$

dot product intuition

The weights **w** represent the relative importance of each feature to the model. The dot product of the weights and the features is a measure of how much the features contribute to the model.

loss function

The loss function takes the model as an argument where

- the model maps the data to some output
- the loss function maps the model to a scalar loss value

A common loss function for linear regression is the mean squared error (MSE) loss function. The MSE loss function is defined as:

$$
\begin{aligned}
\text{loss}_{X,T}(p) &= \frac{1}{n} \sum_{j} (f_p(\mathbf{x}_j) - t_j)^2 && \text{simplified form} \\
\text{loss}_{X,T}(\mathbf{w}, b) &= \frac{1}{n} \sum_{j} (\mathbf{w}^\intercal \mathbf{x}_j + b - t_j)^2 && \text{expanded form}
\end{aligned}
$$

intuition	MSE loss function returns a large value if the difference between prediction and value is large and vv.
residual	This is the difference between the predicted value and the target value
squared residual	We square the residual both to avoid negative and positive residuals cancelling out and to ensure larger residuals effect the loss more heavily than smaller residuals.

A common analogy is that values which yield larger residuals are like rubber bands that are stretched further, pulling the prediction of our model closer.

MSE variations

Some equivalent forms of the MSE loss function are:

$$\sum_j (f_p(\mathbf{x}^j) - y^j)^2$$
$$\frac{1}{n}\sum_j (f_p(\mathbf{x}^j) - y^j)^2$$
$$\frac{1}{2}\sum_j (f_p(\mathbf{x}^j) - y^j)^2$$
$$\sqrt{\frac{1}{n}\sum_j (f_p(\mathbf{x}^j) - y^j)^2}$$

2 Linear Models and Search

We have the **model space** which maps all possible models to their loss value. The surface this map generates is called the **loss surface/loss landscape**.

2.1 optimization

Definition 2.1 (mathematical optimization)**.** Mathematical optimization is the process of trying to find the input model p that minimizes/maximizes the loss function.

Formally for an input $p = \{w_1, w_2, \ldots, w_n, b\}$ the optimization problem is:

$$\hat{p} = \arg\min_p \text{loss}_{X,T}(p) \tag{2}$$

Which again just express that we want to find (in our example) the minimum $\hat{p}$ of the loss function.

Definition 2.2 (machine learning optimization)**.** Machine learning optimization is the process of trying to find the input model p that minimizes/maximizes the loss function for the *test data* and best generalizes to new data.

2.2 random search

Here we make a small step to a nearby point, if the loss goes up, we move back to the previous point, if it goes down we stay. We stop when the loss gets to a predefined level.

Listing 1: random search

```
start with a random model p in model space
loop:
    generate a random model p'
    if loss(p') < loss(p):
        p = p'
```

To pick the next point we pick from the *hypersphere* (e.g. a circle in 2D) with the radius r and the center at the current point.

Definition 2.3 (convex problem)**.** A convex problem is a problem where the loss surface is convex (bowl shaped). It can be identified by the fact that if we draw a line between any two points on the surface, the line will never intersect the surface.

search for non-convex problems

For non-convex problems, so problems with multiple local minima, we can get stuck in one of the local minima. To avoid this we can use *simulated annealing*.

Definition 2.4 (simulated annealing)**.** If the next point chosen has a higher loss then we still pick it, but only with a small probability.

Listing 2: simulated annealing

```
start with a random model p in model space
loop:
    generate a random model p'
    if loss(p') < loss(p):
        p = p'
    else:
        with probability q:
            p = p'
```

In many situations the local minima are okay and we don't need to find the global minimum. We only want to make sure that we can escape bad local minima, but this doesn't imply that all local minima are bad.

variations on random search

We can vary the way we pick the next point.

- fixed radius - we can use a fixed radius for the hypersphere
- random uniform - we can pick the next point from a uniform distribution
- random normal - we can pick the next point from a normal distribution

2.3 continuous vs discrete model space

Definition 2.5 (continuous model space)**.** A continuous model space is a model space where between any two models there is always another model.

Definition 2.6 (discrete model space)**.** A discrete model space is a model space where we don't always have another model between any two models e.g. the space of all decision trees.

How we define closeness also matters. For example, in the case of a discrete model space of decision trees we can define two trees as being close if we can transform one tree to another by adding/removing a single node.

2.4 parallel search

Here we run multiple searches at the same time. We can optimize this by using methods which include a population of agents that communicate with each other called *population methods*.

population parallel search

Definition 2.7 (population methods)**.** Population methods are methods that use a population of searching agents that communicate with each other during their search of the model space to improve the efficiency of the search.

Examples of population methods are :

- evolutionary algorithms
  - genetic algorithms
  - evolutionary strategies
- particle swarm optimization
- ant colony optimization

Listing 3: evolutionary algorithms

```
1  start with a population of k random models
2  loop:
3      rank the population by loss
4      remove the half with the worst loss
5      breed a new population of k models
```

Some pros and cons of population methods are:

pros	cons
• are powerful • its easy to parallelize	• can be slow • can be hard to tune

2.5 black box optimization

Definition 2.8 (black box optimization)**.** Black box optimizations are methods that only require us to compute the loss function, we don't need to know anything about the internals of the model.

Some key properties of black box optimization are:

- they are very simple
- we only need to compute loss function
- they can require many iterations
- they also work for discrete model spaces (e.g. decision trees)

3 Gradient Descent

We start of by developing random search a bit, instead of picking a random point with the lowest loss we look at k random steps near us and pick the one with the lowest loss. This is called *branchnig search*.

Listing 4: branching search

```
start with a random model p in model space
loop:
    pick k random points p_i close to p
    p' ← arg min_{p_i} loss(p_i)
    if loss(p') < loss(p):
        p ← p'
```

3.1 gradient descent: outline

Using calculus we can find the direction in which the loss function decreases the fastest. This direction is opposite of the *gradient* of the loss function.

Definition 3.1 (slope)**.** Slope of a linear function is how much a we move up or down for a given change to the right.

Definition 3.2 (tangent line)**.** The tangent line to a function at a given point is a line that touches the function at that point and has the same slope as the function at that point.

Definition 3.3 (derivative)**.** The derivative of a function at a given point is the slope of the tangent line to the function at that point.

Say we had a loss function $f(x) = \text{loss}(p)$ representing a simple 2D convex curve then at a point p. We define the slope at this point as the derivative of the loss function given the input p, this is denoted as $f'(p)$. The line tangent to this point is then given by the equation:

$$g(x) = f'(p)x + b \tag{3}$$

Definition 3.4 (gradient descent)**.** Gradient descent is a method that uses the gradient of the loss function to iteratively move towards the minimum of the loss function. We literally 'descend the gradient'.

The gradient is simply a generalizes of the derivative, instead of a tangent line we have a tangent (hyper)plane. Just as the tagent line, the tangent hyperplane is an approximation of the loss function at a given point that allows us to easily find the direction of steepest ascent and descent. We formally define the gradient as:

Definition 3.5 (gradient)**.** The gradient of a function at a given point is a vector of slopes of the tangent hyperplanes to the loss function at that point. For a function f with an input $\mathbf{x}$ the gradient is denoted as $\nabla f(\mathbf{x})$ and expressed as follows :

$$\nabla f(\mathbf{x}) = \left(\frac{\partial f(\mathbf{x})}{\partial x_1}, \frac{\partial f(\mathbf{x})}{\partial x_2}, \ldots, \frac{\partial f(\mathbf{x})}{\partial x_n} \right) \tag{4}$$

We can express the general equation for the tangent hyperplane at a point $\mathbf{p}$ as:

$$\begin{aligned} g(\mathbf{x}) &= f'_{x_1}(p)x_1 + f'_{x_2}(p)x_2 + \cdots + f'_{x_n}(p)x_n + b \\ &= \nabla f(\mathbf{p})^\intercal \mathbf{x} + b \end{aligned}$$

direction of steepest ascent and descent

We now want to find the diretion of steepest ascent on the local linear approximation $g(\mathbf{x})$. First we can make a few assumptions

- $b = 0$ - we can always shift the tangent hyperplane up or down without changing the slope
- $||\mathbf{x}|| = 1$ - the magnitude of the step doesn't matter, only the direction
- start = origin - we can always shift the tangent hyperplane to the origin without changing the slope

This simplifies $g(\mathbf{x})$ to:

$$g(\mathbf{x}) = \nabla f(\mathbf{p})^\intercal \mathbf{x} \tag{5}$$

Since $\nabla f(\mathbf{p})$ is constant the question becomes what $\mathbf{x}$ with $||\mathbf{x}|| = 1$ maximizes the dot product $\nabla f(\mathbf{p})^\intercal \mathbf{x}$. To solve this we use the geometric definition of the dot product:

$$\begin{aligned} \nabla f(\mathbf{p})^\intercal \mathbf{x} &= ||\nabla f(\mathbf{p})||\ ||\mathbf{x}||\ \cos(\theta) \\ &= ||\nabla f(\mathbf{p})||\ \cos(\theta) \end{aligned}$$

The maximum value of $\cos(\theta)$ is 1, and this occurs when $\theta = 0$. This means that the direction of steepest ascent on the tangent hyperplane is the direction of $\nabla f(\mathbf{p})$ and conversely the direction of steepest descent is $-\nabla f(\mathbf{p})$. So to move towards the minimum of the loss function we move in the direction of $-\nabla f(\mathbf{p})$.

3.2 gradient descent: algorithm

Starting at some candidate point $\mathbf{p}$ we can use the gradient to iteratively move towards the minimum of the loss function. The algorithm is as follows:

Listing 5: gradient descent algorithm

```
1  pick a random point p in the model space
2  loop:
3      p ← p − η∇loss(p)
```

We subtract because the gradient is pointing in the direction of steepest ascent and we want to move in the direction of steepest descent. The η is the *learning rate*

- it scales down the step size
- its chosen by trail and error
- it stays constant throughout the search

3.3 gradient descent: example

calculating the gradient

Say we are using MSE as our loss function and we have a model with a single feature x_i. The loss function is then:

$$\text{loss}(w, b) = \frac{1}{n} \sum_i (w x_i + b - t_i)^2 \tag{6}$$

To calculate the gradient of the loss function we need to calculate the partial derivatives of the loss function with respect to the weights and the bias. The gradient is then:

$$\nabla \text{loss}(w, b) = \left(\frac{\partial \text{loss}(w, b)}{\partial w}, \frac{\partial \text{loss}(w, b)}{\partial b} \right) \tag{7}$$

The partial derivative with respect to the weights and biases is then:

$$\begin{aligned}
\frac{\partial \text{loss}(w, b)}{\partial w} &= \frac{\partial \frac{1}{n}\sum_i (wx_i + b - t_i)^2}{\partial w} \\
&= \frac{1}{n}\sum_i \frac{\partial (wx_i + b - t_i)^2}{\partial w} \quad \text{(sum rule)} \\
&= \frac{1}{n}\sum_i \frac{\partial (wx_i + b - t_i)^2}{\partial (wx_i + b - t_i)} \frac{\partial (wx_i + b - t_i)^2}{\partial w} \quad \text{(chain rule)} \\
&= \frac{2}{n}\sum_i (wx_i + b - t_i)x_i \\
\frac{\partial \text{loss}(w, b)}{\partial b} &= \frac{\partial \frac{1}{n}\sum_i (wx_i + b - t_i)^2}{\partial b} \\
&= \frac{2}{n}\sum_i (wx_i + b - t_i) \quad \text{(sum rule)}
\end{aligned}$$

example gradient descent algorithm

Given the gradient we computed above our gradient descent algorithm is then:

Listing 6: gradient descent algorithm example

```
1  pick a random point (w, b) in the model space
2  loop:
3      (w, b)  ←  (w, b)  − η ( (2/n) Σ_i (w x_i + b − t_i) x_i , (2/n) Σ_i (w x_i + b − t_i) )
       next guess   current guess                  gradient
```

3.4 properties of gradient descent

Some cons of gradient descent are:

- only works for continuous model spaces
  - with smooth loss functions
  - for which we can compute the gradient
- we can get stuck in local minima

Some pros of gradient descent are:

- it's fast and low memory
- it's very accurate

4 Gradient Descent and Classification

To apply gradient descent to classification we define the model using a similar functional paradigm. We have a hyperplane defined by $f_{\mathbf{w},b}(\mathbf{x})$ which separates the data into two classes. The model is then:

$$f_{\mathbf{w},b}(\mathbf{x}) = \mathbf{w}^\intercal \mathbf{x} + b = \begin{cases} \text{class 1} & \text{if } \mathbf{w}^\intercal \mathbf{x} + b > 0 \\ \text{class 2} & \text{if } \mathbf{w}^\intercal \mathbf{x} + b < 0 \end{cases} \tag{8}$$

4.1 properties of the hyperplane

In 2D $f_{\mathbf{w},\mathbf{b}}(\mathbf{x})$ describes the plane that intersects the decision boundary. The weight vector $\mathbf{w}$ is the vector perpendicular to the plane in the direction $f_{\mathbf{w},\mathbf{b}}(\mathbf{x}) > 0$.

4.2 discontinuous loss surface

In the case a loss surface is not smooth and continuous but instead consisting of flat regions and sharp edges we have a much harder time finding the minimum using any of the methods we have discussed.

random search	would have to do a random walk till it grind a ridge by accident
gradient descent	the gradient is 0 everywhere except at the edges where its undefined so this would just break

purpose of loss function

Loss functions serve two purposes:

1. To express a quantity we want to maximize in our search for a good model
2. To provide a smooth loss surface that we can use to find the path from a good model to a bad one

solving discontinuous loss surfaces

To avoid the issue with discontinuous loss surfaces its a common practice to replace our loss function that has a minimum at (roughly) the same model but has a smooth differentiable loss surface.

4.3 types of loss functions

The types of loss functions covered in the course are

- least squares loss
- log loss / cross entropy loss
- SVM loss

least squares loss function

We define the least-squares loss as the sum of the squared residuals for the positive class instances and the negative class instance. We can express this as follows:

$$\text{loss}(\mathbf{w}, b) = \sum_{i \in \text{pos}} (\mathbf{w}^\intercal \mathbf{x}_i + b - 1)^2 + \sum_{i \in \text{neg}} (\mathbf{w}^\intercal \mathbf{x}_i + b + 1)^2 \tag{9}$$

Some observations

- The minima between the discontinuous loss surface and the smooth loss surface are roughly the same
- The smooth loss surface is differentiable everywhere so we can apply gradient descent