TEST FOR DEV: Introduction to Data Science

WS 23/24, RWTH Aachen

January 12, 2024

1 Regression

In the next three chapters, we'll look at **error-based learning**. In general, this learning approach functions as follows: Error-based learning

- We have a **parameterized prediction model** which is initialized with random parameters.
- An **error function** is then used to evaluate the performance of this model when it makes predictions for instances in a training dataset. Error function
- Based on the results of the error function, the parameters are **iteratively adjusted** to create a more and more accurate model.

There are different approaches to realizing error-based learning:

- Regression (covered in this section)
- SVMs (covered in the next section)
- Neural networks (covered in a later section)
- Genetic algorithms, or other evolutionary approaches

We'll start with **regression** and the following basic idea. Regression

$$\text{Our model: } f : \text{descriptive features} \to \text{target features}$$
$$\text{Goal: find } f \text{ minimizing } error(\text{prediction}, \text{observed data})$$

When we compare this approach to decision trees, we see:

- Decision trees were initially developed for categorical features and then extended to continuous features.
- Regression followed the reverse path, which means it's most **suitable for continuous data**.
- Still, both are supervised learning techniques.

1.1 Simple linear regression

Consider the following simple example where we have:

- Rental price p_r as our target feature, and
- Size s as our descriptive (continuous) feature

We assume a linear dependency $p_r = b + a \cdot s$ and now want to base our prediction of the rental prize on the size. The example will guide us through this subchapter.

General problem regression

The **general problem** is given as follows:

- We have given n data rows in a set $\mathcal{D}$ with a target feature t and descriptive features $\mathbf{d} = (d[1], d[2], \cdots, d[m])$, and
- We want to find a regression function $\mathbb{M}_\mathbf{w}$ with a constant weight and a weight for each feature, where
- We predict $\text{pred}(t) = \mathbb{M}_\mathbf{w}(\mathbf{d}) = \mathbb{M}_{(w[0],w[1],\cdots,w[m])}(d[1], d[2], \cdots, d[m])$

In our example, we only have one descriptive feature $\mathbf{d} = (d[1])$, two weights $\mathbf{w} = (w[0], w[1])$, and the regression function is linear, so $\underbrace{\mathbb{M}_\mathbf{w}(d)}_{p_r} = \underbrace{w[0]}_{b} + \underbrace{w[1]}_{a} \cdot \underbrace{d[1]}_{s}$.

What can be seen is that the weights $\mathbf{w} = (w[0], w[1])$ define the linear function. Our goal is now to find the weights such that the resulting function has the "smallest error". There are different ways to characterize the error with different **error metrics**:

Error metrics

Sum of squared errors L_2

- Sum of squared errors

$$L_2(\mathbb{M}_\mathbf{w}, \mathcal{D}) = \frac{1}{2} \cdot \sum_{i=1}^{n} (t_i - \mathbb{M}_\mathbf{w}(d_i))^2$$

  - For each instance: compute the error, then square it
  - Compute the sum of the results
  - Finally, take half ($\rightarrow$ end result)

Mean squared error MSE

- Mean squared error

$$MSE(\mathbb{M}_\mathbf{w}, \mathcal{D}) = \frac{1}{n} \cdot \sum_{i=1}^{n} t_i - \mathbb{M}_\mathbf{w}(d_i)$$

Root mean squared error $RMSE$

- Root mean squared error

$$RMSE(\mathbb{M}_\mathbf{w}, \mathcal{D}) = \sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} t_i - \mathbb{M}_\mathbf{w}(d_i)}$$

Mean absolute error MAE

- Mean absolute error

$$MAE(\mathbb{M}_\mathbf{w}, \mathcal{D}) = \frac{1}{n} \cdot \sum_{i=1}^{n} \mid t_i - \mathbb{M}_\mathbf{w}(d_i) \mid$$

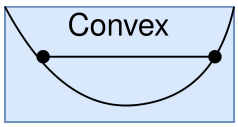
All of the introduced error metrics express the same idea, but usually, L_2 is chosen since it has a simple derivative. We'll take a look at the **partial derivative of error**, concretely of L_2 (here, i refers to the training instance, and j to one of the multiple descriptive features)

$$\begin{aligned} \frac{\delta}{\delta \mathbf{w}[j]} L_2(\mathbb{M}_\mathbf{w}, \mathcal{D}) &= \frac{\delta}{\delta \mathbf{w}[j]} \frac{1}{2} \sum_{i=1}^{n} (t_i - \mathbb{M}_\mathbf{w}(d_i))^2 \\ &= -\sum_{i=1}^{n} \left((t_i - \mathbb{M}_\mathbf{w}(d_i)) \cdot d_i[j] \right) \end{aligned}$$

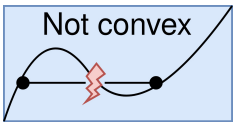
Using this derivate, we can now find the weights minimizing the error.

- Brute force, meaning we try as many values as possible for the weights, isn't feasible in practice (not even for simple linear case).
- But we can use that our error surface is convex and therefore has a global minimum, enabling faster methods.
  - Take the partial derivatives (for linear case: $\delta/\delta\mathbf{w}[0]\ L_2(\mathbb{M}_\mathbf{w}, \mathcal{D})$, and $\delta/\delta\mathbf{w}[1]\ L_2(\mathbb{M}_\mathbf{w}, \mathcal{D})$)
  - Find the correct values for all partial derivatives to result in zero (Needs to be the case for an actual global minimum)

One important thing to mention, the convex property of a function is very useful here:



- One local minimum = global minimum
- Gradient descent works



- Multiple local minima, so global minimum harder to find
- Gradient descent might fail

Figure 1.1: Convex vs. non-convex functions

The technique we can use to quickly find the global minimum is **gradient descent**. For convex functions we know, that we will always end up in the glocal minimum. Still, there is the open question of which direction to walk to:

Gradient descent

- Take the steepest way down, which is known since there is a derivative of the function.
- This leads to a lower point point on each step and therefore converges.

An important decision to make is the **step size**:

Step size

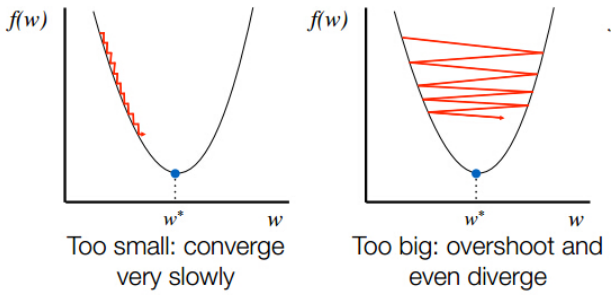


Figure 1.2: Step size for gradient descent

1.2 Multiple descriptive features

We now assume a feature consisting of multiple elements:

$$\begin{aligned}\mathbb{M}_{\mathbf{w}}(\mathbf{d}) &= \mathbf{w}[0] + \mathbf{w}[1]\mathbf{d}[1] + \cdots + \mathbf{w}[m]\mathbf{d}[m] \\ &= \mathbf{w}[0] + \sum_{j=1}^{m} \underbrace{\mathbf{w}[j]}_{\text{weight of the } i\text{-th feature } d[j]} \mathbf{d}[j] \\ &= \sum_{j=0}^{m} \mathbf{w}[j] \underbrace{\mathbf{d}[j]}_{\text{with } d[0]=1} \\ &= \underbrace{\mathbf{w} \cdot \mathbf{d}}_{\text{dot product of vectors}}\end{aligned}$$

In particular, this means we have $m+1$-dimensional vectors $\mathbf{d}_i$ and $\mathbf{w}$ with m as the number of features. This notational convenience extends the normal feature vector $\mathbf{d}'_i$ by $\mathbf{d}_i[0] = 1$. With n instances, we therefore have the error function:

$$L_2(\mathbb{M}_{\mathbf{w}}, \mathcal{D}) = \frac{1}{2}\sum_{i=1}^{n}(t_i - \underbrace{\mathbf{w} \cdot \mathbf{d}_i}_{=\mathbb{M}_{\mathbf{w}}(\mathbf{d}_i)})^2$$

With our now introduced multiple descriptive feature vector, we can write down the **sketch of** the overall **algorithm** of regression:

```
1  Require: set of descriptive features D
2  Require: learning rate α (controls how quickly algorithm converges)
3  Require: function Δ_error (determines the direction in which to adjust given weight w[i] to move
       down the slope of an error surface determined by D)
4  Require: convergence criterion (indicating when the algorithm has been completed)
5
6  w ← random starting point in weight space // randomly pick initial point
7  repeat
8    for each w[j] ∈ w do
9      // run downhill in steepest direction with speed α
10     w[j] ← w[j] + αΔ_error(D, w)[j]
11 until convergence occurs // stop when improvements become too small
```

Listing 1.1: Sketch of regression algorithm

For Δ_{error} we typically choose $\frac{\delta}{\delta \mathbf{w}[j]} L_2$. This means:

- If the derivative is positive, lower the weight $\mathbf{w}[j]$,
- If the derivative is negative, increase the weight $\mathbf{w}[j]$,
- While $\alpha > 0$ determines the speed in both cases.
- Line 10 in 1.1 would then result in $\mathbf{w}[j] \leftarrow \mathbf{w}[j] - \alpha \sum_{i=1}^{n}(t_i - \mathbf{w} \cdot \mathbf{d}_i)\mathbf{d}_i[j]$

1.3 Interpretation of results

For the one-feature case, the interpretation isn't difficult. We can simply see, that the target feature has some linear dependence on the descriptive feature. E.g., in our earlier example: the rental price has a "direct" dependency on the size.

It becomes more difficult when **interpreting results for multiple descriptive features**. This is due to the potentially completely different ranges of those features. The weights change dramatically when the units change (e.g. when changing cm^2 to m^2). This shows that the weight itself is irrelevant, only the sign has meaning.

An alternative approach is the **significance test**. Significance test

In a simple version of this, we have:

- Create a regression model using k descriptive features and measure the error Δ'.
- Create k regression models leaving out one descriptive feature at a time and measure the error Δ_i for $i \in [k]$.
- The difference in error $|\Delta' - \Delta_i|$ indicates the significance of feature i for $i \in [k]$

In a more complex way (no need to know details), we follow this approach:

- Null hypothesis: the feature does *not* have a significant effect on the model.
- Null hypothesis is rejected when *p-level* is too small $(1 - 5\%) \rightarrow$ a smaller p-value indicates a more important feature.
- In statistical hypothesis testing:
  - p-value or probability value: the probability of obtaining test results at least as extreme as results actually obtained during testing
  - Assumes a correct null hypothesis
  - Very small p-value means: such an extreme observed outcome is very unlikely under the null hypothesis

1.4 Handling categorical features

So far we assumed features (both descriptive and target) are continuous. In this chapter, we'll look at categorical descriptive and target features.

- E.g., {true, false}, {A, B, C}, or similar

For **categorical descriptive features**, we'll introduce $\{0, 1\}$-features for every possible value, which is called **one hot encoding**. One hot encoding

- E.g., for {A, B, C}: A= $(1, 0, 0)$, B= $(0, 1, 0)$, C= $(0, 0, 1)$

Next to one hot encoding, we have **single numeric value encoding**:

- Binary values {true, false} which can be translated to $\{0, 1\}$.
- Also, categorical variables with a clear order (ordinal) like {good, average, poor} can be translated to $\{1.0, 0.5, 0.0\}$.

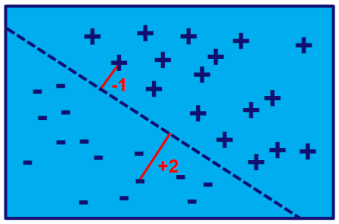
Possible issues are:

- Adding order to unordered categorical variables resulting in nonsense
  - E.g., simple encoding applied to country names maps numbers to countries (but no natural order exists)
- All encodings are approximations, so intermediate values will be considered possible by the "regression machine"
- One hot encoding discards dependencies

  - Say if A$= 1$ then logically B$= 0$, but B$= 0.66$ is also possible
- Approach may introduce many additional features, making the problem computationally expensive

For a **categorical target feature** we can try to derive a numerical (continuous) feature. The naive approach is to find a line separating the results (try to find a line s.t. $\mathbf{wd} = 0$). The line then performs more of a separation than a prediction.

x	y	class
2.1	2.0	+
3.1	1.4	+
1.2	0.8	-
2.9	3.3	+
0.8	1.1	-
...	...	..



Target feature is +/- label, and not the value on the y-axis

Figure 1.3: Seperating target feature (alternative to prediction)

This means, we can use the $\{0, 1\}$ idea for encoding the two catgories (+/-). The model then also produces either 0 or 1:

$$\mathbb{M}_{\mathbf{w}}(\mathbf{d}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{d} \geq 0 \\ 0 & \text{else} \end{cases}$$

After this step, do business as usual e.g. by minimizing the sum of squared errors.

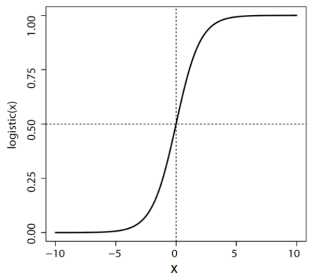
- Notice that in this naive approach, we don't use the distance to the decision boundary yet.
- But this would be desirable to make the decision surface continuous or smooth and thereby more applicable to gradient descent.

1.5 Logistic regression

Logistic regression will help us make a 0/1-decision continuous and smooth.

Logistic function

First, we need the **logisitic function**:



$$logistic(x) = \frac{1}{1 + e^{-x}}$$

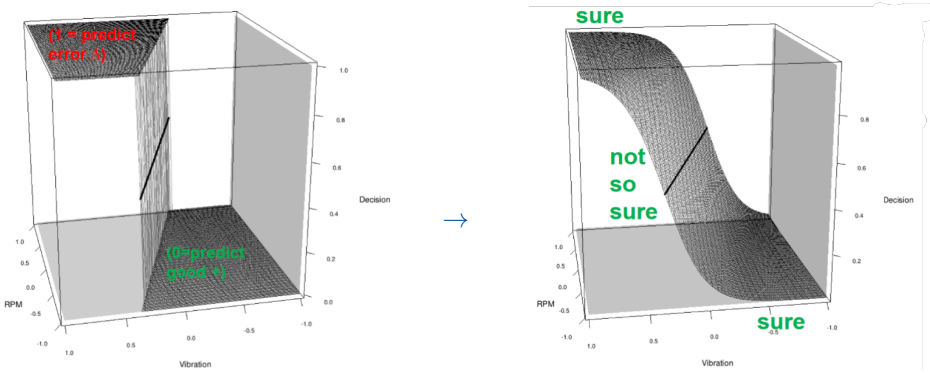
$$\frac{d}{dx} logistic(x) = logistic(x) \cdot \big(1 - logistic(x)\big)$$

Figure 1.4: Logistic function

- Here, $e = 2.7182818 \cdots$ is Eulers number

- Any value is mapped on a value between 0 and 1
  - $logistic(0) = 0.5$, $logistic(-\infty) = 0$, $logistic(+\infty) = 1$
- 0 and 1 are quickly approached, therefore we can use it as a "smooth binary value"

So now, we can use **logistic regression** instead of a hard 0/1-decision. Logistic regression



$$\mathbb{M}_{\mathbf{w}}(\mathbf{d}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{d} \geq 0 \\ 0 & \text{otherwise} \end{cases} \qquad \mathbb{M}_{\mathbf{w}}(\mathbf{d}) = logistic(\mathbf{w} \cdot \mathbf{d})$$

Figure 1.5: Difference 0/1 and logistic

The probabilistic interpretation looks as follows:

- $P[t =' \text{faulty}' \mid \mathbf{d}] = \mathbb{M}_{\mathbf{w}}(\mathbf{d})$
- $P[t =' \text{good}' \mid \mathbf{d}] = 1 - \mathbb{M}_{\mathbf{w}}(\mathbf{d})$
- And the system is more sure about the decision, the further it is away from the separating line.

1.6 Extensions (non-linear and multinomial)

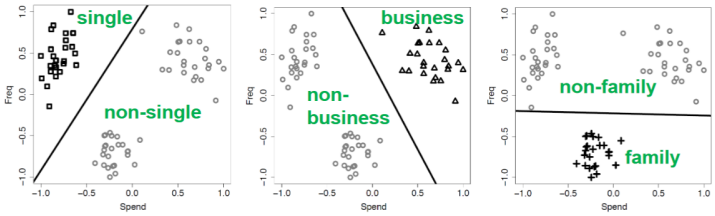
Non-linear functions that can't be separated by a line can still be handled by using linear machinery. The basic idea for that is to transform the data in advance, leading to the following formula: Non-linear relationship

$$\mathbb{M}_{\mathbf{w}}(\mathbf{d}) = \sum_{k=0}^{b} \mathbf{w}[k] \ \phi_k(\mathbf{d})$$

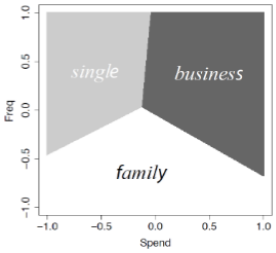
- Data in non-linear relationships is transformed *before* applying the linear machinery.
- For that, apply "basis functions" ϕ_k like $\phi_0(x) = 1$, $\phi_1(x) = x$, $\phi_2(x) = x^2$, etc.

For the case of **multinomial regression**, we look at the problem where categorical data is not binary. As a solution for n possible classes, we can split the decision into n models (per class i one model, classifying as "is class i" or "is not class i"). Those can then be combined back again into one model. Multinomial regression

(a) Original problem



(b) Binary classification



(c) Resulting model (multinomial classification)

Figure 1.6: Combining binary (one-versus-rest) models to multinomial regression