Linear Regression and Logistic Regression

April 17, 2020

Contents

List of Equations

Part I

Definitions

1 Definitions of symbols and notation

s**:** selection function, selects an element of a vector

$s(vec, i)$**:** select the i-th element of the vector vec

θ**:** (Greek letter lowercase »theta«) vector of weights (or »parameters«)

θ_n**:** the n-th vector of weights

$s(\theta_n, i)$**:** i-th element of the vector θ_n

$\hat{\theta}$**:** optimal vector of weights for the model

$h_\theta(s(x, i))$**:**

- hypothesis function for weights θ
- function, which makes predictions given a data point $s(x, i)$
- calculated by performing the matrix multiplication $\theta^T s(x, i)$

x**:** (vector of) data points (or »samples«) without labels

$s(x, i)$**:** the i-th data point, but with $s(s(x, i), 0) = 1$, to enable having an intercept or bias, a constant addend, which determins, where the the fit line intersects with the y-axis

$s(x, k)$**:** k-th feature of a data point

y**:** (vector of) correct labels (or »targets«, or »output variable«)

$s(y, i)$**:** the i-th data point's label

m**:** number of training data points (or »training examples«)

(x, y)**:** training data points (or »training examples«)

$(s(x, i), s(y, i))$**:** i-th training data point

$\frac{\delta}{\delta x} f$**:** (Greek letters lowercase »delta«) derivative of f with regard to x, the notation relates to the h-method $\frac{\delta}{\delta x} f = \lim_{h \to 0} \left(\frac{f(x+h) - f(x)}{h} \right)$, where h is the difference of 2 values of x, x_a and $x_b = x_a + h$ and that difference goes towards 0, to get towards ever more precise values of the slope at x_a.

$L(\theta)$**:** Likelihood, a conditional probability, see 2.

$\ell(\theta)$**:** log-likelihood, the logarithm of the likelihood: $\log(L(\theta))$

$P(A \mid B; v)$**:** The probability of A given B is parameterized by v.

$P(A \mid B; v) \sim$ **distribution:** The probability of A given B parameterized by v is distributed as expressed by the distribution »distribution«.

2 Probability and Likelihood

A likelihood is also a probability, but it is a conditional probability. With regard to machine learning, it is often used as in cases, where one wants to ask the following question: »How likely are we to see this data, assuming these parameters?« This is the same question as: »What is the probability to observe this data, if the generating process behind it follows the model, which is parameterized using these parameters?« For linear regression for example:

$$L(\theta) = P(y \mid x; \theta)$$

In natural language: The likelihood of our parameters θ is the probability of seeing the labels y, given the feature values x, parameterized with θ.

So the labels and feature values are fixed, and the probability is a function of the parameters θ.

3 Principle of Maximum Likelihood

Choose θ to maximize $L(\theta) = P(y \mid x; \theta)$.

Part II

Linear Regression

1 Introduction

Linear regression is a *parametric learning algorithm*. Parametric learning algorithms are algorithms, which have a fixed set of parameters, which are changed during a learning phase, to fit a model to some data. In contrast to *non-parametric learning algorithms* the number of parameters does not grow with the number of training data points.

2 Cost Function

To get a measure of error we would make, if we used a vector of weights θ_n to predict the values of y given the x, we calculate the difference between the predicted values of y and the correct values in the given data set. This is done for all data points and the differences are added up, so that an error in prediction influences the measure, no matter for which training data point the error in prediction is made.

However, not a simple sum is used, but the sum of the squared errors. This has the effect, that all summands are positive, only ever increasing the sum of errors and that bigger errors weigh more than small errors. Then the mean is taken of that sum of squared errors. This is often called *MSE*, which is the abbreviation for *mean squared error*.

$$\frac{1}{m} \sum_{i=0}^{m} \left((h_\theta(s(x,i)) - s(y,i))^2 \right) \tag{2.1}$$

This is not yet the final form of the cost function. A few modifications should be made to the measure, before defining it as the cost function.

The factor $\frac{1}{m}$ in front of the sum always divides by a constant m, which depends on the training data set.

Why use squared error?

If the sum of the plain differences was used errors could be positive or negative, depending on whether the predicted value is above or below the value in the actual data set. Such positive and negative errors could make up for each other, resulting in a sum, which is close to zero, misleading to think, that the chosen weights are good, while they actually cause big errors in prediction. It is therefore important for minimization of error, that all summed values are positive numbers.

One could think, that one could use the absolute errors instead of the squared errors. That is sometimes done as well. It is not affected as much by outliers, because error terms are not squared[1]. When errors are not squared, it is easier for multiple small errors to adding up to make the measure of quality look just as bad as few big errors, as they will be averaged later. However, using the absolute error makes the mathematics required later more difficult[2].

A more detailed reason is, that the mean squared error, given a few specific assumptions, follow naturally from maximizing the likelihood of θ. For more information see 1.

Cost Function Definition and Minimization

To find the weights $\hat{\theta}$, which minimize the cost (mean squared error), giving the best predictions, making the best model for the training data from which the model learns, we try to minimize the costs by changing the weights θ. This is mathematically written as follows:

$$\min_{\theta} J(\theta)$$

In natural language: *Minimize the value of J by trying values for the weights θ.*

Using mean squared errors as cost function $J(\theta)$:

$$\min_{\theta} \left(\frac{1}{m} \sum_{i=0}^{m} \left((h_\theta(s(x,i)) - s(y,i))^2 \right) \right)$$

For minimization, it does not matter, if we add a constant positive factor in front of the term one minimizes, because the factor will be applied to all resulting terms equally and will not change the sign: $\min_\theta(f(\theta,x)) = \min_\theta(k \cdot f(\theta,x))$, where $k \in \mathbb{R} \wedge k > 0$. This allows us to simplify the math later on, by adding a constant factor $\frac{1}{2}$ in front of the minimized term:

$$\min_{\theta} \left(\frac{1}{2} \cdot \frac{1}{m} \cdot \sum_{i=0}^{m} \left((h_\theta(s(x,i)) - s(y,i))^2 \right) \right)$$

Simplified:

$$\min_{\theta} \left(\frac{1}{2m} \cdot \sum_{i=0}^{m} \left((h_\theta(s(x,i)) - s(y,i))^2 \right) \right)$$

[1] Big errors become even bigger in comparison to small errors, when squared.

[2] There are more reasons for using squared errors instead of absolute errors, which I did not personally understand in detail and will not go into in this document, unless I understand them well one day.

Now we are ready to define the cost function $J(\theta)$ in its final form:[3] as follows (equal to the above):

$$J(\theta) = \frac{1}{2m} \sum_{i=0}^{m} \left((h_\theta(s(x,i)) - s(y,i))^2 \right) \tag{2.2}$$

3 (Batch) Gradient Descent

Now, that we defined the cost function J, we need a way to systematically change the values of the weights vector θ. We could select random values repeatedly, but we would have no guarantee to make progress that way, so we need something better than random values.

Gradient descent is an iterative algorithm, which will be used for minimizing the cost function. This version is often called »batch gradient descent«, because it looks at all training data points in each iteration and not only a subset of the training data points.

Algorithm

1. Choose an ϵ as tolerance.
2. Choose an initial weights vector θ_0.
3. Repeat until the following stop criteria is met:

$$|J(\theta_{n+1}) - J(\theta_n)| < \epsilon \tag{3.1}$$

   (a) Simultaneously[4] update all elements in θ_n by updating each weight with the respective partial derivative of the cost function as follows:

$$\theta_{n+1} = \theta_n - \alpha \frac{\delta}{\delta \theta_n} J \tag{3.2}$$

Explanation of the weights vector update formula

To calculate the new weight, we want to take advantage of the progress made towards the minimum costs. So we change the old values of θ_n instead of choosing independent new values for θ_{n+1}. In the update formula this is represented by substracting from θ_n on the right hand side of the update formula.

There is also the derivative of the cost function $\frac{\delta}{\delta \theta_n} J$ in the update formula. The first derivative is the function, which gives the slope of the cost function. To get to a (local, potentially global) minimum of the cost function, we want to follow the slope towards lower cost values, until the values are not getting much lower any longer. The slope depends on all weights in the weights vector θ_n. They tell us how much the slope depends on the feature values of a data points. They are the numbers for how much each feature weighs in. As such, an $s(\theta_n, k)$ determins, how much the value of the function J changes, when changing the corresponding $s(s(x,i),k)$ (i-th data point, k-th feature). This means, that the weight $s(\theta_n, k)$ determins, how much the cost function J is sloped in the direction of feature k.

Then there is the α as a factor for the derivative in the update formula. It is also known as the *step width*, or *learning rate*. It serves as a changing factor, for adjusting how much we move in the

[3] For convenience reasons, we will not write J as dependent on x and y. We will write only $J(\theta)$ instead of $J(\theta, x, y)$. J will be derived with regard to θ and not with regard to x or y anyway.

[4] »Simultaneously« means, that all elements in θ_n are updated at the same moment, after calculating each respective derivative. None of the updated weights take part in calculating the partial derivative of another updated weight. All updated weights are based on non-updated weights exclusively.

direction of the slope. This will help to prevent overshooting the minimum while moving along the slope. Initially we might want to make big steps towards the minimum, to make quick progress. When we almost arrived at the minimum, we want to make only small steps, to make sure, we are not overshooting the minimum, moving further away from the minimum, than we were before. If one wanted to work without an α one could set α to the neutral element of multiplication, which is 1[5] and it would have no effect. It only adds flexibility to the algorithm, by allowing a learning rate not equal to 1.

Derivation of the cost function

Having an intuition for the cost function, we still face the task of determining, what that derivative of the cost function is. Once this is expressed in operations easily performed by a computer, one can go about implementing the optimization.

The math involved with finding the derivative of the cost function is the reason, why we previously preprended an unexplained factor of $\frac{1}{2}$ to the cost function. The cost function contains a squaring, which will compliment the factor in the derivative. To find the derivative, we will look at the case of only having one training data point and then generalize.

The derivative then looks as follows:

$$\frac{\delta}{\delta\theta_n} J(\theta) = \frac{\delta}{\delta\theta_n} \frac{1}{2m} (h_\theta(x) - y)^2$$

The partial derivative with respect to the one available data point. There is no need for indices for x and y because there is only one data point. Also m is 1, but lets not simplify that, as we want to generalize to many data points later on and will need the m.

The structure of the term to derive calls for the so called *chain rule*:

If $f(x) = p(q(x))$, where q is the inner function and p is the outer function, then $\frac{\delta}{\delta x} f = \frac{\delta}{\delta x} p(q(x)) \cdot \frac{\delta}{\delta x} q(x)$.

For example: $f(x) = (x^a - b)^n$ then $u(x) = v^n$, where v is seen as a kind of blackbox, which depends on x, and $v(x) = x^a - b$ and the derivative is $\frac{\delta}{\delta x} f = \frac{\delta}{\delta x} u(v(x)) \cdot \frac{\delta}{\delta x} v(x)$.

u is the *outer function* and v is the *inner function*. We need $\frac{\delta}{\delta x} u$ and $\frac{\delta}{\delta x} v$ in addition to u and v to write down the derivative. In our example we have the term:

$$\frac{1}{2m} (h_\theta(x) - y)^2$$

The outer function is $u(v) = \frac{1}{2m}(v)^2$ and the inner function is $v(\theta, x) = h_\theta(x) - y$.

Since θ is not only one variable, but actually a vector, a partial derivative needs to be calculated with respect to each of the elements of θ. This is also what we need to perform the update of the single elements of the weights vector.

$$\frac{\delta}{\delta s(\theta, i)} u(v) = \frac{\delta}{\delta s(\theta, i)} \frac{1}{2m} (v)^2$$

According to the *power rule* for derivation: $f(x) = ax^n$ then $\frac{\delta}{\delta x} f = nax^{n-1}$. In other words: Multiply by the exponent and substract 1 from the exponent:

[5] I am not sure whether this would even work. Probably one would almost always overshoot the cost minimum, while trying to optimize the weights.

$$\begin{aligned}\frac{\delta}{\delta s(\theta,i)}u(v) = & \frac{\delta}{\delta s(\theta,i)}\frac{1}{2m}(v)^2 \\ = & 2\cdot\frac{1}{2m}v \\ = & \frac{2}{2m}v \\ = & \frac{1}{m}v\end{aligned}$$

v is again seen as a blackbox, dependent on $s(\theta,i)$ in the above transformations.

Now the derivative of $v(\theta,x) = h_\theta(x) - y$:

$$\frac{\delta}{\delta s(\theta,i)}v(x,\theta) = \frac{\delta}{\delta s(\theta,i)}(h_\theta(x) - y)$$

Substituting what $h_\theta(x)$ actually looks like:

$$\frac{\delta}{\delta s(\theta,i)}(h_\theta(x) - y) = \frac{\delta}{\delta s(\theta,i)}((s(\theta,0)\cdot s(x,0) + s(\theta,1)\cdot s(x,1)\ldots + s(\theta,n)\cdot s(x,n)) - y)$$

Taking partial derivative with respect to $s(\theta,i)$ will cause all terms of $h_\theta(x)$ without $s(\theta,i)$ part to disappear, because they do not depend on $s(\theta,i)$:

$$\begin{aligned}\frac{\delta}{\delta s(\theta,i)}v = & \frac{\delta}{\delta s(\theta,i)}((s(\theta,0)\cdot s(x,0) + s(\theta,1)\cdot s(x,1)\ldots + s(\theta,n)\cdot s(x,n)) - y) \\ = & \frac{\delta}{\delta s(\theta,i)}s(\theta,i)\cdot s(x,i) \\ = & s(x,i)\end{aligned}$$

So now we have u, v, $\frac{\delta}{\delta s(\theta,i)}u$ and $\frac{\delta}{\delta s(\theta,i)}v$:

$$\begin{aligned}u = & \frac{1}{2m}(v)^2 \\ \frac{\delta}{\delta s(\theta,i)}u = & \frac{1}{m}v \\ v = & h_\theta(x) - y \\ \frac{\delta}{\delta s(\theta,i)}v = & x_i\end{aligned}$$

Now we need to put the pieces together according to the chain rule. In case of the cost function J this means that the derivative $\frac{\delta}{\delta s(\theta,i)}J$ is $\frac{\delta}{\delta s(\theta,i)}u(v(\theta))\cdot\frac{\delta}{\delta s(\theta,i)}v(\theta)$:

$$\begin{aligned}\frac{\delta}{\delta s(\theta,i)}J(\theta) = & \frac{\delta}{\delta s(\theta,i)}\frac{1}{2m}(h_\theta(x) - y)^2 \\ = & \frac{\delta}{\delta s(\theta,i)}u(v(x))\cdot\frac{\delta}{\delta s(\theta,i)}v(x) \\ = & \frac{1}{m}(h_\theta(x) - y)\cdot x_i\end{aligned}$$

So the θ_{n+1}according to the update formula of gradient descent is then:

$$\begin{aligned}\theta_{n+1} = & \theta_n - \alpha\frac{\delta}{\delta\theta_n}J(\theta) \\ = & \theta_n - \alpha\left(\frac{1}{m}(h_\theta(x) - y)\cdot x_i\right)\end{aligned}$$

4 Stochastic Gradient Descent (TODO: rewrite)

(incremental gradient descent)

Only use a subset of the data points for the update of the weights:

- Repeat until convergence
  - for $j = k$ to m: (Q: At next step a different set of data points? Q: Randomly chosen or in parts?)
    - for all i:

$$\theta_{i+1} = \theta_i - \alpha \cdot \sum_{j=1}^{m} \left(h_\theta \left(x^{(j)} \right) - y^{(j)} \right) \cdot x_i^{(j)}$$

This algorithm is especially useful for training data sets with many data points, for which batch gradient descent might be a too slow.

5 Predictions

After optimizing the weights for minimized costs, meaning minimized error in prediction for values in the training data set, we can make predictions for completely new data points. This is simply done by using the optimized weights in the hypothesis function:

$$h_{\hat{\theta}}(x_{\text{new}}) = \theta^T x_{\text{new}} = y_{\text{pred}}$$

Part III

Logistic Regression

1 Maximum Likelihood

In this section, we will explore maximum likelihood and how it is connected to mean squared error.

Lets make some assumptions.

1. The label $s(y, i)$ can be represented by a linear function of the features $s(x, i)$, plus some kind of epsilon $s(\epsilon, i)$, called *error term*: $s(y, i) = \theta^T s(x, i) + s(\epsilon, i)$. Such error term $s(\epsilon, i)$ can model, deviations (errors in prediction) from that linear function. Reasons for such deviations could be for example:
   (a) There are more features of the observed objects, which were not captured in the data set, which cause slight errors to occur, when the predictions are made without them in the features of the data set.
   (b) The function, which is actually produced the observations captured in the data set, is not linear, so using a linear function for prediction will most likely result in errors.
2. The errors made up for by the error term $s(\epsilon, i)$ are distributed in a Gaußian distribution[6], meaning, that bigger errors are less likely than smaller errors: $s(\epsilon, i) \sim \mathcal{N}(\mu, \sigma^2)$ with $\mu = 0$.

[6] Why assume Gaußian? Apparently when looking at many linear regression problems, the error is in fact distributed like this and that makes it a good assumption and the other reason is, that it seems to make the involved math easier (source: Andrew Ng Stanford Machine Learning lecture). So really no other reason than counting cases and math.

Then the probability for an $s(\epsilon, i)$ is given by $s(\epsilon, i) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{s(\epsilon,i)^2}{2\sigma^2}\right)$. Then $P(s(y,i) \mid s(x,i); \theta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(s(y,i)-\theta^T s(x,i))^2}{2\sigma^2}\right)$ (inserting $s(y,i)$ into the probability function, θ is not seen as a random variable, only as an unknown, spoken as »parameterized by θ«), which is the same statement as $s(y,i) \mid s(x,i); \theta \sim \mathcal{N}(\theta^T s(x,i), \sigma^2)$.

3. The $s(\epsilon, i)$ are independently but identically distributed (they are IID).[7]

With these assumptions in place proceed.

Then the likelihood of θ is written as $L(\theta)$, where $L(\theta) = P(y \mid x; \theta)$, the probability of seeing the labels y, given the features of the data points x, parameterized by the weights θ.

If one calculates the probabilities for all training data points, those independent probabilities need to be multiplied together[8]. So the likelihood of θ, which is the probability of the data parameterized by θ, is given as follows:

$$\begin{aligned} L(\theta) &= \prod_{i=1}^{m} \left(P(s(y,i) \mid s(x,i); \theta)\right) \\ &= \prod_{i=1}^{m} \left(\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(s(y,i) - \theta^T s(x,i))^2}{2\sigma^2}\right)\right) \end{aligned}$$

For mathematical convenience we take the logarithm of the likelihood. This will later extinguish the exponentiation of the Gaußian and simplify the formula. We can do this, because we will later try to maximize the likelihood and taking the logarithm does change the result of maximization, because logarithm is a monotonically increasing function. This logarithm of the likelihood is called *log-likelihood* and is written $\ell(\theta)$.

$$\begin{aligned} \ell(\theta) &= \log\left(L(\theta)\right) \\ &= \log\left(\prod_{i=1}^{m} \left(P(s(y,i) \mid s(x,i); \theta)\right)\right) \\ &= \log \prod_{i=1}^{m} \left(\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(s(y,i) - \theta^T s(x,i))^2}{2\sigma^2}\right)\right) \end{aligned} \tag{1.1}$$

Logarithm of a product is the same as the sum of logarithms of the factors:

$$\begin{aligned} \log\left(L(\theta)\right) &= \log \prod_{i=1}^{m} \left(\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(s(y,i) - \theta^T s(x,i))^2}{2\sigma^2}\right)\right) \\ &= \sum_{i=1}^{m} \left(\log\left(\frac{1}{\sqrt{2\pi}\sigma} \exp(\cdots)\right)\right) \end{aligned}$$

According to the product rule for logarithms: $\log_b(xy) = \log_b(x) + \log_b(y)$. So we can split up the logarithm inside the sum:

[7] TODO: I am not sure, why this assumption is important. It was only mentioned in the lecture I extracted this knowledge from, so I guess it is important. I mean, the assumption of identically distributed makes sense, because otherwise one cannot easily write down a formula for the probability distribution, but why does it need to be independently distributed?

[8] Adding them all up makes no sense and could result in probabilities > 1.

$$\sum_{i=1}^{m}\left(\log\left(\frac{1}{\sqrt{2\pi}\sigma}\exp(\cdots)\right)\right) = \qquad \sum_{i=1}^{m}\left(\log\left(\frac{1}{\sqrt{2\pi}\sigma}\right)+\log\left(\exp(\cdots)\right)\right)$$

This means, that we are adding the constant $\log\left(\frac{1}{\sqrt{2\pi}\sigma}\right)$ in total m times and since it is a constant, we can move it out of the sum:

$$\sum_{i=1}^{m}\left(\log\left(\frac{1}{\sqrt{2\pi}\sigma}\right)+\log\left(\exp(\cdots)\right)\right) = \qquad m\cdot\log\left(\frac{1}{\sqrt{2\pi}\sigma}\right)+\sum_{i=1}^{m}\left(\log\left(\exp(\cdots)\right)\right)$$

log and exp are each other's inverted operations, so they negate each other:

$$\begin{aligned} m\cdot\log\left(\frac{1}{\sqrt{2\pi}\sigma}\right)+\sum_{i=1}^{m}\left(\log\left(\exp(\cdots)\right)\right) = & \qquad m\cdot\log\left(\frac{1}{\sqrt{2\pi}\sigma}\right)+\sum_{i=1}^{m}(\cdots) \\ = \quad & m\cdot\log\left(\frac{1}{\sqrt{2\pi}\sigma}\right)+\sum_{i=1}^{m}\left(-\frac{(s(y,i)-\theta^T s(x,i))^2}{2\sigma^2}\right) \end{aligned}$$

Since we are choosing θ to maximize the likelihood, the other symbols are seen as constants. This means, that for maximizing the likelihood, the part $m\cdot\log\left(\frac{1}{\sqrt{2\pi}\sigma}\right)$ is actually a constant and does not change, when we change θ. That means it does not matter for the whole maximization and we can simply remove it:

$$\begin{aligned} &\max_\theta\left(m\cdot\log\left(\frac{1}{\sqrt{2\pi}\sigma}\right)+\sum_{i=1}^{m}\left(-\frac{(s(y,i)-\theta^T s(x,i))^2}{2\sigma^2}\right)\right) \\ =&\max_\theta\left(\sum_{i=1}^{m}\left(-\frac{(s(y,i)-\theta^T s(x,i))^2}{2\sigma^2}\right)\right) \end{aligned}$$

Here we maximizing a sum of something with a negative sign in front. This is equivalent to minimizing the sum with the negative sign removed:

$$\max_\theta\left(\sum_{i=1}^{m}\left(-\frac{(s(y,i)-\theta^T s(x,i))^2}{2\sigma^2}\right)\right) = \qquad \min_\theta\left(\sum_{i=1}^{m}\left(\frac{(s(y,i)-\theta^T s(x,i))^2}{2\sigma^2}\right)\right)$$

Furthermore, we can simplify the denominator, as it only divides each addend by a constant $2\sigma^2$:

$$\min_\theta\left(\sum_{i=1}^{m}\left(\frac{(s(y,i)-\theta^T s(x,i))^2}{2\sigma^2}\right)\right) = \min_\theta\left(\sum_{i=1}^{m}\left(\frac{(s(y,i)-\theta^T s(x,i))^2}{2}\right)\right)$$

Since every addend is divided by the same denominator, we can also divide the whole sum instead of its parts (distributive property):

$$\min_\theta\left(\frac{\sum_{i=1}^{m}\left((s(y,i)-\theta^T s(x,i))^2\right)}{2}\right)$$

Note, that the value of σ does not matter for the minimization or maximization at all. We will arrive at the same values for θ, no matter what the variance of the Gaußian is.

The minimized term is the same as the not averaged cost function $J(\theta)$:[9]

$$\frac{\sum_{i=1}^{m} \left((s(y,i) - \theta^T s(x,i))^2\right)}{2} = \frac{1}{2} \cdot \sum_{i=1}^{m} \left((s(y,i) - \theta^T s(x,i))^2\right)$$

What does it mean, that the log-likelihood with the assumption of (IID) Gaußian distribution of the error terms is the same as the cost function $J(\theta)$? It means, that using the mean squared error, on which the cost function is based, comes naturally under the given assumptions. With these assumptions using mean squared error is mathematically justified.

2 Scoring Function

To determin a class for a data point we need a way to calculate some kind of value from the feature values of a the data point. A function doing this is called a *scoring function*:

$$\text{score}(x_i)$$

3 Weights Vector

The scoring function need so operate on the feature values or a subset of them. In linear regression we use one weight per feature, which gives us a weight vector w_k. This weight vector decides, how much the value of a feature influences the overall score. The weights vector will need to be learned in the learning phase of the model. As such the w_k will be updated, possibly many times, before arriving at the optimal weights vector $\hat{w}$. The index k stands for the current weight vector. Single weights of a w_k will be expressed by using square brackets: $w_1[j]$, where j is the feature index for the j-th feature.

4 Feature Function

Furthermore the scoring function can operate on the actual feature values or apply a transformation on them before we use them. Such a transformation for one feature is expressed by a *feature function* h_j for the j-th feature:

$$h_j(x_i)$$

The feature function for all features of a data point x_i is then:

$$h(x_i)$$

[9] We defined the cost function $J(\theta)$ earlier as averaged by dividing by m, however, this averaging is not required. So with a slightly different definition of $J(\theta)$, one could say that it is even the same as the cost function $J(\theta)$.

5 Scoring Function in Detail

To apply the weighting of features given by the weights vector to the features and apply any chosen feature function, we multiply the transposed (indicated by $\square^T$) weight vector with the feature function of the features:

$$\text{score}(w_k^T \cdot h(x_i)) \in \mathbb{R} \cup \{-\infty, +\infty\}$$

This is nothing other but the following sum:

$$\sum_{j=0}^{n} (w_k[j] \cdot h_j(x_i))$$

For linear regression we would be done at this step. The score is the value, which the model predicts. For logistic regression, we need to do more, because we need to make a decision about a class assignment for data points based on their score. The actual classification part still needs to be done.

6 Estimate of the Probability of a Class

To classify a data point we need to calculate from the score of the data pointan estimate of how likely the data point is in a class. This will only be an estimate, because the real process behind the creation of the data might not be a linear process. Probabilities lie between 0 and 1. However, the scoring function gives real numbers between $-\infty$ and $+\infty$. This is why we need yet another function to map from the target set of values of the scoring function to $(0; 1)$. The so called *link function*:

$$\hat{P}(y = 1|x_i) = \text{link}(\text{score}(x_i)) = g(\text{score}(x_i))$$

($\hat{P}$ stands for »the best possible estimate from this model«, where P is only one estimate of probability.)

The link function g for logistic regression is the *sigmoid function*:

$$\text{sigmoid}(v) = \frac{1}{1 + e^{(-v)}}$$

Applied to the score of a data point:

$$\text{sigmoid}(\text{score}(x_i)) = \frac{1}{1 + e^{(-\,\text{score}(x_i))}}$$

If the value of the link funktion is > 0.5 it means, that the probability the model estimates, that the data point is in class 1, is higher than 50%, which means the model classifies the data point as part of class 1.

Part IV

Optimization of Weights

Now that we know, how the probability of a data point belonging to a class is estimated by the model, we still need a way to get the optimal weights, which the score is based on. We need a way to optimize them based on the data points presented to the model during the learning phase. Optimizing the weights is what actually makes the learning of the model. The result will be $\hat{w}$ the optimal weights.

What we need are three things:

1. an initial vector of weights w_0
2. a function for calculating how good a vector of weights is (quality measure q) or a function for calculating how much error we make when using the weights vector to make predictions (error measure, cost function)
3. a way of updating the weights, so that there is a chance of getting better weights

1 Initial vector of weights

The initial vector of weights can be chosen at random or set to some predefined value.

2 Quality Measure

Lets enumerate things we can deduce about a quality measure q for the weights vector:

- q should measure the effect the weights vector has on the score for all data points, which are use to learn the model. This is the effect on predictions made by the model using the weights vector. If q did not consider all data points, we would risk missing errors in prediction for some data points.
- q should return values in a within known bounds, otherwise we do not know how to interpret the values. We would not know, whether a value expresses a good quality or a bad quality of the weights vector.
- q should somehow relate to the results of $w^T \cdot h(x_i)$ (the score) for all $x_i \in X$.
- q should give a value meaning good quality, if both of the following are true for most data points:
    - w_k causes the result of the link funktion of the score to be closer to 1, if the data point really is of class 1.
    - w_k causes the result of the linkk funktion of the score to be closer to 0, if the data point really is of class 0.
- q should give a value meaning bad quality, if both of the following are true for most data points:
    - w_k causes the result of the link funktion of the score to be closer to 0, if the data point really is of class 1.

- w_k causes the result of the link funktion of the score to be closer to 1, if the data point really is of class 0.

So what could q be?

We can calculate the probabilities for all data points, which the model estimates using the current weights vector w_k, where y_i is the function giving the correct class of a data point, according to the labels in the data set, the class, which the data point according to observation really has:

$$P(y = y_i \mid x_0, w_k)$$

$$P(y = y_i \mid x_1, w_k)$$

$$P(y = y_i \mid x_2, w_k)$$

$$\vdots$$

All those probabilities are between 0 and 1. To be more precise, they cannot be exactly 0 or 1, because the link function, sigmoid never really becomes 0 or 1 for any value $\in \mathbb{R}$. This means, that their product will also be a value in the interval $(0; 1)$. We could use the product of all estimated probabilities as a quality measure:

$$\mathrm{q}(X, w_k) = \prod_{i=0}^{N} (P(y = y_i \mid x_i, w_k))$$

In natural language: *»The quality* q *of a given weight vector w_k for a data set X is the product of estimates of probability for all data points x_i in X.«*

(This is also called the »data likelyhood«.) Things to note about this product:

- If all the probabilities were perfectly estimated and close to 1 for the correct classes, the result of this product would also be close to 1.
- If the probability are badly estimated by the weights and close to 0 for the correct classes, always predicting the wrong class, the result of the product would also be close to 0.
- Furthermore this considers all data points as required.
- q relates to the score of a data point, because the scores are used to calculate the estimate for the probabilities, which q makes a product of.
- q outputs values in the interval $(0; 1)$, if we only ever allow weights $w_k[j] > 0$.

It seems, that this product of estimates of probabilities satisfies all of our requirements.

We would want this $\mathrm{q}(X, w_k)$ to be as close to 1 as possible, as far away from 0 as possible. This means we need to maximize it. What we are looking at is an optimization problem, more specifically a maximization problem:

$$\max_w(q(X, w_k)) = \max_w \left(\prod_{i=0}^{N} \left(\frac{1}{1 + e^{(-\operatorname{score}(x_i))}} \right) \right)$$

The result will be $\hat{w}$. To get $\hat{w}$ will be the main goal of the linear regression or logistic regression algorithm.

To maximize the quality of the weights, we can use the so called *gradient ascent* algorithm.

Weights Vector Update Function

To maximize the quality of the predictions of the model, the weights need to be optimized. We need a formula for iteratively updating the weights. The formula we need is:

$$w_{k+1}[j] = w_k + \alpha \cdot \frac{d\,\mathrm{q}(X, w_k)}{dw_k[j]}$$

In natural language: *»The weights vector w_k is updated by the derivative of the quality measure (slope in the quality measure plot landscape) with regard to the current weights.«*

However, let us get there step by step, so that the formula is explained.

1. We move stepwise from the current values of w_k to a hopefully better w_{k+1}, w_{k+2}, w_{k+3} and so on, where the next value is always based on the previous value. Each vector of weights stores some progress on our way to the top of the hill. The next values of the weights vector are based on the current values, to make use of the already made progress. That is why the current weights are part of the updated value in the formula. In theory one could take completely arbitrary updated weight values, but that would not guarantee progress towards the top of the hill.
2. In general the way to find the maximum or minimum of a function is to take its derivative and find the point, where the slope is 0, because that will be in a valley (local minimum) or on top of a hill (local maximum) visually.
3. If we take the derivative of the quality measure and walk in its direction, we will walk towards the (local) optimum. We can imagine this visually by imagining standing on a slope of a hill. The mathematical slope of the hill at that point will point downwards and upwards the hill. If we move upwards in the direction it is pointing, we are bound to move a bit higher. If we repeat this process multiple times, we are bound to arrive at higher and higher places.
4. Wanting to find an optimum and walking in the direction of the slope together mean, that we need to take the derivative of the quality measure, because the derivative gives us the slope at all points.
5. The slope and thereby the direction of the slope is determined by the derivative of the quality measure. However, there are multiple derivatives one can derive from the quality measure function, because there are multiple independent weights in the weights vector. For each of the weights one can derive a partial derivate. To get closer to the local (potentially global) maximum of the quality measure function, one needs to consider all the weights, which means deriving all the derivatives and updating all the weights. This is called a *total derivative*. This will result in a step in the direction of the slope of the quality measure function. So this is why we take partial derivatives and change each weight according to its partial derivative.
6. TODO: Why do we need a step width α?
7. For each weight in w_k we need to figure out, what the . We need to update each of the weights in w_k (so all $w_k[j]$) based on the derivative with regard to weight itself. This is called partial derivative.

TODO

3 Cost Function

An alternative to the data likelyhood is making use of a cost function, whose interpretation is not, how good the weights vector w_k is, but instead how bad it is, by calculating the error in prediction, which is made by the model using the weight vector w_k. One would want to minimize error in prediction to get the best weights. This means, that the optimization problem using a cost function becomes a minimization problem, instead of a maximization problem.

Weights Vector Update Function

TODO

TODO: pipes in math formulas -> correct tex math character

Part V

Gradien Ascent Algorithm

Part VI

Miscellaneous

1 Model Persistence

If one stores the learned weights vector w_k, one basically persists the linear regression or logistic regression model, because the usage of the weights vector is defined in the algorithm.