3. Linear regression

Machine Learning 1, UNIZG FER, AY 2022/2023

Jan Šnajder, lectures, v1.13

Last time we introduced the basic concepts of machine learning: the **hypothesis**, which is a function that maps from input data to labels and is defined by parameters theta, and the **model**, which is a set of hypotheses indexed by parameters theta. We have said that machine learning comes down to finding the best hypothesis in the model, i.e., finding optimal parameters. In doing so, we take as the criterion the error of the hypothesis, which we measure on a set of labeled examples, and we call this measure the **empirical error**. We concluded that every ML algorithm has **three components**: a model, a loss function (the expectation of which is the error function), and the optimization procedure.

Today we will look into one specific machine learning algorithm for **regression**. Recall that regression means predicting numerical values, so the labels $\mathcal{Y}$ are numbers. For example, predicting the prices of apartments in Boston. We will focus on one particularly important algorithm, namely **linear regression model**, which is very important in both machine learning and statistics.

In statistics, regression primarily serves to **explain relationships** between two sets of variables and for statistical inference (e.g., confidence intervals, test of statistical significance of predictors). In machine learning, our interest with regression is primarily for making **predictions**, while explaining data is generally of less interest to us. The ramification is that statistics and machine learning approach regression from slightly different angles, although eventually they talk about the same thing.

Today's lecture is structured so that we look at the simplest scenario first, before we move on to more complex scenarios.

1 Simple regression

As a motivating example, imagine us building a model that predicts the price of an apartment based on its features. Obviously, this is a regression problem because we wish to predict a numerical value. Because we're dealing with supervised machine learning here, we also need labeled data. In general, we have at our disposal a labeled dataset, $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}$, where examples are n-dimensional feature vectors, $\mathbf{x} \in \mathbb{R}^n$, and labels are real numbers, $y \in \mathbb{R}$. In our case, the y labels are the prices of apartments in Boston: for each property we have a number of features $\mathbf{x}$ that describe that property, , and we also have its price y in dollars. We now want to build a model that will predict the price of a new, yet unseen property.

In regression, the hypothesis h maps examples to labels, hence $h : \mathbb{R}^n \to \mathbb{R}$, that is, it maps from an input space (instance space) to numeric values. For example, given a property described by features $\mathbf{x}$, the hypothesis h will calculate its price, $h(\mathbf{x})$. In regression, we call the variables that make up the vector $\mathbf{x}$ the **input variables** (or **independent variables** or **predictor variables** or **covariates**), and the output y the **output variable** (or **dependent variable** or **criterion variable**).

Today our focus will be on **linear regression**. In linear regression, output is a linear combination of input variables. By choosing such a model, we implicitly assume that the output variable is linearly dependent on the input variables. Our model (i.e., a set of hypotheses defined up to the parameters) is defined as follows:

$$h(\mathbf{x}; \mathbf{w}) = w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n$$

In regression, the vector of the model parameters is typically denoted by $\mathbf{w}$ instead of $\boldsymbol{\theta}$. Parameters $w_0, \ldots, w_n$ determine the impact that each feature has on the value of the output variable. In machine learning, these parameters are often called **weights**, while in statistics they are called **regression coefficients** or **beta-coefficients**, and are denoted by $\beta_0, \ldots, \beta_n$.

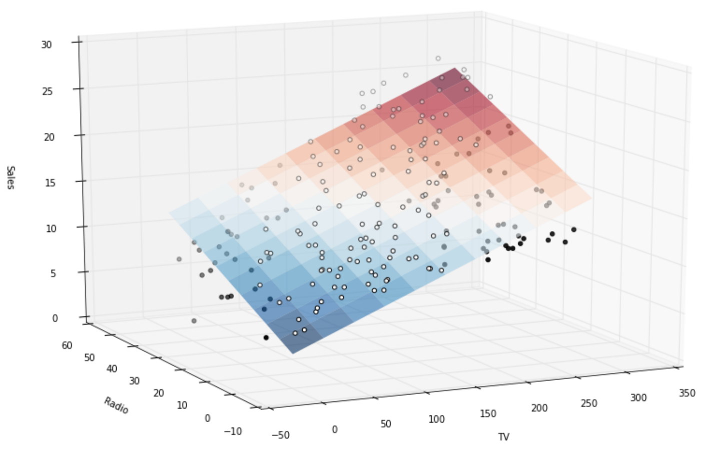
For a one-dimensional input space, $n = 1$, the hypothesis will be a **line**. This is the so-called **simple regression** (hrv. *jednostavna regresija*). It is the simplest model of linear regression, where we have only one input variable:

$$h(x; w_0, w_1) = w_0 + w_1x$$

▸ EXAMPLE

- (1) Savings by age (growing more or less linearly)
- (2) Car price in terms of mileage (falling sharply)

If $n = 2$, then the hypothesis describes a **plane** in vector space that is defined by the domain $\mathcal{X} = \mathbb{R}^2$ and the codomain $\mathcal{Y} = \mathbb{R}$ (more precisely, the vector space is obtained as the Cartesian product of these two sets). This looks as follows:



If $n > 2$, the hypothesis h describes a **hyperplane** in the vectors space $\mathbb{R}^n \times \mathbb{R}$. The position of this hyperplane is determined by the weight vector $\mathbf{w}$.

Let us now take a closer look at the **linear regression algorithm**. First, recall that every machine learning algorithm has to define three components: a model, a loss function, and an optimization procedure. For simplicity, let's limit ourselves to simple regression, $n = 1$. We have already defined the model. What we still need is the loss function and the optimization procedure.

Let's start with the **loss function**. Recall that the loss function is the error that the hypothesis makes on each individual example x, if the correct label of the example happens to be y. The loss function (denoted L) typically used for regression is the **squared difference** between the target and the predicted value:

$$L\big(y, h(x)\big) = (y - h(x))^2$$

In statistics, the difference between the target and the predicted value, $y - h(x)$, is called the **residual**. We can thus say that the loss function is equal to **the square of the residual**.

Here we can legitimately ask ourselves: why should one use specifically the squared difference for the loss function? Why not merely the difference between the target and the predicted value? Because a loss is a loss, regardless of direction. If we only calculated the differences, the positive and negative losses would cancel each other out, and we don't want that. All good, you say, but couldn't we then use the absolute value of that difference? The answer to this question lies in the need to optimize the error function. Namely, the absolute value function is not differentiable, whereas the squared difference function is. If the loss function is not differentiable, then the error function – defined as the expectation of the loss function – will certainly also not be differentiable. For this reason, we prefer the squared difference. However, we'll see later that there are deeper reasons to use the squared difference.

What about the **error function**? Let us recall that the loss and error are not the same. In the previous lecture, we explained that that the error of a hypothesis is actually the **expected value of the loss of the hypothesis**, which we empirically calculate as the average value of the loss function on a set of labeled examples. We will likewise define the error of a regression hypothesis:

$$E(h|\mathcal{D}) = \frac{1}{2}\sum_{i=1}^{N} L(y^{(i)}, h(x^{(i)})) = \frac{1}{2}\sum_{i=1}^{N}\big(y^{(i)} - h(x^{(i)})\big)^2$$

with the difference that instead of the average, i.e., instead of $1/N$, we use $1/2$ for later mathematical convenience. This slight change, however, has no effect whatsoever on the optimization procedure: the minimizer (i.e., the value for which the function assumes a minimum) will be the same, regardless of the constant by which we multiply the loss function.

The third component of the regression algorithm is the **optimization procedure**: the search for a hypothesis in the model, $h \in \mathcal{H}$, that minimizes the empirical error. That is:

$$h^* = \underset{h\in\mathcal{H}}{\operatorname{argmin}}\, E(h|\mathcal{D}) = \underset{h\in\mathcal{H}}{\operatorname{argmin}}\,\frac{1}{2}\sum_{i=1}^{N}\big(y^{(i)} - h(x^{(i)})\big)^2$$

Since hypothesis h is indexed by parameters (w_0, w_1), what we are actually looking for are the model parameters that minimize the error, i.e.:

$$(w_0, w_1)^* = \operatorname{argmin}_{w_0,w_1}\frac{1}{2}\sum_{i=1}^{N}\big(y^{(i)} - (w_1 x^{(i)} + w_0)\big)^2$$

Let us consider this optimization problem from a mathematical point of view. This is a function of two variables and we want to find its minimum. What line of attack comes to mind first? Well, we should find the roots of this function: find the derivative, equate it with zero, and solve the system of equations. (Note that this is a convex function, with only one root, which will certainly be the minimum of the function.) Let's show what this would look like. We calculate the derivative of the error function and equate it with zero:

$$\nabla_{w_0,w_1} E(h|\mathcal{D}) = 0$$

Since this is a function of two variables, we will calculate the partial derivatives by the first parameter (w_0) and by the second parameter (w_1) and equate them with zero:

$$\frac{\partial}{\partial w_0}\Big[\frac{1}{2}\sum_{i}^{N}\big(y^{(i)} - (w_1 x^{(i)} + w_0)\big)^2\Big] = 0$$

$$\frac{\partial}{\partial w_1}\Big[\frac{1}{2}\sum_{i}^{N}\big(y^{(i)} - (w_1 x^{(i)} + w_0)\big)^2\Big] = 0$$

After a few steps, which we will skip here because they are not particularly interesting, we arrive at the following solution:

$$
\begin{aligned}
w_0 &= \bar{y} - w_1\bar{x} \\
w_1 &= \frac{\sum_i^N x^{(i)}y^{(i)} - N\bar{x}\bar{y}}{\sum_i^N (x^{(i)})^2 - N\bar{x}^2}
\end{aligned}
$$

where $\bar{x}$ and $\bar{y}$ are the mean values of the x and y, respectively, across all examples in the set $\mathcal{D}$.

Two things are worth noting here. The first is that our optimization has been reduced to simply plugging in numbers into a formula: if I want to find parameters (w_0, w_1) that minimize the quadratic error function, all I have to do is apply the above formulas. For problems like this one, where the minimizer is given by a formula and optimization comes down do plugging in numbers into the formula rather than doing some heuristic search, we say they have a **closed form solution** (hrv. *rješenje u zatvorenoj formi*). This means that the solution is one mathematical expression that makes use of "widely accepted" functions and operators (the logarithm, the root, trigonometric functions, etc.) and can be computed in a finite number of steps. In machine learning, we quite like when our optimization problem has a closed-form solution because it usually means that the optimization is simple and exact.

The other thing I want you to notice is that the formulas for w_0 and w_1 are actually different, and that the formula for w_1 is actually pretty ugly. What you may not notice here, but you could easily see for yourself if you try to do it, is that for a three-parameter model, (w_0, w_1, w_2), we would get more complex formulas, for a four-parameter model even more complex, etc. This is far from ideal, since we prefer to have a general solution for optimizing model's parameters, in the sense that the formula does not depend on the number of parameters (a specific solution, of course, must depend on the specific number of parameters as it depends on the specific set of labeled examples). You might already suspect that the way to achieve such a generality would be to express the optimization problem in matrix form (because then the solution will then be valid for design matrices and label vectors of arbitrary dimensions). And, indeed, this is what we will do (a few pages further below).

2 Types of regression

After looking at this simple case of regression, let's try to make our approach more general. We dealt with simple regression, which has only one input variable. This is only of limited use: the really interesting real-life phenomena generally cannot be well described by only one independent variable. Luckily, simple regression is just one type of regression. There are other types of regression. In particular, if one considers the number of **input (independent, predictor)** variables, we can have **single regression** (or **simple regression**), where $n = 1$ (there is only one feature at the input) or **multiple regression** (or **multiple regression**), where $n > 1$ (each example is described with multiple features). On the other hand, if one looks at the number of **output (dependent, criterion)** variables, we can have **univariate regression** (or **single output regression**), where $h(\mathbf{x}) = y$, or **multivariate regression** (or **multi-output regression**), where $h(\mathbf{x}) = \mathbf{y}$ (the output variable is also a vector). When we combine these two dimensions, we get four types of regression, as follows:

	Single output (y)	**Multiple output** $(\mathbf{y})$
One input (x)	(Univariate simple)	Multivariate simple
Multiple inputs $(\mathbf{x})$	(Univariate) multiple	Multivariate multiple

Regression is typically assumed to be univariate; if this is not the case, then it is good to explicitly say that it is multivariate. Therefore, univariate multiple regression in the literature is often simply referred to as **multiple regression** (engl. *multiple regression*).

We will focus on **multiple regression** in this course, since it is the most commonly used and corresponds best to the classification models that we will deal with later. So, we will focus on regression with multiple inputs and with a single output, i.e., for each example we predict one number. For example, the price of real estate in Boston, where the property is described by a number of features (number of rooms, distance from the main road, whether there is an elevator, whether the neighbors are bearable, etc.). Univariate simple regression is, of course, just a special case of univariate multiple regression where $n = 1$.

3 Three components of a linear regression algorithm

Let us now look again at the three basic components of the linear regression algorithm, this time for the case of multiple regression, i.e. when $n > 1$.

1. **Model**

$$h(\mathbf{x};\mathbf{w}) = w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n = \sum_{j=1}^{n} w_jx_j + w_0$$

   As we have already suggested above, calculations will be much simpler if we move to matrix calculus. To this end, we will expand the vector of each example, $\mathbf{x}^{(i)}$, with the so-called **dummy feature**, $x_0^{(i)} = 1$. The value of this feature will always be equal to 1, and serves merely to be multiplied with weight w_0 so that all weights can be treated equally. The model is then:

$$h(\mathbf{x};\mathbf{w}) = \mathbf{w}^{\mathrm{T}}\mathbf{x}$$

   where $\mathbf{w}^{\mathrm{T}}\mathbf{x}$ is the scalar product of the vectors $\mathbf{w}$ and $\mathbf{x}$, written as the scalar product of the row vector $\mathbf{w}^{\mathrm{T}}$ (transposed column vector) and the column vector $\mathbf{x}$.

2. **Loss function** (and the associated **error function**)

   The regression algorithm uses the squared difference as the loss function, aka **quadratic loss** (hrv. *kvadratni kubitak*):

$$L(y^{(i)}, h(\mathbf{x}^{(i)})) = \big(y^{(i)} - h(\mathbf{x}^{(i)})\big)^2$$

   The error function is defined as the sum of the losses over all examples from the labeled set $\mathcal{D}$, divided by two (and this is proportional to the empirical expected value of the loss):

$$E(h|\mathcal{D}) = \frac{1}{2}\sum_{i=1}^{N}\big(y^{(i)} - h(\mathbf{x}^{(i)})\big)^2$$

3. **Optimization process**

   Same as last week, we will first define the optimization process in very abstract terms:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}}\, E(\mathbf{w}|\mathcal{D})$$

   In case of regression, where the error function is defined by the quadratic loss, this procedure is called the **least-squares method** (hrv. *metoda najmanjih kvadrata*) or **ordinary least squares, OLS** (hrv. *obični najmanji kvadrati*). The name reflects the fact that we are searching for parameters $\mathbf{w}$ for which the sum of the squared differences (the squared residuals) are the smallest.

   The good news is that with a linear regression model the solution to this optimization problem always exists in **closed form**, and this is the case regardless of the number of features n. This means that we will be able to write the solution as a mathematical expression calculable a finite number of steps. (Admittedly, as we are about to see, this calculation will not be entirely trivial, as it involves calculating the inverse of a matrix, which is non-trivial for large matrices).

4 Least-squares method

Let us now consider in more detail the least-squares method, that is, how to obtain the the parameters $\mathbf{w}$ that are optimal in terms of the sum of squared residuals, an let's do this for multiple linear regression, where $n > 1$. We have a set of examples (in regression, these are input variables or "independent variables"):

$$\mathbf{X} = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} \dots & x_n^{(2)} \\ \vdots & & & \\ 1 & x_1^{(N)} & x_2^{(N)} \dots & x_n^{(N)} \end{pmatrix}_{N\times(n+1)}$$

Recall that the matrix $\mathbf{X}$ is called the **design matrix**. We also have a vector of output values (i.e., the labels or "dependent variables"):

$$\mathbf{y} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{pmatrix}_{N\times 1}$$

4.1 A dead end: Exact solution

Let's try first with something other than the least-squares method, and that will turn out to be a bad idea. Namely, we can think of our optimization problem as a system of equations. Ideally, we would like to find a solution for which

$$(\mathbf{x}^{(i)}, y^{(i)}) \in \mathcal{D}.\ h(\mathbf{x}^{(i)}) = y^{(i)}$$

that is

$$(\mathbf{x}^{(i)}, y^{(i)}) \in \mathcal{D}.\ \mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)} = y^{(i)}$$

This would be an **exact solution** of our problem: a solution in which the regression model perfectly predicts the output value for each input value. A problem defined in this way is actually a system of N equations with $(n+1)$ unknowns. The system can be elegantly written as a **matrix equation**:

$$\begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} \dots & x_n^{(2)} \\ \vdots & & & \\ 1 & x_1^{(N)} & x_2^{(N)} \dots & x_n^{(N)} \end{pmatrix} \cdot \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{pmatrix} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(N)} \end{pmatrix}$$

that is

$$\mathbf{X}\mathbf{w} = \mathbf{y}$$

(convince yourself that this is the same as the system of equations from above). The exact solution is obtained by multiplying the equation on the left by $\mathbf{X}^{-1}$:

$$\mathbf{w} = \mathbf{X}^{-1}\mathbf{y}$$

This is nice, but it won't always work. Namely, the problem will often be **ill-defined** (engl. *loše definiran*), which means that the above system of linear equations either has no solution (**it is inconsistent**) or it **has no unique solution** (there are infinitely many solutions). The first and most banal problem is that for the design matrix $\mathbf{X}$ to have an inverse, it must be **square**.

However, the number of examples can generally be both greater than the number of parameters, $N > n+1$, or it may be smaller, $N < n+1$. In the first case we say that a system is **overdetermined** (engl. *preodređen*), while in the latter we say it is **underdetermined** (engl. *pododređen*). In both cases the matrix $\mathbf{X}$ will not be square and will not have an inverse. Truth be told, the requirement that the matrix $\mathbf{X}$ be square plays a role only if we want to solve the system of equations using the matrix inversion. We are free to use another method, e.g., Gaussian elimination, which is also applicable to rectangular matrices. But this, of course, still doesn't mean that the system of equations will have a solution nor that the solution will be unique. More concretely, if the **rank** of the matrix $\mathbf{X}$ is smaller than the rank of the augmented matrix $\mathbf{X}|\mathbf{y}$, then the system of equations is inconsistent. Furthermore, if the rank of the matrix $\mathbf{X}$ is not equal to the number of columns (i.e., the number of parameters, which is equal to $n+1$), then the system has infinitely many solutions. In other words, for a system to have a single and unique solution, the ranks of both the matrices $\mathbf{X}$ and $\mathbf{X}|\mathbf{y}$ must be equal to $n+1$.

▸ **EXAMPLE**

The following simple examples demonstrate that the exact solution of a system of linear equations will not bring us very far.

Let the set of training examples for simple regression be $\mathcal{D} = \{(x^{(i)}, y^{(i)})\} = \{(1,1),(1,2)\}$. Although the design matrix $\mathbf{X}$ is square, the system is inconsistent (matrix $\mathbf{X}$ has rank 1, while the augmented matrix $\mathbf{X}|\mathbf{y}$ has rank 2). This is because we have two different labels for instance $x = 1$. Conversely, for the training set set $\mathcal{D} = \{(1,1),(1,1)\}$ the matrix $\mathbf{X}$ is still square, but the matrix $\mathbf{X}|\mathbf{y}$ has rank 1, whereas $n+1 = 2$, so the system has an infinite number of solutions. Intuitively, this is because the training set has only a single instance, so an infinite number of lines can pass through a single point.

Let us now consider the set of examples $\mathcal{D} = \{(1,1),(2,2),(3,3)\}$. These points lie on the line and we expect to find the exact solution for the regression line. However, we cannot find this solution by the above equation (using matrix inverse) because the design matrix $\mathbf{X}$ is not square (instead, the dimensions are 3×2), i.e, the system is overdetermined. However, the system is consistent (matrix $\mathbf{X}$ is of rank 2, which is equal to the rank of matrix $\mathbf{X}|\mathbf{y}$) and has a unique solution (the rank is equal to $n+1 = 2$), so the solution ($w_0 = 0, w_1 = 1$) can be obtained, for example, using the method of Gaussian elimination. But in reality the data will not be so ideal. Recall the **noise** we talked about last time. It suffices that, due to the noise, one of these three points shifts a little from the straight line, and the whole thing will break down. For example, if the set of training examples is $\mathcal{D} = \{(1,1),(2,2.1),(3,3)\}$ (due to the noise the label of the second example is 2.1 instead of 2), we already have an inconsistent system of equations (rank of $\mathbf{X}$ is 2 and rank of $\mathbf{X}|\mathbf{y}$ is 3). Obviously, this is because we cannot draw a line through these three points, as they do not lie on the same line.

Similarly, consider a set of examples $\mathcal{D} = \{((1,1),1),((2,2),2)\}$ for a two-input ($n = 2$) multiple regression. The corresponding augmented matrix $\mathbf{X}|\mathbf{y}$ looks like this:

$$\left(\begin{array}{ccc|c} 1 & 1 & 1 & 1 \\ 1 & 2 & 2 & 2 \end{array}\right)$$

The ranks of $\mathbf{X}$ and $\mathbf{X}|\mathbf{y}$ are both equal to 2, which is less than the number of parameters ($n+1 = 3$), hence the system of equations has an infinite number of solutions. This is because we have only two examples: each example is one point in a 3D space defined by coordinates (x_1, x_2, y), and one can draw an infinite number of planes through two points in 3D space.

The above examples show us that, from a machine learning perspective, the exact solution is not robust enough. Even if the data is generated by some linear function, the training set will contain some noise, and we therefore cannot expect all the points to lie perfectly on a line. Furthermore, even if we have fewer examples than parameters, we would still like to have some solution. All in all, an exact solution assumes that the world is perfect, but it is not. In an imperfect world, an approximate solution is perfectly good.

4.2 The way forward: Least squares solution

Because of the limitations we have just established, instead of trying to find an exact solution of the system $\mathbf{X}\mathbf{w} = \mathbf{y}$, we will go for an **approximate** solution. What do we mean by "approximate"? We could define this in a number of ways, but when it comes to the least squares method "approximate" is defined in terms of squared differences. We actually did that already, when we looked at the very first example with simple regression. There, we defined the loss function via the quadratic loss function, which actually measures the squared differences between the solution and the labels in the dataset. Now we would like to generalize this approach to multiple regression, i.e., the case with multiple features, $n > 1$.

Enter **matrix calculus**. Namely, now that we have more features, it is much simpler to devise the optimization procedure use matrix calculus, which will amount to doing certain operations on matrices.

Recall, our regression error function is as follows:

$$E(\mathbf{w}|\mathcal{D}) = \frac{1}{2}\sum_{i=1}^{N}\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)} - y^{(i)}\right)^2$$

The matrix form of this function is:

$$E(\mathbf{w}|\mathcal{D}) = \frac{1}{2}(\mathbf{X}\mathbf{w} - \mathbf{y})^{\mathrm{T}}(\mathbf{X}\mathbf{w} - \mathbf{y})$$

Take some time here to convince yourself that this is indeed the case. (You'll know that you understand when you know how to explain where the square and the sum disappeared.) Now let's expand the right side of the expression. We apply the transposition to individual summands (because $(\mathbf{A}+\mathbf{B})^{\mathrm{T}} = \mathbf{A}^{\mathrm{T}}+\mathbf{B}^{\mathrm{T}}$), then we apply distributivity and then the following two equations from the matrix calculus:

$$(\mathbf{A}^{\mathrm{T}})^{\mathrm{T}} = \mathbf{A}$$
$$(\mathbf{A}\mathbf{B})^{\mathrm{T}} = \mathbf{B}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}$$

Doing this gives us:

$$E(\mathbf{w}|\mathcal{D}) = \frac{1}{2}(\mathbf{w}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X}\mathbf{w} - \mathbf{w}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{y} - \mathbf{y}^{\mathrm{T}}\mathbf{X}\mathbf{w} + \mathbf{y}^{\mathrm{T}}\mathbf{y})$$

This is where we need to recall that $E(\mathbf{w}|\mathcal{D})$ is actually a scalar, which means that all the expressions we add up here must actually be of the dimensions 1×1. Indeed, for instance for $\mathbf{w}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{y}$:

$$\underbrace{1 \times (n+1)}_{\mathbf{w}^{\mathrm{T}}} \cdot \underbrace{(n+1) \times N}_{\mathbf{X}^{\mathrm{T}}} \cdot \underbrace{N \times 1}_{\mathbf{y}} = 1 \times 1$$

If these are scalars, then we can transpose them without causing any change, because the transposition of scalars gives the same scalar. And that then means that the middle two summands in the above expression are equal, because

$$\mathbf{w}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{y} = (\mathbf{w}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{y})^{\mathrm{T}} = \left((\mathbf{w}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}})\mathbf{y}\right)^{\mathrm{T}} = \mathbf{y}^{\mathrm{T}}(\mathbf{w}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}})^{\mathrm{T}} = \mathbf{y}^{\mathrm{T}}(\mathbf{X}\mathbf{w})$$

where we used the above two equations from the matrix calculus. So in the end we get:

$$E(\mathbf{w}|\mathcal{D}) = \frac{1}{2}(\mathbf{w}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X}\mathbf{w} - 2\mathbf{y}^{\mathrm{T}}\mathbf{X}\mathbf{w} + \mathbf{y}^{\mathrm{T}}\mathbf{y})$$

Now we want to find a minimizer of this function. This will require the use of rules for **matrix differentiation**. There are a number of rules, but luckily only the following two will suffice:

$$\frac{\mathrm{d}}{\mathrm{d}\mathbf{x}}\mathbf{A}\mathbf{x} = \mathbf{A}$$
$$\frac{\mathrm{d}}{\mathrm{d}\mathbf{x}}\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} = \mathbf{x}^{\mathrm{T}}(\mathbf{A} + \mathbf{A}^{\mathrm{T}})$$

Before we continue, let's stop for a moment to think about what we actually want to achieve. We want to find the derivative of the error function $E(\mathbf{w}|\mathcal{D})$ by the vector $\mathbf{w}$, because we are looking for the minimum by the parameters $\mathbf{w}$. The E function is a function of the vector $\mathbf{w}$, so it is a function of $(n+1)$ variables. What will be the derivative of the function E by the vector $\mathbf{w}$, i.e. what is $\nabla_{\mathbf{w}}E$? This is going to be the **gradient** of the E function. The gradient is again a function, namely a vector function, which for a point $\mathbf{w}$ gives the direction (vector) of the fastest growth of the function E. So $\nabla_{\mathbf{w}}E$ is a $(n+1)$ dimensional vector (i.e., the gradient). A stationary point of a function is one in which the gradient is equal to zero (more precisely, the zero vector), and for the function E the stationary point will be the minimum point (remember, the function E is convex, so its stationary point is certainly its minimum). Ultimately, then, we are interested in the point where the gradient of the function E is equal to zero. Having refreshed our memory on gradients, let's move from words to deeds...

Let's find the derivative of the error function and equate it with the zero vector:

$$\nabla_{\mathbf{w}}E = \frac{1}{2}\Big(\mathbf{w}^{\mathrm{T}}\big(\mathbf{X}^{\mathrm{T}}\mathbf{X} + (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{\mathrm{T}}\big) - 2\mathbf{y}^{\mathrm{T}}\mathbf{X}\Big) = \mathbf{w}^{\mathrm{T}}(\mathbf{X}^{\mathrm{T}}\mathbf{X}) - \mathbf{y}^{\mathrm{T}}\mathbf{X} = \mathbf{0}$$

from which it follows:

$$\mathbf{w}^{\mathrm{T}}(\mathbf{X}^{\mathrm{T}}\mathbf{X}) = \mathbf{y}^{\mathrm{T}}\mathbf{X}$$

that is, after transposition (applying the above rules for transposition of the matrix product):

$$\mathbf{X}^{\mathrm{T}}\mathbf{X}\mathbf{w} = \mathbf{X}^{\mathrm{T}}\mathbf{y}$$

What we got is a system of so-called **normal equations**. The solution of the system is obtained by multiplying by $(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}$ on the left:

$$\mathbf{w} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y} = \mathbf{X}^{+}\mathbf{y}$$

which gives us the solution for the parameter vector $\mathbf{w}$ that minimizes the squared differences.

A few comments are in order. First, we see that what we've got is again a closed-form solution, which is rather nice. Second, the matrix we denoted by $\mathbf{X}^{+}$, defined as $\mathbf{X}^{+} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}$, is the so-called **pseudoinverse** (more precisely: the Moore-Penrose inverse) of matrix $\mathbf{X}$. What is a pseudoinverse? Pseudoinverse is a generalization of the concept of matrix inverse. Unlike ordinary inverse, which exists only for full-rank square matrices, pseudoinverse **exists for just about every matrix**. As a special case, if the matrix $\mathbf{X}$ is square and of full rank, then the pseudoinverse is equal to the ordinary inverse, $\mathbf{X}^{+} = \mathbf{X}^{-1}$.

What is important to note here is that the pseudoinverse is in and of itself the solution to least squares problem. Namely, the initially defined the problem in terms of squared differences, i.e., the quadratic error function was built into the problem from the onset. This means that calculating the pseudoinverse will give us the parameters $\mathbf{w}$ for which the hyperplane defined by $h(\mathbf{x};\mathbf{w})$ is optimal in terms of least square differences from labeled data.

More precisely, the solution $\mathbf{w}$ given by this equation is such that the distance in terms of squared differences between vectors $\mathbf{X}\mathbf{w}$ and $\mathbf{y}$ is minimal, that is, it a solution minimizing the L_2-norm $\|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2$. This is also the reason why we call the above equations "normal".

If the system of equations is underdetermined and has multiple solutions, then the pseudoinverse gives the solution with the smallest norm $\|\mathbf{w}\|_2$. If the system is overdetermined, the pseudoinverse gives a solution that minimizes $\|\mathbf{X}\mathbf{w} - \mathbf{y}\|$, but if such a solution is not unique, it again gives the one with the smallest norm $\|\mathbf{w}\|_2$.

4.3 Computational aspects of the least squares method

The good thing here is that our optimization procedure gives a closed-form solution. However, obtaining this solution – which falls under the purview of **numerical mathematics** – can still be quite computationally demanding.

Notice, namely, that we have to calculate the **inverse** of the product of design matrix with itself: $\mathbf{X}^\mathrm{T}\mathbf{X}$. We call this matrix the **Gram matrix**. The Gram matrix can be quite large, so the computing its inverse can be quite expensive – typically $\mathcal{O}(n^3)$ (e.g., using the LU decomposition method). The key question, therefore, is the following: what are the dimensions of the matrix we are inverting, $(\mathbf{X}^\mathrm{T}\mathbf{X})^{-1}$? The answer is: the dimensions are $(n+1) \times (n+1)$). Therefore, the complexity of computing the matrix inverse depends only on the number of features n and not on the number of examples N. It's good to keep that mind.

Another thing to note is that although the Gram matrix is obviously always square (its dimensions are $(n+1) \times (n+1)$), it need not be of full rank. Namely, it can be shown that the rank of the Gram matrix is equal to the rank of the design matrix $\mathbf{X}$. If that rank is equal to $n+1$, then the Gram matrix is of full rank and the pseudoinverse can be calculated as described above, that is: $\mathbf{X}^+ = (\mathbf{X}^\mathrm{T}\mathbf{X})^{-1}\mathbf{X}^\mathrm{T}$. However, if the rank of the design matrix is less than $n+1$ (which will be the case if the design matrix is "shallow", that is, when $N < n+1$, i.e., when we have fewer examples than features), then the pseudoinverse cannot be computed in this way. However, in this case the pseudoinverse can still be computed using other methods, typically **singular value decomposition (SVD)**. But we will not go into details here. It suffices to know that we can always compute the pseudoinverse, one way or another. We'll have more to say on computational aspects of pseudoinverses next time, when we talk about overfitting of the regression model and regularization as a way to prevent it.

5 Probabilistic interpretation of regression

We defined the loss as **squared differences**, or squared residuals, and the empirical error as the sum of the squared residuals. However, our justification for using the squared differences was somewhat unconvincing. In machine learning, we should not remain content with such superficial justifications. Is there a deeper reason why we would want to use the quadratic loss? There is one, indeed. But to see this, we need to consider machine learning from a probabilistic perspective. While we will adopt this perspective in the second half of the course, it is now already useful to make a short excursion into the world of probabilistic machine learning. We will see that all these perspectives are intertwined, and that it is useful, and event necessary, to gain an understanding of an algorithm from multiple perspectives.

Let us look, then, at a probabilistic interpretation of linear regression. The linear regression model is as follows:

$$h(\mathbf{x}; \mathbf{w}) = \mathbf{w}^\mathrm{T}\mathbf{x}$$

Suppose now that our labeled examples are in fact generated by some function $f : \mathbb{R}^n \to \mathbb{R}$. This function is often referred to as the **"function underlying the data"** (hrv. *funkcija iza podataka*). In reality, of course, this function **is unknown** to us.

As we already know all too well, labeled data inevitably contains some **noise**, which is being superimposed on the data's underlying function. So the labels we observe are in fact the value of the function plus noise:

$$y^{(i)} = f(\mathbf{x}^{(i)}) + \varepsilon_i$$

And this is where we are entering probabilistic waters. We will model the noise as a random variable, distributed according to **normal (Gaussian) distribution**, centered at zero and with variance σ^2. We write this as:

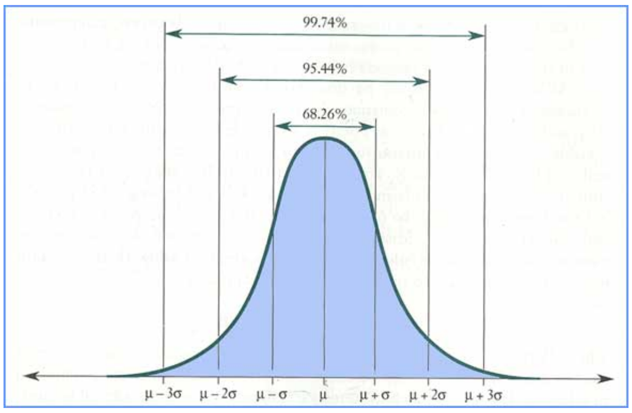
$$\varepsilon \ \sim \ \mathcal{N}(0, \sigma^2)$$

As a matter of fact, noise, like many other phenomena in nature, can be well described by the normal distribution, so this assumption is perfectly reasonable. Furthermore, we assume that the noise is the same for each individual example, $\varepsilon = \varepsilon_i$.

Recall, the Gaussian distribution (aka normal distribution) has the following **probability density function (PDF)** (hrv. *funkcija gustoće vjerojatnosti*):

$$p(Y = y|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp\Big(-\frac{(y-\mu)^2}{2\sigma^2}\Big).$$

Let's also recall what this function looks like:



We can now write that for a given input value $\mathbf{x}$ the output value y will be distributed around the value $f(\mathbf{x})$ (with deviations due to noise):

$$p(y|\mathbf{x}) = \mathcal{N}\big(f(\mathbf{x}), \sigma^2\big)$$

$p(y|\mathbf{x})$ is the probability that example $\mathbf{x}$ is labeled y, if we know that the data was generated by the $f(\mathbf{x})$ function. (Actually, $\mathbf{x}$ is a continuous variable, so p is the probability density of the y label rather than the probability, but this is less importance here and the idea still stands.)

With this, we modeled the label probability for an individual example. The next step is to model the probability of all labels in the entire set of labeled examples $\mathcal{D}$. Recall, the set of labeled examples $\mathcal{D}$ can be decomposed into a design matrix $\mathbf{X}$, which contains only the examples, and a label vector $\mathbf{y}$, which contains labels for all examples. Now, the probability that the set of examples $\mathbf{X}$ is labeled with $\mathbf{y}$ is:

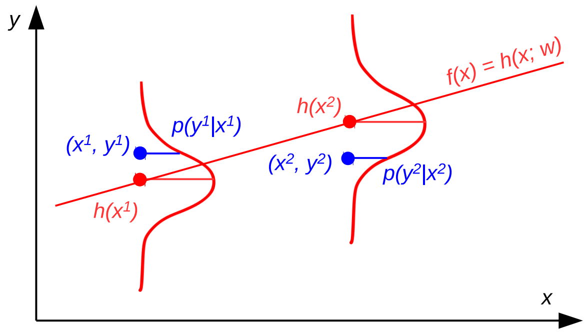
$$p(\mathbf{y}|\mathbf{X}) = \prod_{i=1}^{N} p(y^{(i)}|\mathbf{x}^{(i)}) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\Big(-\frac{(y^{(i)} - f(\mathbf{x}^{(i)}))^2}{2\sigma^2}\Big).$$

How did we get to this? We assumed that the example labels are **independent** of each other and come from the **same distribution**. If so, then the probability of a set of labels is simply the product of the probabilities of the individual labels. This is a reasonable assumption to make here. The assumption is also very often used in machine learning, and goes under the name of **independently and identically distributed variables (IID)** (engl. *nezavisne i identično distribuirane varijable*).

The above probability therefore tells us what is the probability of examples $\mathbf{X}$ being labeled with $\mathbf{y}$ if examples are generated by the function $f(\mathbf{x})$. However, let us recall that the function $f(\mathbf{x})$ is unknown to us: we don't know which function generated the data. In fact, our task is precisely to find that function, and we model this function with our hypothesis $h(\mathbf{x}|\mathbf{w})$. So, we are searching for the hypothesis $h(\mathbf{x}; \mathbf{w})$ that generated the examples, i.e., the hypothesis satisfying $h(\mathbf{x}; \mathbf{w}) = f(\mathbf{x})$. What this simply means is that instead of $f(\mathbf{x})$ we can write $h(\mathbf{x}; \mathbf{w})$. So, we have:

$$p(\mathbf{y}|\mathbf{X}) = \prod_{i=1}^{N} p(y^{(i)}|\mathbf{x}^{(i)}) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\Big(-\frac{(y^{(i)} - h(\mathbf{x}^{(i)}; \mathbf{w}))^2}{2\sigma^2}\Big).$$

Thus, this is the probability of examples $\mathbf{X}$ having labels $\mathbf{y}$, assuming they are generated by the $h(\mathbf{x};\mathbf{w})$ function. We also refer to this as "the probability of the data under the model" (meaning, if the hypothesis h is true, the probability of examples being labeled as this is such-and-such). Graphically, this looks as follows:



So at each point $\mathbf{x}^{(i)}$ we have one normal distribution centered at $f(\mathbf{x}^{(i)})$. The label $y^{(i)}$ may deviate from the label predicted by the hypothesis, $h(\mathbf{x}^{(i)};\mathbf{w})$, and this deviation is modeled by the normal distribution. The label $y^{(i)}$ will most probably land in the middle of the normal distribution, i.e., $y^{(i)} = h(\mathbf{x}^{(i)};\mathbf{w})$, because this is where the probability density $p(y^{(i)}|\mathbf{x}^{(i)})$ is the highest. The farther away the label $\mathbf{x}^{(i)}$ from that center, the less likely it is. The total probability (actually, the total probability density) of the entire set of labels $\mathbf{y}$ is the product of the probability density $p(y^{(i)}|\mathbf{x}^{(i)})$ for all points $\mathbf{x}^{(i)}$. The product will be the higher (i.e., the probability of the labeled set the larger) the closer the $y^{(i)}$ labels to the center of their corresponding normal distributions. Analogously, the product will be the higher (i.e. the probability of the labeled set the larger) the closer function $h(\mathbf{x};\mathbf{w})$ passes through each label $y^{(i)}$.

This last observation is crucial: the probability of a set of labeled examples depends on function $h(\mathbf{x};\mathbf{w})$. This probability will be maximal if the hypothesis h passes as close as possible to labels $\mathbf{y}$. Our goal then is to find a hypothesis h, (or, alternatively, the weights $\mathbf{w}$) so to maximize the probability of the set of examples. In other words, we want to find a hypothesis such that it makes the data that we have most likely. In doing so, we actually assume that the data $\mathcal{D}$ we happen to have at our disposal the is the data that is the most likely to occur: if we have gotten this data, then we will assume that these are the most probable data, for otherwise we would more probably have ended up with different data.

We are, therefore, after a hypothesis that **maximizes the probability of labels**. It will be easier if we switch to logarithmic scale (and we'll soon see why). Also, instead of hypothesis h we can write weights $\mathbf{w}$ because it is weights what we are actually looking for. The probability of $\mathbf{y}$ for weights $\mathbf{w}$ is as follows:

$$\ln p(\mathbf{y}|\mathbf{w}) = \ln \prod_{i=1}^{N} p(y^{(i)}|x^{(i)}) = \ln \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{ -\frac{\left(y^{(i)} - h(\mathbf{x}^{(i)};\mathbf{w})\right)^2}{2\sigma^2} \right\}$$

Now we can rearrange the expression a bit (apply the logarithm to the product), which gives:

$$\underbrace{-N\ln(\sqrt{2\pi}\sigma)}_{=\text{const.}} - \frac{1}{2\sigma^2}\sum_{i=1}^{N}\left(y^{(i)} - h(\mathbf{x}^{(i)};\mathbf{w})\right)^2$$

As we are interested in maximization by $\mathbf{w}$, the term σ^2 may be dropped because it is constant. We are left with:

$$-\frac{1}{2}\sum_{i=1}^{N}\left(y^{(i)} - h(\mathbf{x}^{(i)};\mathbf{w})\right)^2$$

Remember, this is the probability we want to **maximize**. Consequently, the expression after the minus sign is what we want to **minimize**. And that expression is exactly the **quadratic error function**! In other words, assuming normally distributed noise, the hypothesis that will maximize the probability of data being labeled as it it happens to be labeled is the one that minimizes the sum of squared differences between predicted and true labels. And that, finally, is the probabilistic justification for using the quadratic loss function!

The procedure we have followed here can be summarized as follows: we have expressed the probability of the labels by modeling each label as a normally distributed variable whose mean value equals the prediction of the hypothesis, after which we expressed the probability of the set of labels assuming IID. We then searched for hypothesis' parameters that maximize that probability. That brought us to the expression that we want to maximize, which turned out to be equal to the negative empirical error, which we want to minimize. The principle we just described is not random, but a general principle of how one can define error functions in machine learning. We'll talk more about this in some of the future lectures.

Summary

- Regression is used to predict **numeric values** (dependent variables) based on input examples (independent variables)
- In **linear regression**, the output value is a linear combination of input values
- The loss function is the **quadratic loss** (the squared difference between the predicted and correct label)
- Optimization uses the **least squares method**, computed using the **pseudoinverse** of the design matrix
- Assuming normally distributed noise, the least squares solution is equivalent to maximizing the probability of the labels, which gives a **probabilistic justification** for the quadratic loss