50.039 – Theory and Practice of Deep Learning

Alexander Binder

Tuesday, Week 01: Discriminative ML - quick intro

Supervised Discriminative Machine Learning

Learning goals for today

- being able to reproduce the basic philosophy of discriminative machine learning
- being able to execute the steps needed to set up a discriminative machine learning task
- Discriminative Learning problems consist usually of a least four basic components
  1. Input space $\mathcal{X}$ and output space $\mathcal{Y}$
  2. the prediction mapping connecting these $f : \mathcal{X} \to \mathcal{Y}$
  3. a loss function
  4. a method to select a prediction mapping f / or its parameters from a dataset
- the data to be expected to be seen is modeled by a probability distribution
- training and testing data are finites samples from it
- Learning $\approx$ finding a good predictive mapping f of an input space to an output space.
- *good* mapping means: one that minimizes an expectation of a loss function. The expectation of the loss is based on a probability of the data expected to be seen at application time.
- that probability distribution is unknown, but the expected loss can be approximated using training and testing data sets
- occams razor for selecting models

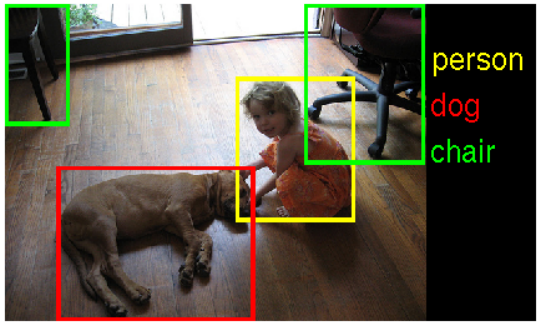
Lets look at some examples to understand about Input space $\mathcal{X}$ and output space $\mathcal{Y}$

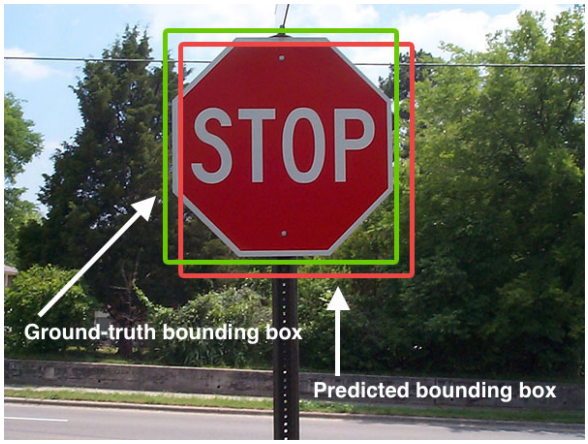
- Classification of Images

```
top 10 classes and their scores: 670 | n03791053 motor scoo 0.636128
top 10 classes and their scores: 880 | n04509417 unicycle,  0.261106
top 10 classes and their scores: 665 | n03785016 moped 0.0432479
top 10 classes and their scores: 573 | n03444034 go-kart 0.0156194
top 10 classes and their scores: 444 | n02835271 bicycle-bu 0.00917528
top 10 classes and their scores: 920 | n06874185 traffic li 0.00490618
top 10 classes and their scores: 518 | n03127747 crash helm 0.00412505
top 10 classes and their scores: 822 | n04311174 steel drum 0.0024649
top 10 classes and their scores: 733 | n03976657 pole 0.00223075
top 10 classes and their scores: 557 | n03355925 flagpole,  0.00199626
```

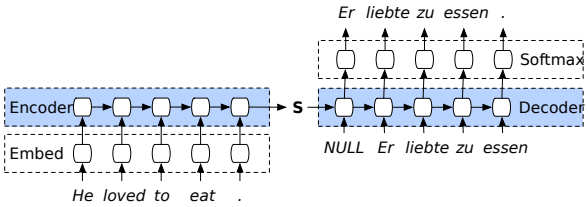
  - input: image (how to represent it?)
  - output: concept label
- Classification of sounds
  - input: sound recording (how to represent it?)
  - output: concept label
- Regression of blood pressure based on amount of food, percentage of fibers, sugars,fats, proteins
  - input: ?
  - output: ?

- Detection of a objects in images
  - input: image



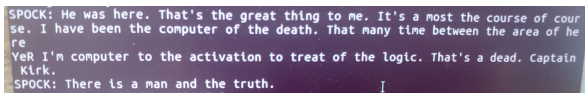


  - output: ?
- Machine Translation
  - input: (how to represent it?)



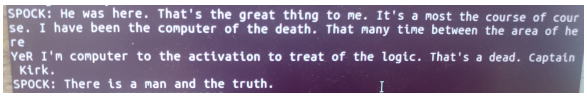
(from smerity.com)

  - output: ?
- Text Summarization
  - input: sequence of words $(w_1, \ldots, w_K)$ same as above



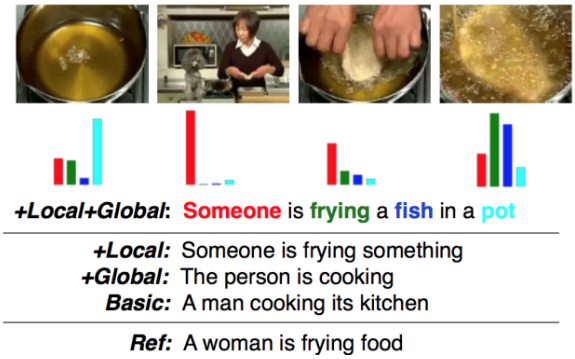
  - output: ?

- Key word assignment
  - input: sequence of words $(w_1, \ldots, w_K)$ same as above



  - output: ?

- Video captioning
  - input: ?



(from nvidia)

  - output: ?

Always the same structure:

- x – data sample, from an input space $\mathcal{X}$ get mapped onto an output y in some space $\mathcal{Y}$
- Prediction by a mapping $f : \mathcal{X} \to \mathcal{Y}$, $f(x) = y \in \mathcal{Y}$.

Usually the first task in designing any learning system is to think how to describe the input and output spaces $\mathcal{X}, \mathcal{Y}$

Assumptions for discriminative learning setups

Assumption 1: Supervised Learning
Have a dataset for which we know for every input sample what prediction is desired:

- x – an input sample
- y – its ground truth prediction .
- a data sample/point comes as a pair (x, y)

Assumption 2: Loss function
able to define/measure a loss $l(f(x), y)$ for how wrong our prediction $f(x)$ is relative to its ground truth y.
(is groundtruth always unique??)

Example for a loss function – classification with C classes.

Output space $\mathcal{Y} = \{0, \ldots, C-1\}$.
Classification is wrong on (x, y) if $f(x) \neq y$

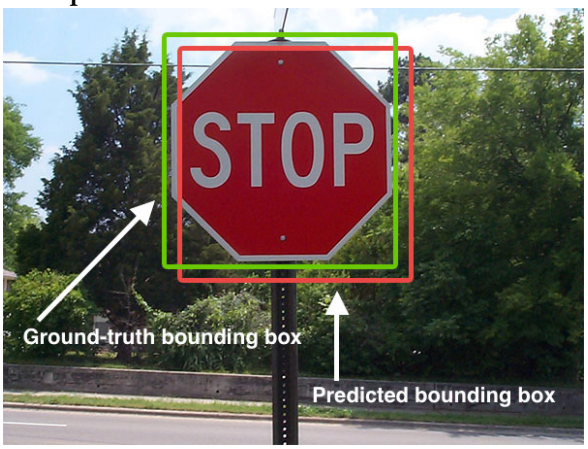
$$L(f(x), y) = 1[f(x) \neq y] = \begin{cases} 0 & \text{if } f(x) == y \\ 1 & \text{if } f(x) \neq y \end{cases}$$

Example for a loss function – Regression

$$L(f(x), y) = (f(x) - y)^2$$

is 0 if prediction exactly matches the regression groundtruth.

Example for a loss function – Detection:



Intersection over union

$$IoU(gt, pred) = \frac{Area(gt \cap pred)}{Area(gt \cup pred)}$$

Area of intersection of predicted vs true bounding box, divided by area of union of those.

What goal to achieve for the prediction? Part 1

We have

- know how to describe input $\mathcal{X}$ and output $\mathcal{Y}$ spaces
- have a class of functions $f : \mathcal{X} \to \mathcal{Y}, f(x) = y$.
- loss function $L(f(x), y)$ for measuring disagreement $f(x) \leftrightarrow y$ (prediction vs ground truth)

How to select/learn a good mapping f? Guiding it by what?

When we deploy the mapping f at test time after training/research, it should *perform well* when deployed.

> Perform well $\longleftrightarrow$ low loss $L(f(x), y)$.

If we would know what samples (x, y) we will see at test time, then we can collect them into a test dataset $T_n = \{(x_i, y_i), i = 1, \ldots, n\}$, and then it is obvious to choose f such that

$$\sum_{(x_i,y_i)\in T_n}^{n} L(f(x_i), y_i) = \sum_{i=1}^{n} L(f(x_i), y_i) \to \text{minimal}$$

$$f^* = \text{argmin}_f \sum_{i=1}^{n} L(f(x_i), y_i)$$

... it performs well on average on the test samples from T_n.

Observation: Uncertainty what input samples x we will see at deployment time.

Idea: model the uncertainty by assuming that the input samples will be drawn from a probability distribution. Include output labels in the same distribution.

$$\text{Test/Deployment time: } (x, y) \sim P_{test}$$

The goal of discriminative learning

Choose the model f^* such that it has low loss on average under the test time/deployment time probability distribution. This average under the test probability distribution is the expectation $E_{(x,y)\sim P_{test}}[\cdot]$:

$$E_{(x,y)\sim P_{test}}[L(f(x),y)] \rightarrow \text{minimal}$$
$$f^* = \text{argmin}_f E_{(x,y)\sim P_{test}}[L(f(x),y)]$$

recap Expectation:
expectation is an average of function values over all your data

$E_{X\sim P}[t(X)] = \sum_{x\in Values} t(x)p(x)$ if X takes countable (infinite or finite) number of values

$E_{X\sim P}[t(X)] = \int t(x)f(x)dx$ if P follows a density defined by $f(x)$

A blocker

usually we do not have an explicit definition of P_{test}!

Solution: Law of large numbers. If we have drawn a dataset of samples $T_n = \{(x_i, y_i), i = 1, \ldots, n\}$ **independently and identically distributed** from P_{test}, then

$$\frac{1}{n}\sum_{(x_i,y_i)\in T_n}^{n} L(f(x_i),y_i) \overset{n\rightarrow\infty}{\longrightarrow} E_{(x,y)\sim P_{test}}[L(f(x),y)]$$

$$\text{and thus } E_{(x,y)\sim P_{test}}[L(f(x),y)] \approx \frac{1}{n}\sum_{(x_i,y_i)\in T_n}^{n} L(f(x_i),y_i)$$

Approximation becomes better as n gets larger

definition

The empirical risk on a labeled dataset S_n is defined as

$$R(f, L, S_n) = \frac{1}{n}\sum_{(x_i,y_i)\in S_n} L(f(x_i),y_i)$$

basic idea for machine learning

- draw a training dataset D_n from the test time distribution P_{test}
- select a mapping f^* such that

$$f^* = \text{argmin}_f R(f, L, D_n)$$

This is called empirical risk minimization

There is something missing: after we have selected a mapping f, we need to report its performance.

How to measure the performance of the selected mapping?

Use a evaluation/testing dataset T_n **disjoint (!!!) from the training data** D_n . Measure empirical risk $R(f, L, T_n)$.

It is important: one cannot reliably evaluate the performance on the data D_n used for training. T_n must be disjoint from D_n for reliable performance estimates.

performance measurement for machine learning

- draw a test dataset T_n from the test time distribution P_{test} **disjoint** to the training dataset D_n
- report for the selected mapping f^*

$$R(f^*, L, T_n)$$

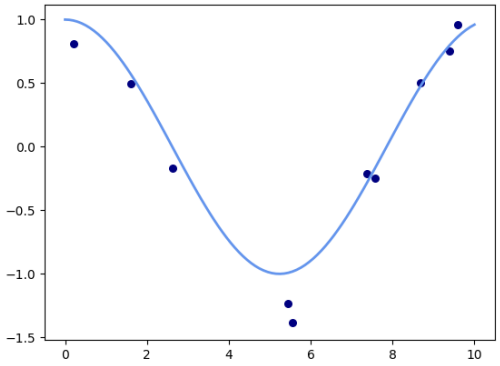
It should be noted here: sometimes one uses a different loss L for training than what is used to report performance at test time. This is because some criteria used at test time are not differentiable, and therefore cannot be used with optimization toolboxes.

Why we have to evaluate the performance on a testset which is disjoint from the training set?

The reason is: We choose a mapping f based on minimizing loss on the training set. It is very easy to select f such that the loss on the training set becomes 0 or very low, but the loss on data not used for training is very high.

Lets give an example:

We have a data of 10 points drawn from a data source:



We know for this toy problem how the data was generated:

$$y = cos(0.6x) + 0.2\epsilon, \epsilon \sim N(0,1)$$

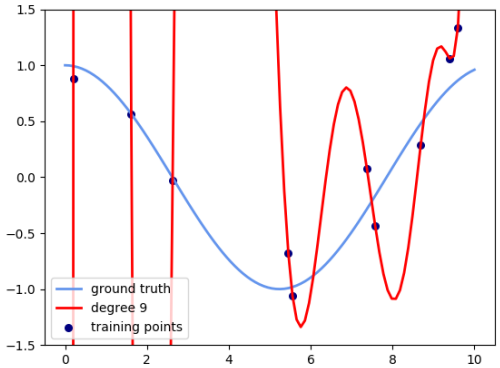
That is: there is the true relationship $\bar{y} = cos(0.6x)$ with added zero-mean noise. The best function which we could learn is the true relationship.

Goal fit a function $f(x)$ which predicts y from x well. Regression problem. Chosen loss between prediction $f(x)$ and groundtruth y is

$$L(f(x), y) = (f(x) - y)^2$$

We choose regression with polynomial basis functions (next lecture).

For polynomial of order 9:

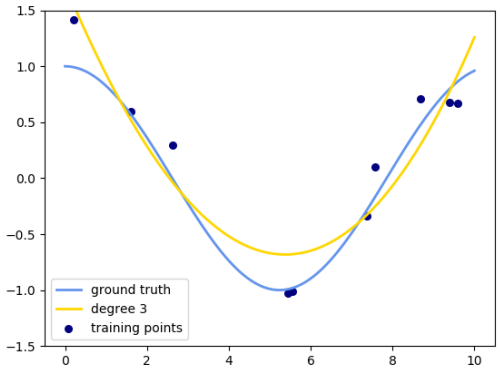


Prefectly fits each training data point. Loss on training data is zero.

But: if we draw new datapoints from $y = cos(0.6x) + 0.2\epsilon, \epsilon \sim N(0,1)$, then these points will be close to the underlying structure

$\bar{y} = cos(0.6x)$, and thus the prediction by the red curve will result in a very high error on most sets of new data points (with a high probability of draws of testsets T_n)

A better choice is to for a polynomial of order 3:



The fit has a non-zero loss on training data (thus higher loss on training data), but on new data sets the fit will have a much lower square loss

The example with the polynomials gives one weak principle of selecting prediction models, known as Occam's razor `https://simple.wikipedia.org/wiki/William_of_Ockham`.

Occam's razor as a rule of thumb / weak guideline

Among two hypotheses, which both explain an observation, the simpler is preferable.
The idea behind: the more assumptions, the more likely one of them does not apply.

Above a proof by example X-(**why one must evaluate the performance on a disjoint testset, and never on the training set.**

Why training and test set must be disjoint – a deeper look

We have now seen a case where we get zero training error and very bad test error. If the loss on the training data approximates the expected error, then it should be close to the test error, and be equally high as the test error (and not near zero). So why this does not hold??

Remember the law of large numbers above said:

$$R(f, L, T_n) = \frac{1}{n} \sum_{(x_i, y_i) \in T_n}^{n} L(f(x_i), y_i) \overset{n \to \infty}{\longrightarrow} E_{(x,y) \sim P_{test}}[L(f(x), y)]$$

But observing on training data zero training error while test error is high, contradicts that! What is the problem? Understanding this will give an explanation why the test set must be disjoint from the training set, not just a proof by example.
Need to examine convergence under law of large numbers and what learning means a bit closer!

Suppose we have an infinite sequence of datapoints that are used to construct training datasets:
$(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots, (x_i, y_i), \ldots$

Then we construct a training set at step n by $D_n := \{(x_i, y_i), i \leq n\}$ which just means, that at step n we take the first n points from this sequence to be our training set D_n.

Now the meaning of

$$R(f, L, D_n) \underset{n \to \infty}{\longrightarrow} E_{(x,y) \sim P_{test}}[L(f(x), y)]$$

is that the sequence of numbers
$R(f, L, D_1), R(f, L, D_2), R(f, L, D_3), \ldots, R(f, L, D_{1000}) \ldots$ gets closer and closer to the value $E_{(x,y) \sim P_{test}}[L(f(x), y)]$.

The important thing is: **checking the law of large numbers: this convergence holds for a fixed mapping f.**

When we train a mapping f using a training dataset D_n, and measure on the same dataset D_n, something different happens:

We select the best function f by using the dataset D_n, that is f is not a fixed function, but f depends on D_n – for a new draw of D_n we might choose a different function f by means of our training procedure. the chosen f for D_1 is often not the same as for the chosen f for D_2, D_3, and so on!!!!

so, in fact, training a predictor means that $f = f(D_n)$ becomes a function of the training dataset.

Now we have a sequence of numbers
$R(f(D_1), L, D_1), R(f(D_2), L, D_2), R(f(D_3), L, D_3), \ldots, R(f(D_{1000}), L, D_{1000}) \ldots$

Here, law of large numbers does not apply anymore, because there is no fixed function f but instead a sequence of functions $f(D_n)$, and it is unclear to what it converges.

How does using an independent test set T_k fix that problem? Now we consider

$$R(f(D_n), L, T_k) = \frac{1}{k} \sum_{(x_i,y_i)\in T_k} L(f(D_n)(x_i), y_i)$$

and look into convergence of this number as a function of the test set indices k, not n. If we take the limit across sets $T_1, T_2, T_3, \ldots$ for fixed training data set D_n, then we look at a sequence
$R(f(D_n), L, T_1), R(f(D_n), L, T_2), \ldots, R(f(D_n), L, T_{1000}), \ldots$
This converges

$$R(f(D_n), L, T_k) \underset{k\to\infty}{\longrightarrow} E_{(x,y)\sim P_{test}}[L(f(D_n)(x), y)]$$

to the expectation of the loss of the function $f(D_n)$ which was selected on training dataset D_n, and is a fixed function relative to the independent test sets T_k.[1]

[1] if you would take the limit in n, then no one knows to where the trip goes!

For comparison see the difference:

$$R(f(D_n), L, T_k) \underset{k\to\infty}{\longrightarrow} \quad E_{(x,y)\sim P_{test}}[L(f(D_n)(x), y)]$$
$$R(f(D_n), L, D_n) \underset{n\to\infty}{\longrightarrow} \quad ???$$

> Consequence: using an disjoint testset T_k gives you a reasonable approximation of the expected loss of the learned function $f(D_n)$ under P_{test} : $E_{(x,y)\sim P_{test}}[L(f(D_n)(x), y)]$

The central limit theorem gives you (under additional requirements on the variance of the $L(f(D_n)(x), y)$) that the loss on test data would be normally distributed with a convergence rate of $\frac{1}{\sqrt{n}}$ to the expectation

Out of class:

Actually, one can show that the reported loss by function selection on training data always underestimates the loss of the best possible function.
We select f^* by: $f^* = \text{argmin}_f R(f, L, D_n)$, so the loss of the selected f^* on the dataset D_n is $\inf_f R(f, L, D_n)$.[2]

[2] For experts: technically I mix up here minimum and infimum to avoid lengthy definition hassles. The infimum may not be realized by a function as it is the limit over a function sequence. Yet that is not a storybreaker. Optimization over a large space does not return the infimum anyway. Instead it may get arbitrarily close to the infimum. Formally correct one must do all arguments below for a function f^* which is ϵ-close to the infimum. but then the idea gets harder to grasp.

$$E_{D_n\sim P_{test}^{(\otimes n)}}[\inf_f R(f, L, D_n)] \leq \inf_f E_{D_n\sim P_{test}^{(\otimes n)}}[R(f, L, D_n)]$$

What does that say ?

1.$E_{D_n \sim P_{test}^{(\otimes n)}}[R(f, L, D_n)]$ is the expected loss of a function f.
$\inf_f E_{D_n \sim P_{test}^{(\otimes n)}}[R(f, L, D_n)]$ merely expresses the lowest possible expected error over all possible functions f.

2. $\inf_f R(f, L, D_n)$ is the loss $\inf_f R(f, L, D_n)$ of the selected function on our training data.

So the meaning is: The expectation $E_{D_n \sim P_{test}^{(\otimes n)}}[\cdot]$ of the loss $\inf_f R(f, L, D_n)$ of the selected function on our training data is never larger than the expected error $E_{D_n \sim P_{test}^{(\otimes n)}}[R(f, L, D_n]$ of the best function possible.

Back to in-class: Overfitting

The effect of choosing a mapping with very low training error but high error on new unseen data is called **overfitting** (on the training data). Other mappings may have lower loss on evaluation data or unseen data – and thus be better choices, even if their loss on training data is much higher.

Overfitting has many definitions. low training error and high error on new unseen data is a definition based on observation of loss terms.
There are different explanations like: Choosing a function f by a learning algorithm that is too sensitive to noise in the training data. Selecting a function from a function class with too high complexity.

Machine learning steps in practice:

In practice one often has not only to learn a mapping but also select from a set of hyperparameters such as data preprocessing choices or any kind of algorithm parameters which are not part of the main training process. This makes it necessary to split the available data into training data, validation data and testing data.

Machine learning steps in practice

1– get clarity on input space and output space

2– choose a suitable loss function L (predicted outputs vs ground truth outputs)

3– choose a class of mappings used to predict outputs from inputs (e.g. recurrent neural networks for sequential inputs)

4– choose an algorithm to select a mapping based on training data, that is a way how to compute $f_h = \text{argmin}_f R(f, L, D_n)$

Often one starts not with a given train and test dataset, but just with one dataset, which must be splitted into training, validation and test set.

5– split your dataset into training D_n/validation V_m/test data T_l (use nested crossvalidation if too small sample sizes)

6– get clarity what hyperparameters h will need to be optimized (hyperparameters can be for example regularization parameters, or choices of preprocessing steps or loss functions)

7– for every choice h of hyperparameters:

7–1– train on your training dataset D_n: $f_h = \text{argmin}_f R(f, L, D_n)$

7–2– measure performance on validation dataset $R(f_h, L, V_m)$

Now we have for every choice h of hyperparameters a trained mapping f_h and its performance on validation data $R(f_h, L, V_m)$

8– select best choice of hyperparameters h^* on the validation dataset: $h^* = \text{argmin}_h R(f_h, L, D_n)$

9– train final model on training dataset D_n or on united train and validation dataset $D_n \cup V_m$ using the best choice of hyperparameters h^*: $f^*_{h^*} = \text{argmin}_{f_{h^*}} R(f_{h^*}, L, D_n \cup V_m)$

10– report performance of final model on test set $R(f^*_{h^*}, L, T_l)$

Pitfalls:

- inner/outer crossvalidation for small sample sizes

- only evaluate the final model on test set, never optimize hyperparameters on test set T_l - why?
- what can happen if one class has only 1% abundance, and you do a random split 70-30? - **think what statistical properties to ensure when splitting data**

two main challenges in machine learning

- the training dataset comes from a different distribution, not from the test time distribution P_{test}
- initially, during research, the training dataset comes the test time distribution P_{test}, but P_{test} changes over time

What is if we would know the data generating distribution P in classification problems?

We assumed we do not know the distribution P that generates our new unseen data (x, y).

I like to show you: if you know $P(y, x)$ in case of classification, then one does not need to learn anything, and one needs no training data, but can write down immediately the best classifier.

Now suppose:

- we have a two class classification problem $y \in \mathcal{Y} = \{-1, +1\}$
- we know for every x the conditional distribution $P(y|x)$.
- Lets fix one x. Our mapping $f(x)$ must choose one label $y \in \mathcal{Y} = \{-1, +1\}$ for x.

Let us consider what **expected loss** do we make in point x when we choose $f(x) = +1$ for the loss function $1[f(x) \neq y]$?

$I[T]$ is an indicator function that is one if the logical expression T is *true* and zero if T is *false*. $P(y|x)$ is discrete: Y can take only $\{-1, +1\}$. We know

$$\begin{aligned} E_{(x,y)\sim P(x,y)}[I[f(x) = +1 \neq y]|x] &= I[1 \neq 1]P(y = +1|x) \\ &\quad + I[1 \neq -1]P(y = -1|x) \\ &= 0 + P(y = -1|x) \end{aligned}$$

Result: For $f(x) = +1$ the loss will be $P(y = -1|x)$.

Let us consider what **expected loss** do we make in point x when we choose $f(x) = -1$ for the loss function $1[f(x) \neq y]$?

By the same steps as above, the expected loss will be

$$E[I[f(x) = -1 \neq y]|x] = P(y = +1|x)$$

So:

$$E[I[f(x) \neq y]|x] = \begin{cases} P(y = -1|x) & f(x) = +1 \\ P(y = +1|x) & f(x) = -1 \end{cases}$$

How to minimize our loss ? The loss is either, $P(y = -1|x)$ or $P(y = +1|x)$, depending on the prediction we choose – that is the probability of the opposite class.
Therefore we choose as prediction

$$f_{opt}(x) = \text{argmax}_y P(y|x)$$

, because then the loss will be for the case of two classes $\text{argmin}_y P(y|x)$

Note: if $P(y = +1|x) \neq \{0, 1\}$, then we will always have a non-zero expected loss, no matter what prediction we choose.

if we know $P(y|x)$ for each input x in a classification setting

- in general the expected loss $E_{(x,y)\sim P}[L(f(x), y)]$ under the best possible prediction is **not zero**.
- If we know $P(y|x)$, then we can determine the optimal classifier without training or any datasets. Its prediction is $f_{opt}(x) = \text{argmax}_y P(y|x)$ – we predict the label y that has the highest probability $P(y|x)$ in x.
- $f_{opt}(x) = \text{argmax}_y P(y|x)$ is called **Bayes-classifier**

Back to reality – we do not know $P(x, y)$ – but we can approximate the loss $E_{(x,y)\sim P}[L(f(x), y)]$ by looking at finite datasets.