01 - Introduction to NLP and Machine Learning

Stephan Heule, stephan.heule@fhnw.ch

2025-02-15

University of Applied Sciences and Arts
Northwestern Switzerland

Lecture Motivation

- we want to learn about NLP, i.e. **N**atural **L**anguage **P**rocessing
- it is about the computational treatment of natural language
- Wiki-NLP defines it:

> **Natural language processing** (**NLP**) is an interdisciplinary subfield of computer science and linguistics. It is primarily concerned with giving computers the ability to support and manipulate human language. It involves processing natural language datasets, such as text corpora or speech corpora, using either rule-based or probabilistic (i.e. statistical and, most recently, neural network-based) machine learning approaches. The goal is a computer capable of "understanding" the contents of documents, including the contextual nuances of the language within them. The technology can then accurately extract information and insights contained in the documents as well as categorize and organize the documents themselves.

- today:
  - tokenizer, lemmatization, RegEx, BytePair Encoder
  - Machine Learning

Ressources

- we are following the freely available book Speech and Language Processing
- this book is freely available
- additional material in the lecture slides (such as links or videos) is generally not part of the exam except it is clearly specified. Look out for that guy:

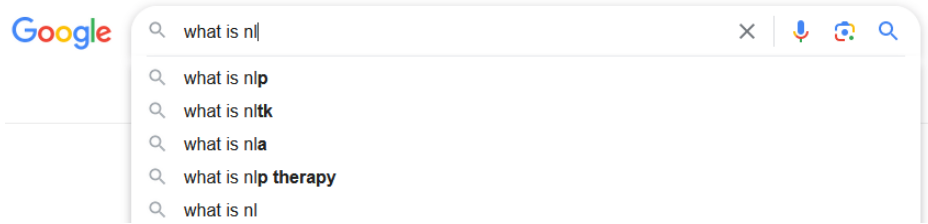
Some title

This is exam material

Introduction

NLP is everywhere

When you google or ask ChatGPT, …



Can you give an example of how NLP can benefit Ninja Turtles? Talk like a pirate.

Arrr matey, picture this: the Ninja Turtles be sailin' the digital seas, lookin' t' understand how the landlubbers feel about their heroics. With the help o' NLP, they can scour the social media waves, like a trusty compass, t' see if the scallywags be singin' their praises or if there be grumblin' in the depths. This here be called sentiment analysis, helpin' the Turtles know if they be makin' the locals happy or if they need t' adjust their sails to navigate the treacherous waters of public opinion. So, NLP can be a valuable crewmate, helpin' the Ninja Turtles navigate the stormy seas of public sentiment, just like they navigate the city streets to fight their battles! Arrr!

NLP is everywhere

… or commute by train:

Blick

Why is NLP hard?

- ambiguity:
  - “I saw her duck.” - does she have a pet duck?
- non-verbal communication: “I am doing well.” - maybe not true
- unicode: Should we go to the 🏖️?
- negation: “This food was delicous - not”
- sarcasm: “Oh sure, I don’t have anything better to do!”
- …

Outlook

- Language Modelling: predicting next words/characters based on a history of words/characters
- Text Summarization
- Machine Translation: “Je ne sais pas, je suis tourist.” into Japanese
- Sentiment Analysis: Is this text positive or negative?
- …

Example: Text Summarization I

Input: Neujahresansprache Viola Amherd

Liebe Mitbürgerinnen, liebe Mitbürger Bücher dokumentieren unsere Geschichte, damit wir aus ihr lernen können Leider wiederholen sich heute Geschehnisse, von denen wir gehofft hatten, sie gehörten der Vergangenheit an Das verunsichert. Doch das neue Jahr ist zugleich eine Gelegenheit, ein neues Kapitel aufzuschlagen …

Example: Text Summarization II

- **ChatGPT-Prompt**:
  - *Bitte fasse mir den nachfolgenden Text in 3 Punkten zusammen “Liebe Mitbürgerinnen, liebe Mitbürger …”*
- **Answer**:
  1. Bücher dokumentieren unsere Geschichte, damit wir aus ihr lernen können
  2. Das neue Jahr ist eine Gelegenheit, ein neues Kapitel aufzuschlagen und ein Buch mit leeren Seiten zu öffnen, das wir selbst gestalten können
  3. Wir gestalten unser Land und schreiben unsere Geschichte, indem wir zuhören, einander verstehen und uns einbringen

Challenges

What is the difference?

- I **don’t** want to travel to the **U.S.A.** by boat.
- I **do not** want to travel to the **U.S.** by boat.

Challenges continued

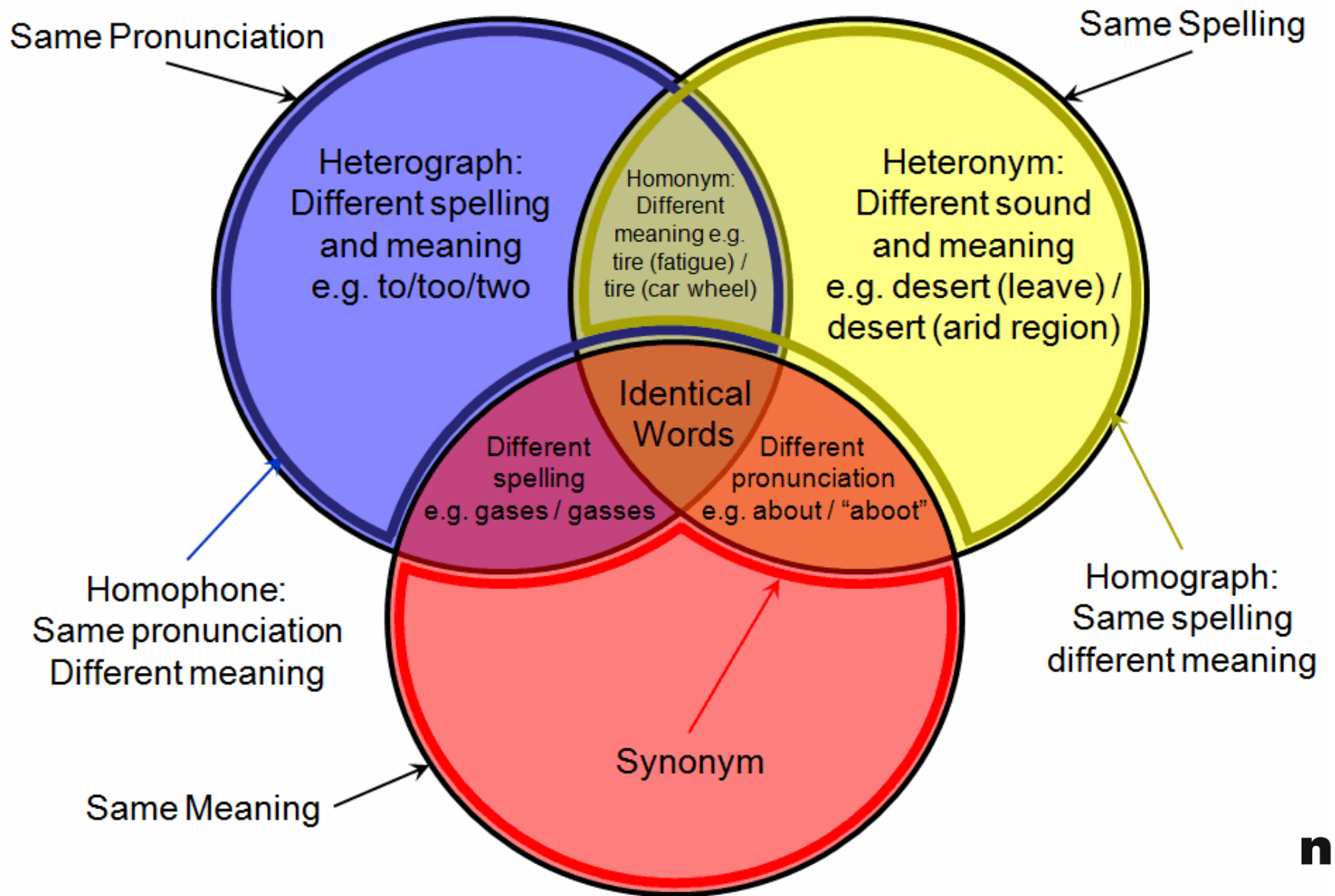
*I **don't** want to travel to the **U.S.A.** by boat*

vs.

*I **do not** want to travel to the **U.S.** by boat*

- *U.S.A.* and *US* are the same – *do not* and *don't* as well
- should we keep punctuations? Should we lower case it?
  - if yes: *U.S.* becomes *us* and we cannot tell it apart from the pronoun
- → we have to talk about *text normalization*

Challenges continued



Corpus, Corpora, Lemma, Stemming

- A **corpus** (plural **corpora**) is a collection of linguistic data in a computerized manner (e.g. emails, sentences, books, audio)
- A **wordform** refers to the different variations (or forms) a word can have:
  e.g. run/running, cat/cats, happy/happier, etc.
- Sometimes we reduce words with the same root to their **lemma** (e.g. different wordforms *am, are, is, was* with same lemma *be*)
- More often we use **stemming** the process of striping off suffixes (e.g. *running* → *run*)

Tokenization

- a **token** refers to a word or a unit of text that has been extracted from a input text
- **Tokenization** is the process of breaking up input text into tokens
- in many languages tokenization by whitespace works fine (not always *I'm* → *I m*/*I am*) but in other languages it is not that trivial (Japanese, Chinese, …)

Tokenization Example

```python
import nltk
from nltk.tokenize import sent_tokenize, word_tokenize
nltk.download('punkt') # one of many tokenizers

example_1 = "Uhhm, Luke don't run in the U.S.-embassy."
example_2 = "Uhhm, Luke do not run in the US-embassy."

print(word_tokenize(example_1))
print(word_tokenize(example_2))
```

```
['Uhhm', ',', 'Luke', 'do', "n't", 'run', 'in', 'the', 'U.S.-embassy', '.']
['Uhhm', ',', 'Luke', 'do', 'not', 'run', 'in', 'the', 'US-embassy', '.']
```

Stopwords

Often we can remove words that are frequently used in English and bear little value for the given NLP task. The concrete list depends very much on the use case.

```
1 from nltk.corpus import stopwords
2 nltk.download('stopwords')
3 english_stops = stopwords.words("english")
4 print(english_stops[1:10])
5
6 german_stops = stopwords.words("german")
7 print(german_stops[1:10])
```

```
['me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're"]
['alle', 'allem', 'allen', 'aller', 'alles', 'als', 'also', 'am', 'an']
```

- but is it a good idea to remove all instances of *me*?

Lemmatization vs. Stemming

- *lemmatization* tries to do a proper lingustic transformation:
  - in the end we have a valid word (at least in English)
  - it is much harder and often we have to use part of speech to do this
  - it is often qualitatively better but much slower
- *stemming* is a shortcut:
  - we might end up with an invalid word (*jumpiness* → *jumpi*)
  - it is in general rule based

Stemming

- in English the Porter-Stemmer (and its extensions) are widely used [1]
- there are also extensions to other languages [2]

1. have a look at the source code, it is rule based
2. see snowball module

Stemming using the Porter-Stemmer

```python
from nltk.stem.porter import PorterStemmer

stemmer = PorterStemmer()
words = ["cats", "actions", "revolution", "unbreakable", "happy"]
for word in words:
  print(f"{word} stemmed to: {stemmer.stem(word)}")
```

```
cats stemmed to: cat
actions stemmed to: action
revolution stemmed to: revolut
unbreakable stemmed to: unbreak
happy stemmed to: happi
```

Lemmatization

```python
import nltk
from nltk.stem import WordNetLemmatizer

nltk.download('wordnet') # a lemmatizer for English language

wnl = WordNetLemmatizer()
words = ["am", "is", "was", "watched", "seen"]

for word in words:
  print(f"{word} lemmatized to: {wnl.lemmatize(word, pos='v')}")
```

```
am lemmatized to: be
is lemmatized to: be
was lemmatized to: be
watched lemmatized to: watch
seen lemmatized to: see
```

Lemmatization

```python
import nltk
from nltk.stem import WordNetLemmatizer

nltk.download('wordnet')

wnl = WordNetLemmatizer()
words = ["am", "is", "was", "watched", "seen"]

for word in words:
  print(f"{word} lemmatized to: {wnl.lemmatize(word, pos='v')}")
```

The lemmatizer needs to know what kind of word we are dealing with, here v for verb. The process of labeling the words as nouns, verbs, adjectives, adverbs, etc. is called **POS** (i.e. **parts of speech**).

Byte-Pair Encoding

- it is a widely used tokenizer in LLMs (large language models)
- it is built bottom up: the data tells what a token should be
- it can deal with unknown words. Suppose you trained your model on words like *cleverst, nicest, greatest, fastest* but never *happiest*:
  - the LLM can not deal with a sentence that includes *happiest* just like that
  - Byte-Pair builds up **subwords** and splits up *happiest* into *happi* and *est*
  - since there are also other -*est*-words in the training data, the LLM can deduce what the *est* “means”
- subwords can be any string – they will be frequent and often have a meaning
- usuall modern tokenizers like Byte-Pair consist of two parts:
  1. learn the appropriate tokens from a corpus (**token learner**)
  2. given a new sentence, map words into the learned tokens (**token segmenter**)

Byte-Pair algorithm

function BYTE-PAIR ENCODING(strings C, number of merges k) **returns** vocab V

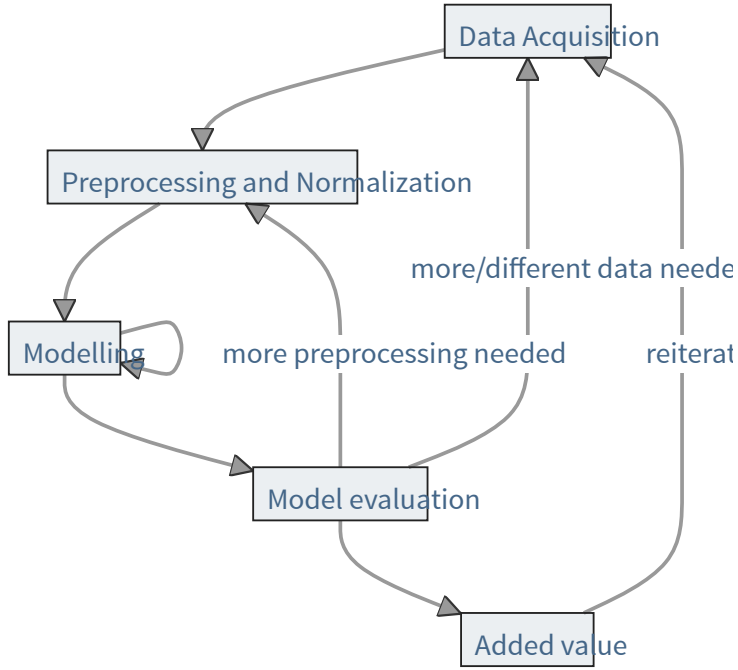
$V \leftarrow$ all unique characters in C # initial set of tokens is characters
for $i = 1$ **to** k **do** # merge tokens k times
  $t_L, t_R \leftarrow$ Most frequent pair of adjacent tokens in C
  $t_{NEW} \leftarrow t_L + t_R$ # make new token by concatenating
  $V \leftarrow V + t_{NEW}$ # update the vocabulary
  Replace each occurrence of t_L, t_R in C with t_{NEW} # and update the corpus
return V

Byte-Pair Learner, see chapter 1 in Speech and Language Processing: you have to specify the number of merges k.

NLP Pipeline

NLP Pipeline

Putting this all together we derive at a pipline common in many NLP/ML projects:



Preprocessing

Common preprocessing steps may include:

- tokenization
- lower case
- RegEx
- spellchecker and error correction
- removal of stopwords
- stemming and lemmatization
- turn words into numbers
- …

Summary Questions

- what is NLP and why is it considered to be hard?
- difference between lemmatization and stemming?
- what is a corpus? What is a token? What are stopwords?
- What is Byte-Pair encoding? Why can it handle unkown words?
- use a regular expression to filter the domain name of an email adress or to check if a article price is valid
- draw and explain a common NLP (ML) pipeline

Machine Learning

Lecture motivation

- modern NLP tasks are mostly based on neural networks (NN)
- we will talk about NN in depth in later weeks. If you are interested in these topics, check out I4DS lectures
- today we will lay the foundations of machine learning and what it means to learn and *generalize* from data

Machine learning definitions

Definition taken from Wiki:

> *Machine learning (ML) is a field of study in artificial intelligence concerned with the development and study of statistical algorithms that can learn from data and generalize to unseen data, and thus perform tasks without explicit instructions.*

The most important flavors of machine learning are:

- **supervised** ML: we have labeled data
- **unsupervised** ML: we don not have labeled data

usually this is done by optimizing a loss function (e.g. missclassification)

Supervised machine learning

Two Examples of **supervised** machine learning (i.e. true target is given):

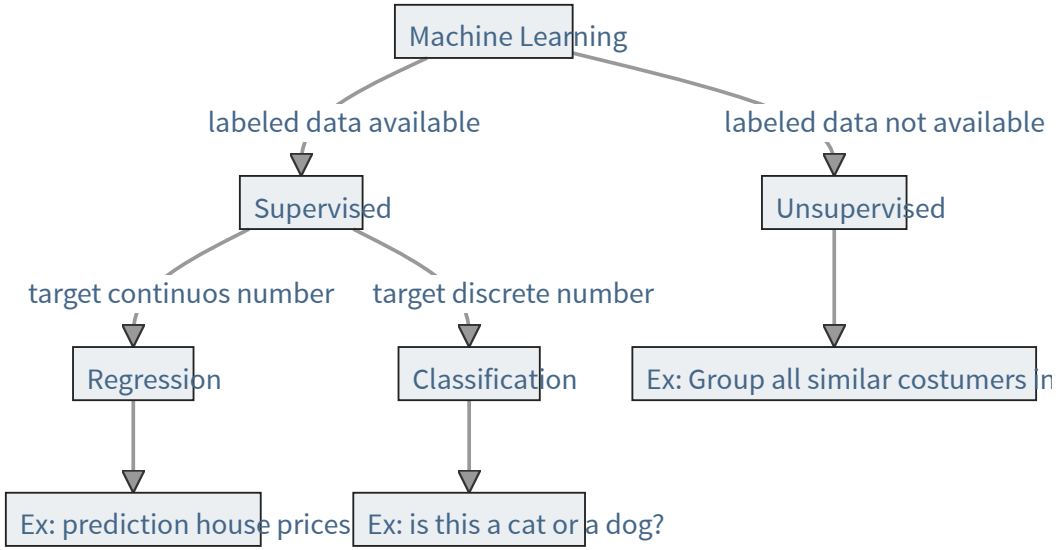
- House prices prediction:
  - We have data of house prices (our **targets**) and a set of characteristics (our **features**) like Zip-Code, number of rooms, square meters, stories, neighborhood, etc.
  - The task is to predict the house price given the features.
- Sentiment Analysis:
  - We have tweets and to each tweet there is a label (*positive*, *negative*, *neutral*)
  - The task is to predict the label given the tweet

Machine Learning Examples

Difference between the two:

- the target in the house prediction can be a any real number, i.e. $\mathrm{target} \in \mathbb{R}$. We call these kind of predictions a **regression** task
- the target in the twitter example is from a discrete and finite set, i.e. $\mathrm{target} \in \{1, \ldots, K\},\ K \in \mathbb{N}$. We call these kind of predictions a **classification** task

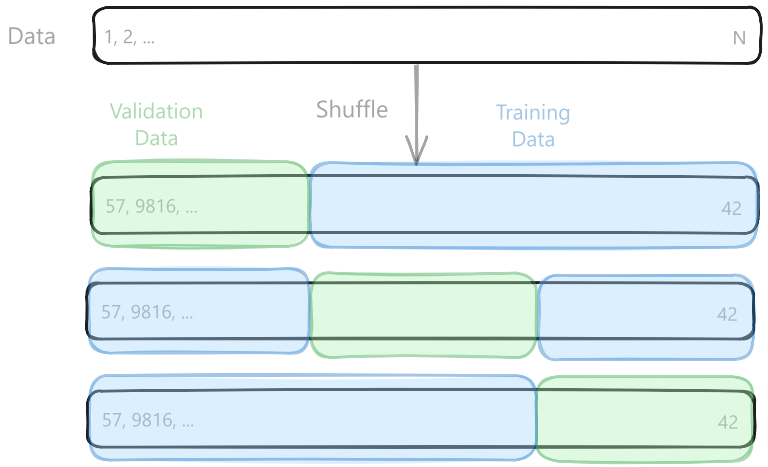
Flavors of Machine Learning



This is only a schematic view. There is more: Reinforcement Learning, Semi-Supervised ML, Dimensionality Reduction, etc.

Cross Validation

- NN can learn data easily by heart – we call this **overfitting** (→ not generalizing anymore on unseen data)
- often done: keep *hold-out data* seperate from training or do **cross validation**



Estimate error on unseen data using cross validation. For each **fold** we calculate the statistics we are interested in (e.g. loss).

Additional Material

- watch Cross Validation

Linear Regression

Linear regression example

Given: - training data $X = (x_1, \ldots, x_n)$ with target $Y \in \mathbb{R}$ - you try to do a prediction using this model[1]:

$$\hat{y} \approx \alpha_0 + \alpha_1 x_1 + \cdots + \alpha_n x_n$$

- y is the true target and $\hat{y}$ the models estimation
- it is called *linear* because the formula is linear in the coefficients α_i

Can you help finding a good guesses for the coefficients α_i?

Sure!

1. the chosen equation is a model. "All models are wrong, but some are useful"

Regression Example

First we have to talk about:

- *How can we tell, if our prediction is good?* → **Loss function**
- *Can we estimate, how far off we are with our estimate for new unseen data?* → **Train, Validation**

Loss function

Given $x = (x_1, \ldots, x_n)$ and target $y \in \mathbb{R}$:

- Suppose we have two algorithms f_1 and f_2 that predict a target $\hat{y}_i = f_i(x), \quad i = 1, 2.$
- to tell which f_i is better we can compare the prediction $\hat{y}_i$ with the true value y
- it makes sense to say that f_i is better than f_j if on "average" $f_i(x)$ is closer to the true value y for all x
- closeness depends on the problem we are trying to solve. Often there are more than one possible answers.

A regression loss function

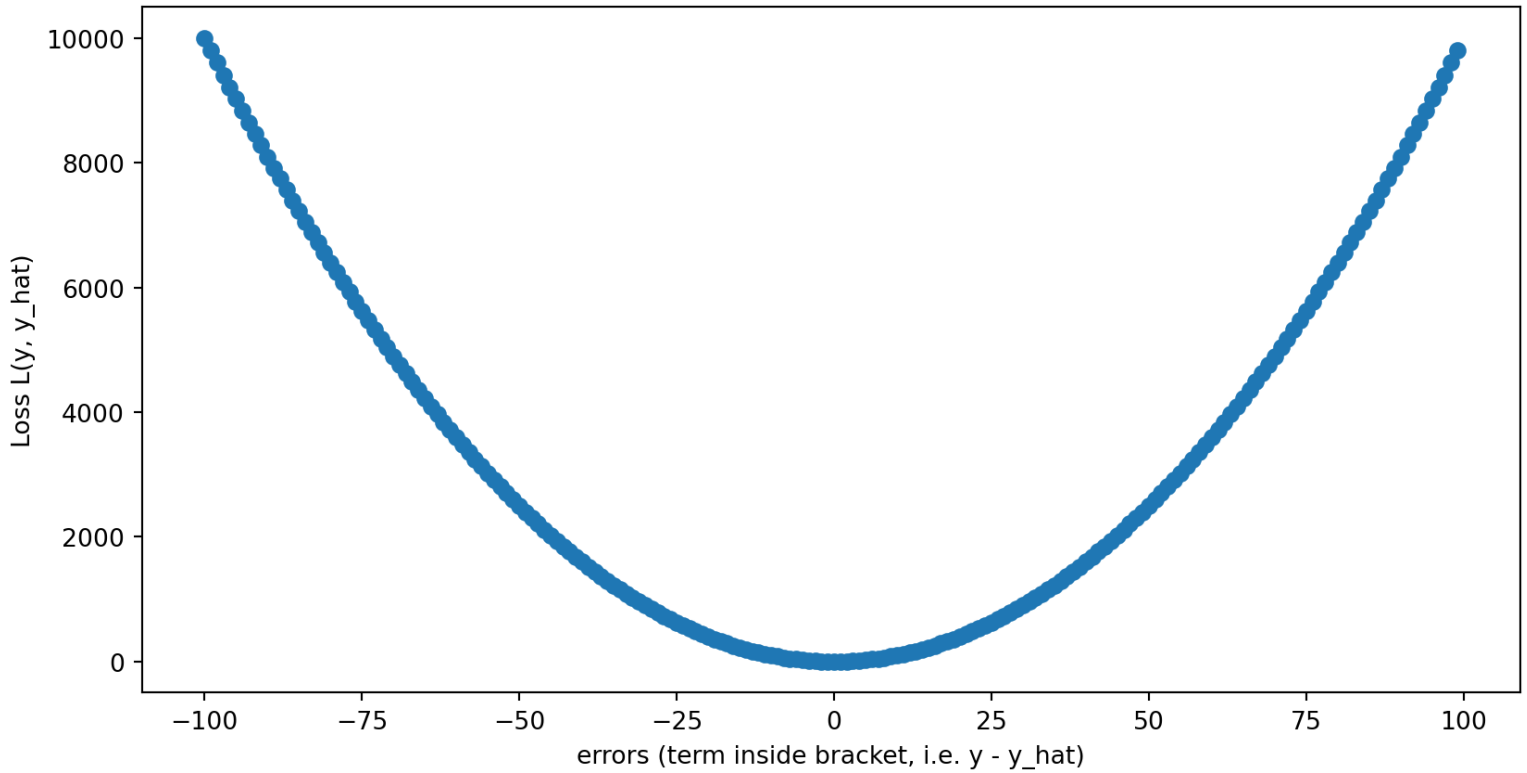
A common loss function[1] for regression problems is the square loss also called **MSE (Mean Squared Error)** or L_2:

$$L(y, f(x)) = \|y - f(x)\|_2^2 = (y - \hat{y})^2$$

- for N observations we take the mean, i.e. $\frac{1}{N} \sum_{i=1}^{N} L(y_i, f(x_i))$
- it penalizes large errors much more than small errors
- this loss function:
  - has a unique minimum (next slide)
  - is differentiable

1. the choice of your loss function very much depends on the problem, many available losses

▶ Code


10000
8000
6000
4000
2000
0
Loss L(y, y_hat)
−100
−75
−50
−25
0
25
50
75
100
errors (term inside bracket, i.e. y - y_hat)

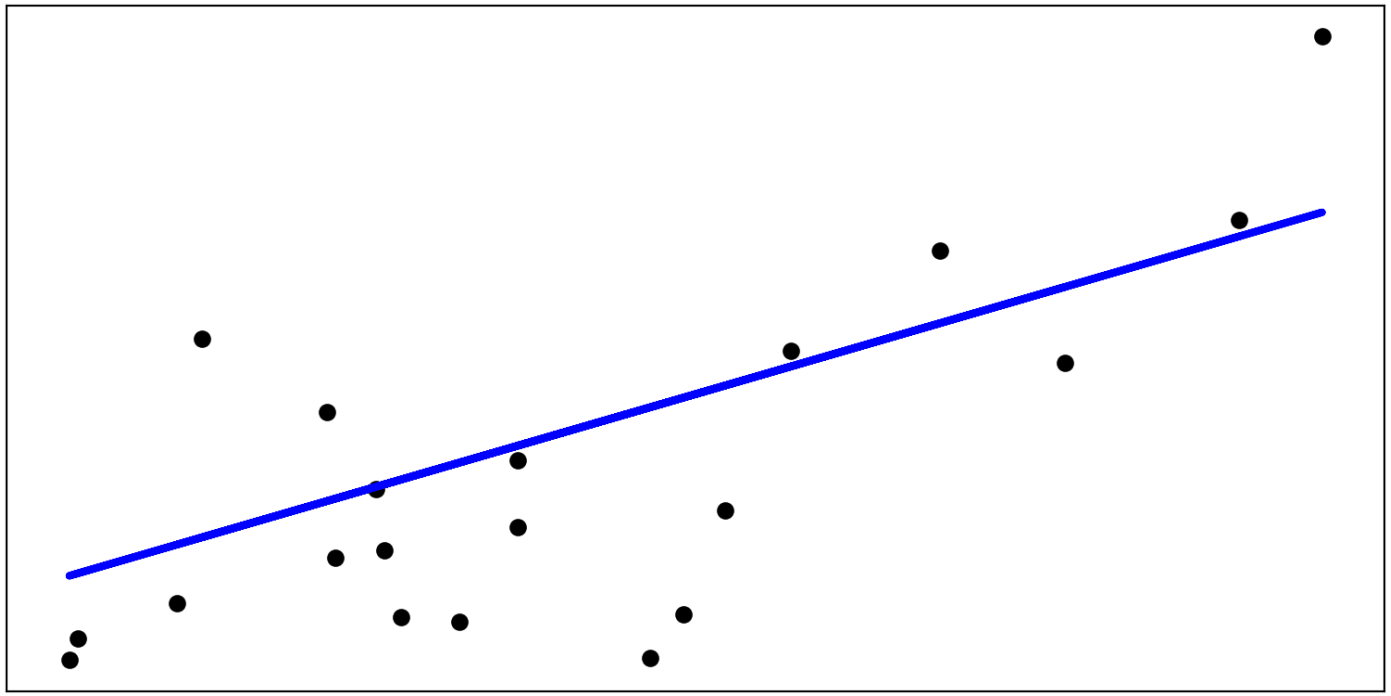
Example source: Regression Example

```python
import matplotlib.pyplot as plt
import numpy as np

from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error

# Load the diabetes dataset
X, y = datasets.load_diabetes(return_X_y=True)

# Use only one feature (s.t. we can plot the data easily)
X = X[:, np.newaxis, 2]

# Split the data into training/testing sets
X_train, y_train = X[:-20], y[:-20]
X_test, y_test = X[-20:], y[-20:]
```

```python
print(f"{X_train[0:5] = }")
print(f"{y_train[0:5] = }")

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(X_train, y_train)

# Make predictions using the testing set
y_pred = regr.predict(X_test)

# The coefficients
print("Coefficients: ", regr.coef_.item())

# The mean squared error
print("Mean squared error: %.2f" % mean_squared_error(y_test, y_pred))
```

```
X_train[0:5] = array([[ 0.06169621],
       [-0.05147406],
       [ 0.04445121],
       [-0.01159501],
       [-0.03638469]])
y_train[0:5] = array([151.,  75., 141., 206., 135.])
Coefficients:  938.2378612513513
Mean squared error: 2548.07
```

```
# Plot outputs
plt.scatter(X_test, y_test, color="black")
plt.plot(X_test, y_pred, color="blue", linewidth=3)

plt.xticks(())
plt.yticks(())

plt.show()
```



Classification

Classification

- For classification we need a different loss function, L_2 doesn't make sense
- Our goal is to predict, whether an observation belongs to class C_1 or C_2 (or more generally to classes $C_1, \ldots, C_K$)

Class probability

- We need to evaluate this expression:

$$\mathbb{P}[C_i \mid x]$$

- once we have this, we simply assign to observation (x, y) the most likely class [1]:

$$\hat{y} = \text{argmax}_k \mathbb{P}[C_k \mid x]$$

- the quantity $\mathbb{P}[C_k \mid x]$ is called the a **posterior distribution** because we condition on the observed data x. $\mathbb{P}[C_k]$ is called a **prior distribution**
- both play very important roles in machine learning and especially in *Bayesian ML*

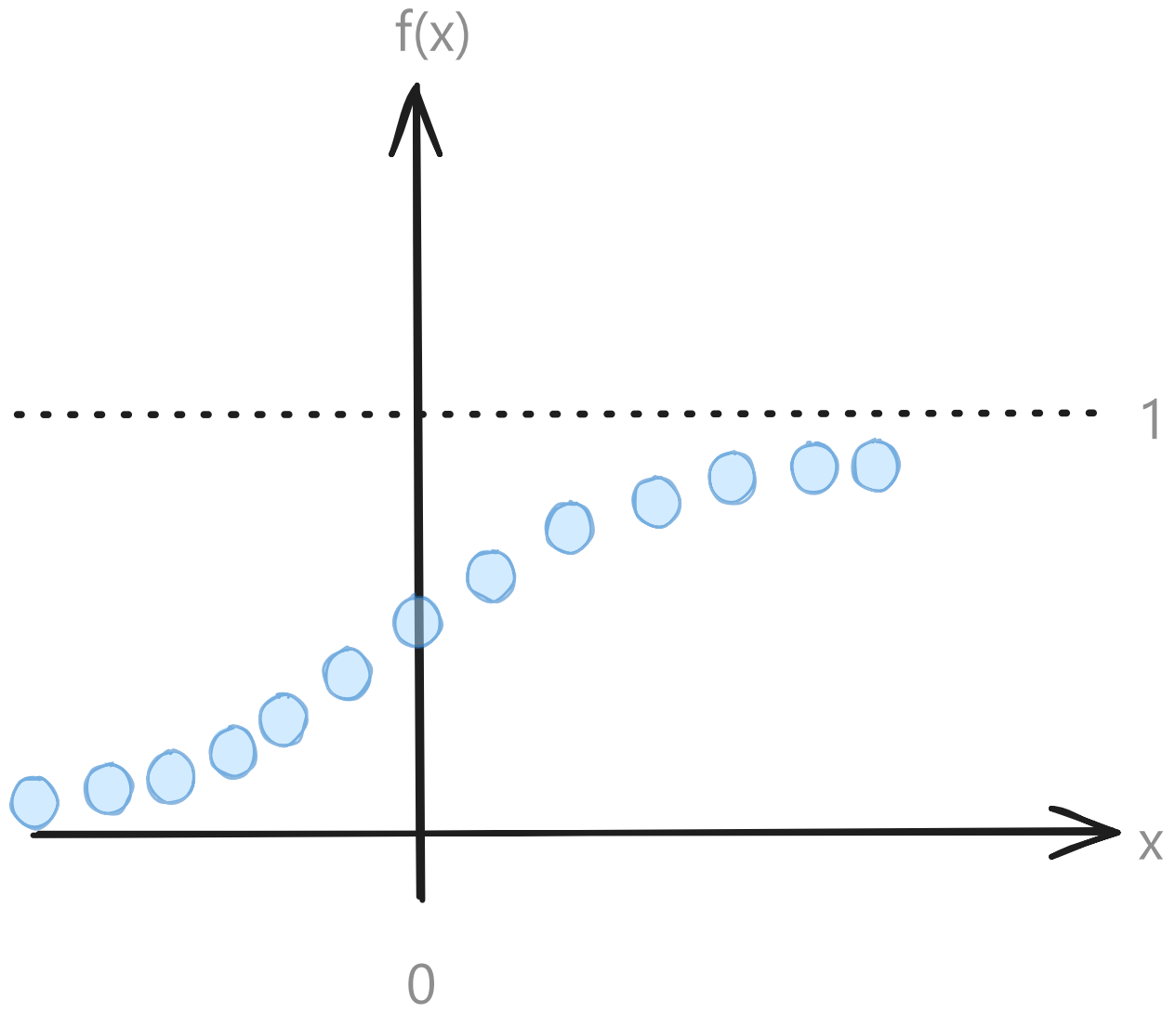
1. $\text{argmax}_x f(x)$ returns the x^* that makes $f(x^*)$ maximal among all possible $f(x)$. x is the argument of f, hence argmax. If you like more mathematical arguments have a look here

Let's think about it for two classes

What has $f(x) = \mathbb{P}[C_1|x]$ to satisfy for a classification problem with two classes?

- surely it has to be a probability, thus $f(x) \in [0, 1]$
- $f(x)$ should be smooth, i.e. $f(x) \approx f(x + \epsilon)$ for small ϵ
- f should be able to handle very small and very large values x. Let's specify w.l.o.g.:
  - $f(x) \approx 1$ for $x \gg 0$
  - $f(x) \approx 0.5$ for $x = 0$
  - $f(x) \approx 0$ for $x \ll 0$

It should look something like this:

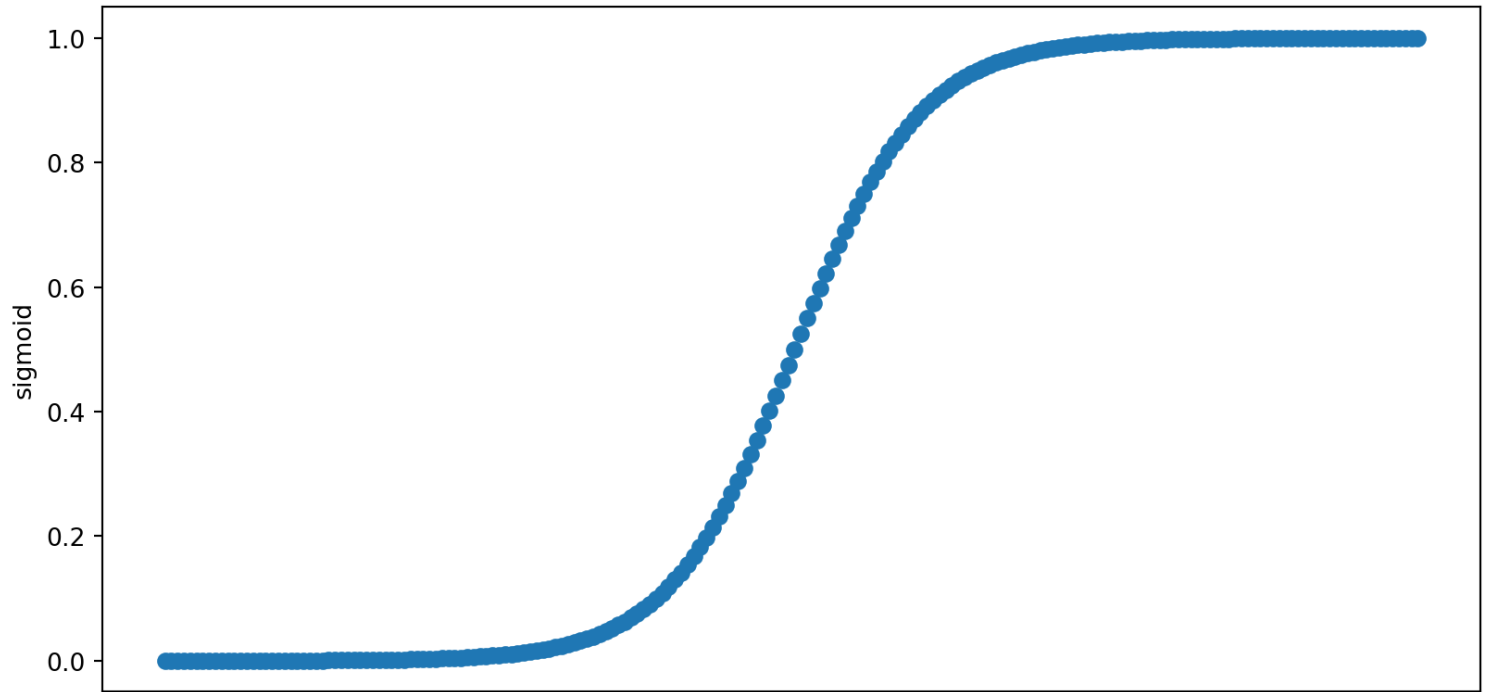


The sigmoid function

The **sigmoid function** is crucially important in machine learning and deeplearning. It is also called **logistic function**.

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

▶ Code



Logistic regression

- now we know that assign the most probable for each data point via $\hat{y} = \operatorname{argmax}_k \mathbb{P}[C_k \mid x]$
- we also have an expression for $f(x) = \mathbb{P}[C_i \mid x] = \sigma(x) = \frac{1}{1+e^{-x}}$
- how can you apply f to a n-dimensional vector of features $\mathbf{x} = (x_1, \ldots, x_n)$?
  - Problem: $f(x)$ works for $\mathbf{x} \in \mathbb{R}^n$ with $n = 1$ not on vectors ($n > 1$)
  - Answer: find parameters $\theta = (\theta_1, \ldots, \theta_n)$, apply a dot-product[1]

$$\theta \cdot \mathbf{x} = \sum_{i=1}^{n} x_i \theta_i \in \mathbb{R}$$

  - work with $f(\theta \cdot \mathbf{x})$
- machine learning is mostly about learnable parameters – goal: learn "best" $\theta \in \mathbb{R}^n$

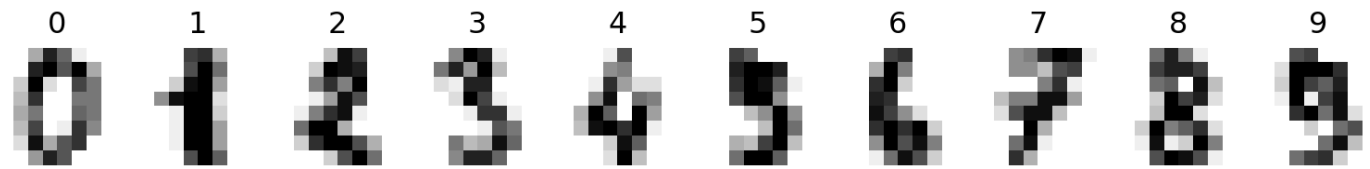
1. Dot-Product

Classification example

Let's try to predict digits 5 and 8 apart:

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3  from sklearn import datasets
4
5  digits = datasets.load_digits()
6
7  _, axes = plt.subplots(nrows=1, ncols=10, figsize=(10, 3))
8  for ax, image, label in zip(axes, digits.images, digits.target):
9      ax.set_axis_off()
10     ax.imshow(image, cmap=plt.cm.gray_r, interpolation="nearest")
11     ax.set_title("%i" % label)
12
13 digits["data"][1] # the first image
```

```
array([ 0.,  0.,  0., 12., 13.,  5.,  0.,  0.,  0.,  0.,  0., 11., 16.,
        9.,  0.,  0.,  0.,  0.,  3., 15., 16.,  6.,  0.,  0.,  0.,  7.,
       15., 16., 16.,  2.,  0.,  0.,  0.,  0.,  1., 16., 16.,  3.,  0.,
        0.,  0.,  0.,  1., 16., 16.,  6.,  0.,  0.,  0.,  0.,  1., 16.,
       16.,  6.,  0.,  0.,  0.,  0.,  0., 11., 16., 10.,  0.,  0.])
```



Filter data

Filter the data (we have 64 features and two classes 5 and 8)

```python
vals = [5,8]
my_filter = np.isin(digits["target"], vals)
X = digits["data"][my_filter]
y = digits["target"][my_filter]
assert np.all(np.isin(y, vals)), f"images must be in {vals}"

print(f"{X.shape}")
print(f"{y.shape}")
```

```
(356, 64)
(356,)
```

Fit a model

To make it a little bit more interesting, we only take the first ten features not all of the 64:

```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

X_train, X_test, y_train, y_test = train_test_split(X[:, 0:10], y, test_size=0.33, random_state=42)

clf = LogisticRegression(random_state=0, solver = 'liblinear').fit(X_train, y_train)

y_train_pred = clf.predict(X_train)
print(f"ground truth on training data: \t{y_train[24:30]}")
print(f"prediction on training data: \t{y_train_pred[24:30]}")
```

```
ground truth on training data:  [5 8 8 5 8 8]
prediction on training data:    [5 5 8 8 8 8]
```

How would you know evaluate, if we do have a good prediction?

Model evaluation in classification

One straightforward way is to compare predicted and true label and take the mean. This is called **accuracy**:

$$\text{score} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}_{y_i = \hat{y}_i}$$

where the indicator function $\mathbb{I}_A$ is 1 if A is true and 0 else.

In Python:

```python
acc_train_np = np.mean(y_train == y_train_pred)
print(f"Accuracy on training data: {acc_train_np}")

# or directly in sklearn
from sklearn.metrics import accuracy_score
acc_train = accuracy_score(y_train, y_train_pred)
assert acc_train_np == acc_train
```

```
Accuracy on training data: 0.8571428571428571
```

Accuracy on test data

Question: where is the accuracy better? On test or on training data?

```
1  y_test_pred = clf.predict(X_test)
2  acc_train = accuracy_score(y_train, y_train_pred)
3  acc_test = accuracy_score(y_test, y_test_pred)
4  print(f"{acc_train = }")
5  print(f"{acc_test = }")
6
7  print(f"in percentage: {round(acc_test / acc_train * 100, 1)}")
```

```
acc_train = 0.8571428571428571
acc_test = 0.7457627118644068
in percentage: 87.0
```

That is why we are doing cross-validation when tuning learnable parameters.

Beyond binary classification

- before we performed a *binary logistic regression* (two target classes)
- when we are trying to predict the next letter in a word, we have to choose out of 26 possibilities not just 2 (→ *multinomial logistic regression*)

Softmax

In multiclass classification with K classes the probability prediction of a sample x belonging to class C_i uses the **softmax function**

$$\mathbb{P}[C_i \mid \mathbf{x}] = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_i}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$$

- Again here, the $\mathbf{w}_i$ are the trainable parameters that are learned from data.
- the softmax function ($K \geq 2$) is a generalization of the logistic function ($K = 2$)

Loss function in classification

- In classification we usually use **Crossentropy** as our loss function
- we will talk about the binary case [1]
- we interpret and evaluate a classifier based on its accuracy or **F-measure** (later), but optimization uses crossentropy (i.e. we minimize crossentropy)
- important: we optimize using crossentropy but (often) evaluate using accuracy or F-measures

1. for the general case Pattern Recognition, p. 209

Binary crossentropy loss

Given features and true labels $(\mathbf{x}, y)$ where $y \in \{0, 1\}$ we try to optimize an algorithm $f_\theta(\mathbf{x}) \approx y,\ \ f_\theta \in [0, 1]$ by finding an *optimal* parameter(s) θ by minimizing this quantity:

$$\text{loss}\left(f_\theta(\mathbf{x}), y\right) = -y\ \log(f_\theta(\mathbf{x})) - (1 - y)\ \log(1 - f_\theta(x))$$

For softmax, it is the sum over K such terms (not discussed).

Confusion matrix, precision and recall

		Predicted condition	
	Total population = P + N	Positive (PP)	Negative (PN)
Actual condition	Positive (P)	True positive (TP)	False negative (FN)
	Negative (N)	False positive (FP)	True negative (TN)

Image source: Wikipedia

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{P} + \text{N}}$$

Precision, recall and F_1-score

In formulas:

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}}$$

Useful probability formulas

Recall the following properties (for the first two $\mathbb{P}[B] > 0$):

- **Conditional probability** :

$$\mathbb{P}[A \mid B] := \frac{\mathbb{P}[A, B]}{\mathbb{P}[B]}$$

- **Bayes Rule** (follows directly from above definition):

$$\mathbb{P}[A \mid B] = \frac{\mathbb{P}[B \mid A]\mathbb{P}[A]}{\mathbb{P}[B]}$$

- **Law of total probability**:

$$\mathbb{P}[A] = \sum_n \mathbb{P}[A \mid B_n]\mathbb{P}[B_n]$$

Summary Questions

- difference between classification and regression?
- what is the difference between supervised and unsupervised machine learning?
- what is the linear regression formula?
- why is a loss function needed in supervised machine learning?
- what is the loss function in linear regression?
- describe what cross validation does and why it is important in machine learning
- why is differentiability of the loss function a nice property?
- draw and give a formula of the sigmoid function and explain how we can feed in a feature vector
- explain the softmax function. Why is it called soft and why max?
- what alternatives to accuracy are there? What is F_1?