<table>
<thead>
<tr>
<th>Date</th>
<th>Day</th>
<th>no.</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oct 05</td>
<td>Mon</td>
<td>1</td>
<td>A Hands-On Introduction</td>
</tr>
<tr>
<td>Oct 07</td>
<td>Wed</td>
<td>2</td>
<td>Bayesian Decision Theory, Generative Probabilistic Models</td>
</tr>
<tr>
<td>Oct 12</td>
<td>Mon</td>
<td>3</td>
<td>Discriminative Probabilistic Models</td>
</tr>
<tr>
<td>Oct 14</td>
<td>Wed</td>
<td>4</td>
<td>Maximum Margin Classifiers, Generalized Linear Models</td>
</tr>
<tr>
<td>Oct 19</td>
<td>Mon</td>
<td>5</td>
<td>Estimators; Overfitting/Underfitting, Regularization, Model Selection</td>
</tr>
<tr>
<td>Oct 21</td>
<td>Wed</td>
<td>6</td>
<td>Bias/Fairness, Domain Adaptation</td>
</tr>
<tr>
<td>Oct 26</td>
<td>Mon</td>
<td>-</td>
<td>no lecture (public holiday)</td>
</tr>
<tr>
<td>Oct 28</td>
<td>Wed</td>
<td>7</td>
<td>Learning Theory I</td>
</tr>
<tr>
<td>Nov 02</td>
<td>Mon</td>
<td>8</td>
<td>Learning Theory II</td>
</tr>
<tr>
<td>Nov 04</td>
<td>Wed</td>
<td>9</td>
<td>Deep Learning I</td>
</tr>
<tr>
<td>Nov 09</td>
<td>Mon</td>
<td>10</td>
<td>Deep Learning II</td>
</tr>
<tr>
<td>Nov 11</td>
<td>Wed</td>
<td>11</td>
<td>Unsupervised Learning</td>
</tr>
<tr>
<td>Nov 16</td>
<td>Mon</td>
<td>12</td>
<td>project presentations</td>
</tr>
<tr>
<td>Nov 18</td>
<td>Wed</td>
<td>13</td>
<td>buffer</td>
</tr>
</tbody>
</table>
The goal of (supervised) machine learning is

- use a training set \( D = \{(x^1, y^1), \ldots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y} \)
- to find a prediction function \( f : \mathcal{X} \rightarrow \mathcal{Y} \) (="learning")
- that works well on future data.
The goal of (supervised) machine learning is

- use a training set \( \mathcal{D} = \{(x^1, y^1), \ldots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y} \)
- to find a prediction function \( f : \mathcal{X} \rightarrow \mathcal{Y} \) (="learning")
- that works well on future data.

Many learning techniques have been developed, such as

- decision trees
- (k-)nearest neighbor
- Perceptron
- Boosting ← today
- Artificial Neural Networks ← today
The goal of (supervised) machine learning is

- use a training set  \( D = \{ (x^1, y^1), \ldots, (x^n, y^n) \} \subset \mathcal{X} \times \mathcal{Y} \)
- to find a prediction function  \( f : \mathcal{X} \to \mathcal{Y} \) (="learning")
- that works well on future data.

Many learning techniques have been developed, such as

- decision trees
- (k-)nearest neighbor
- Perceptron
- Boosting ← today
- Artificial Neural Networks ← today

Some phenomena are universal:

- models with too small complexity underfit the data
  - high training error, high test error
- models with too high complexity overfit the data
  - low training error, high test error
Given: training examples $D = \{(x_1, y_1), \ldots, (x_n, y_n)\} \subset \mathcal{X} \times \mathcal{Y}$ with $\mathcal{Y} = \{\pm 1\}$.

Problem:
- it’s hard to guess a strong (=good) classifier.
- it’s easy to guess weak (=slightly better than random) classifiers.

Question [Kearns, Valiant. 1988/89]:
- Given enough weak classifiers, can one always construct a strong one?

Answer [Schapire. 1990]:
- Yes, by Boosting!
Boosting – Weak Classifiers

For example: if our features are

<table>
<thead>
<tr>
<th>property</th>
<th>possible values</th>
</tr>
</thead>
<tbody>
<tr>
<td>eye color</td>
<td>blue/brown/green</td>
</tr>
<tr>
<td>handsome</td>
<td>yes/no</td>
</tr>
<tr>
<td>height</td>
<td>short/tall</td>
</tr>
<tr>
<td>sex</td>
<td>male (M)/female (F)</td>
</tr>
<tr>
<td>soccer fan</td>
<td>yes/no</td>
</tr>
</tbody>
</table>

define (weak) classifiers:

\[
\begin{align*}
    h_1(x) &= \begin{cases} 
        +1 & \text{if eye color = brown} \\
        -1 & \text{otherwise.}
    \end{cases} \\
    h_2(x) &= \begin{cases} 
        +1 & \text{if eye color = blue} \\
        -1 & \text{otherwise.}
    \end{cases} \\
    h_3(x) &= \begin{cases} 
        +1 & \text{if eye color = green} \\
        -1 & \text{otherwise.}
    \end{cases} \\
    h_4(x) &= \begin{cases} 
        -1 & \text{if eye color = brown} \\
        +1 & \text{otherwise.}
    \end{cases} \\
    h_5(x) &= \begin{cases} 
        +1 & \text{if handsome = yes} \\
        -1 & \text{otherwise.}
    \end{cases} \\
    h_6(x) &= \begin{cases} 
        -1 & \text{if handsome = yes} \\
        +1 & \text{otherwise.}
    \end{cases}
\end{align*}
\]

Set of all possible combinations: \( \mathcal{H} = \{ h_1, \ldots, h_J \} \).
input training set $D$, set of weak classifiers $\mathcal{H}$, number of iterations $T$.

\[
d_1 = d_2 = \cdots = d_n = 1/n \quad \text{(weight for each example)}
\]

for $t=1,\ldots,T$ do

for $h \in \mathcal{H}$ do

$e_t(h) = \sum_{i=1}^{n} d_i \left[ h(x^i) \neq y^i \right] \quad \text{(weighted training error)}$

$h_t = \text{argmin}_{h \in \mathcal{H}} e_t(h) \quad \text{("best" of the weak classifiers)}$

$\alpha_t = \frac{1}{2} \log\left(\frac{1-e_t(h_t)}{e_t(h_t)}\right) \quad \text{(classifier importance, } \alpha_t = 0 \text{ if } e_t(h_t) = \frac{1}{2})$

for $i = 1,\ldots,n$ do

$\tilde{d}_i \leftarrow d_i \times \begin{cases} e^{\alpha_t} & \text{if } h_t(x^i) \neq y^i, \\ e^{-\alpha_t} & \text{otherwise.} \end{cases}$

end for

for $i = 1,\ldots,n$ do

$d_i \leftarrow \tilde{d}_i / \sum_i \tilde{d}_i$

end for

output classifier: $f(x) = \text{sign} \sum_{t=1}^{T} \alpha_t h_t(x)$
Task: $\mathcal{X} = \mathbb{R}^2$, weak classifiers look at each dimension separately.
Iteration $t = 1$, $d_1, \ldots, d_n = \left( \frac{1}{11}, \frac{1}{11}, \frac{1}{11}, \frac{1}{11}, \frac{1}{11}, \frac{1}{11}, \frac{1}{11}, \frac{1}{11}, \frac{1}{11}, \frac{1}{11} \right)$
Iteration $t = 1$, best weak classifier, $e_1(h_1) = \frac{1}{11}$, $\alpha_1 = 1.15$
Iteration $t = 1$, best weak classifier, $e_1(h_1) = \frac{1}{11}$, $\alpha_1 = 1.15$
Iteration $t = 2$, $d_1, \ldots, d_n \approx \left( \frac{1}{20}, \frac{1}{20}, \frac{1}{20}, \frac{1}{20}, \frac{1}{20}, \frac{1}{20}, \frac{1}{20}, \frac{1}{20}, \frac{1}{20}, \frac{1}{20} \right)$
Iteration $t = 2$, best weak classifier, $e_2(h_2) = \frac{1}{20}$, $\alpha_2 = 1.47$
Iteration $t = 2$, best weak classifier, $e_2(h_2) = \frac{1}{20}$, $\alpha_2 = 1.47$
AdaBoost – Example

Iteration $t = 3$
Iteration $t = 3$
AdaBoost – Example

Iteration $t = 4$
AdaBoost – Example

Iteration $t = 5$
Final classifier: $f(x) = \text{sign}(1.15h_1(x) + 1.5h_2(x) + \cdots + 0.9h_5(x))$
Artificial Neural Networks are predictive models inspired by (early) Neuroscience.

Main idea:
- stack layers of simple elements ("neurons")
- one layer’s outputs are next layer’s input.

Network parametrizes a function:
- each neuron $N_i$ computes a linear/affine function
  $$a_i = \langle w_i, x_{\text{input}} \rangle + b_i$$
  "activation"

  followed by a componentwise non-linear transformation, $\sigma : \mathbb{R} \to \mathbb{R}$,
  $$o_i = \sigma(a_i)$$

  e.g. $\sigma(t) = \max\{0, t\}$

Since approx. 2012, re-popularized under the name of Deep Learning → lectures 7 and 8.
Understanding Machine Learning Methods
Decision Theory (for Supervised Learning Problems)

Goal:
• Understand existing algorithms
• Develop new algorithms with specific (optimal?) properties

For this, we’ll rely on mathematics. Forget about implementation, finite data etc... (for now)

Notation

We treat all quantities of interest as random variables:

• input: random variable, $X$, taking values $x \in \mathcal{X}$
  (we think of $\mathcal{X}$ as continuous, but use discrete notation for simplicity)

• output: random variable, $Y$, taking values $y \in \mathcal{Y}$.

• joint probability distribution $p(X = x, Y = y) = p(Y = y | X = x) p(X = x)$
  ▶ $p(X = x)$: how likely is it that any $x \in \mathcal{X}$ will occur?
  ▶ $p(Y = y | X = x)$: what’s the probability that $y \in \mathcal{Y}$ is the correct answer for $x \in \mathcal{X}$?

• we write $p(x, y)$ for of $p(X = x, Y = y)$, $p(y | x)$ instead of $p(Y = y | X = x)$, etc.
First first look at classification, $\mathcal{Y} = \{1, \ldots, M\}$, or $\mathcal{Y} = \{-1, +1\}$.

**Question:** What’s the best classifier for a fully known problem?
First first look at classification, \( \mathcal{Y} = \{1, \ldots, M\} \), or \( \mathcal{Y} = \{-1, +1\} \).

**Question:** What’s the best classifier for a fully known problem?

**Definition (Generalization error)**

Let \( c : \mathcal{X} \rightarrow \mathcal{Y} \) be a decision rule. The generalization error, \( \mathcal{R} \), of \( c \) is the probability of \( c \) making a wrong prediction, i.e.

\[
\mathcal{R}(c) := \Pr_{(x,y) \sim p(x,y)} \{ c(x) \neq y \}.
\]
First first look at classification, $Y = \{1, \ldots, M\}$, or $Y = \{-1, +1\}$.

**Question:** What’s the best classifier for a fully known problem?

**Definition (Generalization error)**

Let $c : \mathcal{X} \to Y$ be a decision rule. The *generalization error*, $R$, of $c$ is the probability of $c$ making a wrong prediction, i.e.

$$R(c) := \Pr_{(x,y) \sim p(x,y)} \{c(x) \neq y\}.$$ 

**Definition (Bayes Classifier, Bayes Risk)**

The smallest achievable generalization error,

$$R_{\text{Bayes}} = \min_{c : \mathcal{X} \to \mathcal{Y}} R(c)$$

is called *Bayes error*. A classifier, $c^*$, that achieves the base error is called **Bayes classifier**.
Lemma

For any $x \in \mathcal{X}$ with $p(x) > 0$, a Bayes classifier has the decision rule

$$
\hat{c}(x) \in \arg\max_{y \in \mathcal{Y}} p(y|x) \quad \text{or (equivalently)} \quad \hat{c}(x) \in \arg\max_{y \in \mathcal{Y}} p(x, y) \quad (*)
$$

Proof. First: both rules are equivalent, because

$$
\arg\max_{y} p(x, y) = \arg\max_{y} p(y|x)p(x) = \arg\max_{y} p(y|x).
$$

1) We rewrite the risk in terms of per-point contributions. For any $c : \mathcal{X} \rightarrow \mathcal{Y}$

$$
\mathcal{R}(c) = \Pr_{(x,y) \sim p(x,y)} \{c(x) \neq y\}
$$

$$
= \mathbb{E}_{(x,y) \sim p} \left[ c(x) \neq y \right]
$$

$$
= \mathbb{E}_{x \sim p(x)} \mathbb{E}_{y \sim p(y|x)} \left[ c(x) \neq y \right]
$$

$$
= \sum_{x \in \mathcal{X}} p(x) \underbrace{\mathbb{E}_{y \sim p(y|x)} \left[ c(x) \neq y \right]}_{=:\mathcal{R}_{x}(c)}
$$
Lemma

For any $x \in \mathcal{X}$ with $p(x) > 0$, a Bayes classifier has the decision rule

$$\hat{c}(x) \in \arg\max_{y \in \mathcal{Y}} p(y|x) \quad \text{or (equivalently)} \quad \hat{c}(x) \in \arg\max_{y \in \mathcal{Y}} p(x,y) \quad (\ast)$$

1. For any $x \in \mathcal{X}$:

$$R_x(\hat{c}) = \mathbb{E}_{y \sim p(y|x)} [\hat{c}(x) \neq y] = \mathbb{E}_{y \sim p(y|x)} [\hat{c}(x) \neq y]$$

$$= 1 - \mathbb{E}_{y \sim p(y|x)} [\hat{c}(x) = y]$$

$$= 1 - \mathbb{P}(Y = \hat{c}(x) | x) \geq p(y|x) \text{for all } y \in \mathcal{Y}$$

$$\leq 1 - p(Y = c^*(x) | x)$$

$$= \mathbb{E}_{y \sim p(y|x)} [c^*(x) \neq y] = R_x(c^*)$$

2. Let $A$ be the set of points where $c^*$ does not fulfill $(\ast)$. Then, for all $x \in A$:

$$p(Y = \hat{c}(x) | x) > p(Y = c^*(x) | x) \quad \text{and} \quad R_x(\hat{c}) < R_x(c^*)$$
Lemma

For any $x \in \mathcal{X}$ with $p(x) > 0$, a Bayes classifier has the decision rule

$$\hat{c}(x) \in \arg\max_{y \in \mathcal{Y}} p(y|x) \quad \text{or (equivalently)} \quad \hat{c}(x) \in \arg\max_{y \in \mathcal{Y}} p(x, y) \quad (*)$$

1) $R(c) = \sum_{x \in \mathcal{X}} p(x) R_x(c)$

2) for every $x \in \mathcal{X}$: $R_x(\hat{c}) \leq R_x(c^*)$

3) for every $x \in A$: $R_x(\hat{c}) < R_x(c^*) = R_{\text{Bayes}}$

4) Consequently, if there’s at least one point in $x \in A$ with $p(x) > 0$, then

$$R(\hat{c}) < R(c^*)$$

But that inequality is impossible by the definition of $R_{\text{Bayes}}$. Therefore, no such point exists.

In summary: in all points $x$ with $p(x) > 0$, the Bayes classifier fulfills $(*)$. 

20 / 36
In binary classification we can write $c^*$ in closed form:

**Lemma**

For $\mathcal{Y} = \{-1, +1\}$, the Bayes classifier is given by

\[
c^*(x) = \text{sign} \left[ \log \frac{p(x, +1)}{p(x, -1)} \right],
\]

as well as

\[
c^*(x) = \text{sign} \left[ \log \frac{p(+1|x)}{p(-1|x)} \right].
\]

Proof: Exercise.
Should we use $c^*$ to decide for every problem?

- $c^*$ is optimal when trying to minimize the number of wrong decision.
- That’s often a good strategy, but not always.

**Reminder**

To evaluate a learning task, we use a *loss function* $\ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$.

$\ell(y, \bar{y})$ is the loss incurred when predicting $\bar{y}$ if the correct answer is $y$. 

Example: Doctor’s dilemma

A patient has a cough but no fever. Should you make her a COVID19 suspect?

- $x$: symptoms.
- $y \in \{\text{yes}, \text{no}\}$: COVID19

- $\ell(\text{yes}, \text{yes}) = 0$ (you did your job well)
- $\ell(\text{no}, \text{no}) = 0$ (you did your job well)
- $\ell(\text{yes}, \text{no}) = 50$ (the patent goes home and might infect many others)
- $\ell(\text{no}, \text{yes}) = 1$ (the patient has to take an unpleasant unnecessary test)
Should we use $c^*$ to decide for every problem?

- $c^*$ is optimal when trying to minimize the number of wrong decision.
- That’s often a good strategy, but not always.

Reminder

To evaluate a learning task, we use a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$.

$\ell(y, \bar{y})$ is the loss incurred when predicting $\bar{y}$ if the correct answer is $y$.

Example: Doctor’s dilemma

A patient has a cough but no fever. Should you make her a COVID19 suspect?

$x$: symptoms. $y \in \{\text{yes, no}\}$: COVID19

- $\ell(\text{yes, yes}) = 0$ (you did your job well)
- $\ell(\text{no, no}) = 0$ (you did your job well)
- $\ell(\text{yes, no}) = 50$ (the patient goes home and might infect many others)
- $\ell(\text{no, yes}) = 1$ (the patient has to take an unpleasant unnecessary test)

Common: one outcome is rare, but has bad consequences if mispredicted
Instead of minimizing the error probability, minimize the *expected loss*!

**Definition**

The classifier of minimal expected $\ell$-risk is given by

$$c_\ell^*(x) := \underset{y \in \mathcal{Y}}{\text{argmin}} E_{\tilde{y} \sim p(\tilde{y}|x)} \ell(\tilde{y}, y).$$

**Lemma**

For $\mathcal{Y} = \{-1, +1\}$, and $\ell(y, \tilde{y})$ given by the table

<table>
<thead>
<tr>
<th>$y$</th>
<th>$\tilde{y}$</th>
<th>$-1$</th>
<th>$+1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-1$</td>
<td>$a$</td>
<td>$b$</td>
<td></td>
</tr>
<tr>
<td>$+1$</td>
<td>$c$</td>
<td>$d$</td>
<td></td>
</tr>
</tbody>
</table>

the risk w.r.t. $\ell$ is minimized by the decision rule

$$c_\ell^*(x) = \text{sign}[ \log \frac{p(x, +1)}{p(x, -1)} + \log \frac{c - d}{b - a} ],$$

or equivalently

$$c_\ell^*(x) = \text{sign}[ \log \frac{p(+1|x)}{p(-1|x)} + \log \frac{c - d}{b - a} ].$$

Proof: Exercise...
Observation
The generalization error is the risk for 0/1-loss, i.e. $\ell(y, y') = \mathbf{1}[y \neq y']$.

Question: What’s the best classifier for a fully known problem?

Question answered. We have identified the optimal classifiers!
In the real world, $p(x, y)$ is unknown, but we have a training set $\mathcal{D}$. What to do?

**Definition**

Given a training set $\mathcal{D}$, we call it

- a **generative probabilistic approach**: if we use $\mathcal{D}$ to build a model $\hat{p}(x, y)$ of $p(x, y)$, and then define

  $$c(x) := \operatorname{argmax}_{y \in \mathcal{Y}} \hat{p}(x, y) \quad \text{or} \quad c_\ell(x) := \operatorname{argmin}_{y \in \mathcal{Y}} \mathbb{E}_{\bar{y} \sim \hat{p}(x, \bar{y})} \ell(\bar{y}, y).$$

- a **discriminative probabilistic approach**: if we use $\mathcal{D}$ to build a model $\hat{p}(y|x)$ of $p(y|x)$ and define

  $$c(x) := \operatorname{argmax}_{y \in \mathcal{Y}} \hat{p}(y|x) \quad \text{or} \quad c_\ell(x) := \operatorname{argmin}_{y \in \mathcal{Y}} \mathbb{E}_{\bar{y} \sim \hat{p}(y|x)} \ell(\bar{y}, y).$$

- a **decision theoretic approach**: if we use $\mathcal{D}$ to directly search for a classifier $c$. 
Generative Probabilistic Models

Setting

We are given

- a **training set** of examples $\mathcal{D} = \{(x^1, y^1), \ldots, (x^n, y^n)\}$,
  (note: technically rather a multi-set, elements can occur more than once)

Assumption:

- $\mathcal{D}$ are *independent and identically distributed* (i.i.d.) samples from the unknown probability distribution $p(x, y)$.

Shorthand notation,

- $\mathcal{D}^X := \{x^1, \ldots, x^n\}$, input part of $\mathcal{D}$,
- $\mathcal{D}^Y := \{y^1, \ldots, y^n\}$, output part of $\mathcal{D}$,
- $\mathcal{D}_y := \{(x^i, y^i) \in \mathcal{D} : y^i = y\}$, all examples of label $y$. 
Let’s use $\mathcal{D}$ to form an estimate of $p(x, y)$.

**Definition**

There’s (at least) three approaches:

- **parametric estimate:**
  - fix a model class $\hat{p}(x, y; \theta)$,
  - estimate parameters $\hat{\theta}$ such that $\hat{p}(x, y; \hat{\theta}) \approx p(x, y)$.
  - the size of $\theta$ is independent of how large $\mathcal{D}$ is

- **non-parametric estimate:**
  - estimate any $\hat{p}(x, y) \approx p(x, y)$
  - the number of parameters/complexity of $\hat{p}(x, y)$ can grow with $|\mathcal{D}|$

- hybrids of the two
Generative Probabilistic Models: Multinomial

If $\mathcal{X}$ and $\mathcal{Y}$ are finite, we can represent any $p(x, y)$ as a table of values.

To simplify notation, we look at generic $z \in \mathcal{Z}$ (think: $z = (x, y)$):

**Definition (Empirical estimate)**

Let $z^1, \ldots, z^n$ be samples from $p(z)$, then we call

$$\hat{p}_n(z) := \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}[z^i = z]$$

the empirical estimate of $p(z)$ from $n$ samples.

Example: flipping a coin, $\mathcal{Z} = \{\text{heads, tails}\}$

- observed outcomes ($n = 6$): heads, heads, heads, tails, heads, tails
- $\hat{p}_6(\text{heads}) = \frac{1}{6} (1 + 1 + 1 + 0 + 1 + 0) = \frac{2}{3}$
- $\hat{p}_6(\text{tails}) = \frac{1}{6} (0 + 0 + 0 + 1 + 0 + 1) = \frac{1}{3}$
Theorem (Convergence of the empirical estimate)

Let $z^1, z^2, \ldots$ be i.i.d. samples from $p(z)$. For every possible value $z \in \mathcal{Z}$

$$\Pr \left\{ \lim_{n \to \infty} \hat{p}_n(z) = p(z) \right\} = 1.$$ 

Proof.

Every textbook on statistics: *law of large numbers* (strong version).
The curse of dimensionality

Setting:
Let $\mathcal{Z} = \mathcal{Z}_1 \times \cdots \times \mathcal{Z}_d$, i.e. data decomposes into $d$ non-trivial "features", "attributes", or "dimensions". Let $m_j := |\mathcal{Z}_j| \geq 2$ for $j = 1, \ldots, d$.

Lemma

The number of samples needed to estimate $\hat{p}(z)$ grows exponentially in $d$ (unless we made additional assumptions).

Proof.

$\hat{p}(z)$ has $|\mathcal{Z}| = \prod_{j=1}^{d} m_j \geq 2^d$ entries. Without further assumptions, each entry can be set arbitrarily, independently, except for the one constraint that they must sum to 1. Each sample influences only one bin, so we need at least $2^d - 1$ samples (in practice, many times that, of course).
## Example (Dating agency table)

<table>
<thead>
<tr>
<th>TRAINING</th>
<th>eyes</th>
<th>height</th>
<th>handsome</th>
<th>sex</th>
<th>soccer</th>
<th>date?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apu</td>
<td>blue</td>
<td>tall</td>
<td>yes</td>
<td>male</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Bernice</td>
<td>brown</td>
<td>short</td>
<td>yes</td>
<td>female</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Itchy</td>
<td>brown</td>
<td>short</td>
<td>no</td>
<td>male</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

Can we estimate $p(x, y)$ here?

- $|\mathcal{X} \times \mathcal{Y}| = (3 \times 2 \times 2 \times 2 \times 2) \times 2 = 96$, $\rightarrow p(x, y)$ has 95 free parameters
- We have 9 samples.
- **Most possible combinations we have never seen!**
Example (Dating agency table)

<table>
<thead>
<tr>
<th>TRAINING</th>
<th>eyes</th>
<th>height</th>
<th>handsome</th>
<th>sex</th>
<th>soccer</th>
<th>date?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apu</td>
<td>blue</td>
<td>tall</td>
<td>yes</td>
<td>male</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Bernice</td>
<td>brown</td>
<td>short</td>
<td>yes</td>
<td>female</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Itchy</td>
<td>brown</td>
<td>short</td>
<td>no</td>
<td>male</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

Can we estimate $p(x, y)$ here?

- $|\mathcal{X} \times \mathcal{Y}| = (3 \times 2 \times 2 \times 2 \times 2) \times 2 = 96, \quad \rightarrow \quad p(x, y) \text{ has } 95 \text{ free parameters}$
- We have 9 samples.
- **Most possible combinations we have never seen!**

**Bayes classifier from** $\hat{p}(x, y): \quad c(x) := \arg \max_{y \in \mathcal{Y}} \hat{p}(x, y)$

- $\hat{p}(\text{Apu}, \text{yes}) = \frac{1}{9}, \quad \hat{p}(\text{Apu}, \text{no}) = 0, \quad \rightarrow \quad c(\text{Apu}) = \text{yes},$
- $\hat{p}(\text{Jimbo}, \text{yes}) = 0, \quad \hat{p}(\text{Jimbo}, \text{no}) = 0, \quad \rightarrow \quad c(\text{Jimbo}) = ???,$

No clue about previously unseen patterns $\rightarrow$ very little generalization ability
Naive Bayes Model

**Definition**

Let $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_d$. The *Naive Bayes (NB)* estimate of $p(x, y)$ is

$$
\hat{p}_{NB}(x, y) := \hat{p}(y) \prod_{j=1}^{d} \hat{p}_j(x_j|y),
$$

where

- $\hat{p}(y)$ is an estimate of $p(y)$,
- $\hat{p}_j(x_j|y)$ are estimates of $p(x_j|y)$ for every $j = 1, \ldots, d$. 

Lemma

The number of free parameters in $p_{NB}(x, y)$ grows linearly with $d$ (instead of exponentially).

Proof.

$p_{NB}(x, y)$ has $|Y| \left[ 1 + \sum_{j=1}^{d} (m_j - 1) \right] - 1$ degrees of freedom.

32 / 36
Naive Bayes Model

Definition
Let \( \mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_d \). The Naive Bayes (NB) estimate of \( p(x, y) \) is

\[
\hat{p}_{\text{NB}}(x, y) := \hat{p}(y) \prod_{j=1}^{d} \hat{p}_j(x_j|y),
\]

where
- \( \hat{p}(y) \) is an estimate of \( p(y) \),
- \( \hat{p}_j(x_j|y) \) are estimates of \( p(x_j|y) \) for every \( j = 1, \ldots, d \).

Lemma

The number of free parameters in \( p_{\text{NB}}(x, y) \) grows \textit{linearly} with \( d \) (instead of exponentially).

Proof.

\( p_{\text{NB}}(x, y) \) has \(|\mathcal{Y}|[1 + \sum_{j=1}^{d}(m_j - 1)] - 1 \) degrees of freedom.
Definition

The Naive Bayes classifier is given by

\[ c(x) := \underset{y \in Y}{\text{argmax}} \hat{p}_{NB}(x, y) \]

A Naive Bayes classifier needs much fewer examples for 'training' than one based on a full probability table.

Remark

Even for \( n \to \infty \), we likely won't have \( \hat{p}_{NB}(x, y) \to p(x, y) \)!

So, most likely, the NB model is wrong as a density estimate.

But that doesn't mean it doesn't work for making decisions!

In fact, NB is very successful, e.g. in Spam filtering (text classification).

"All models are wrong, but some are useful." (George E. P. Box, 1979)
Naive Bayes Classifier

**Definition**

The *Naive Bayes* classifier is given by

\[
c(x) := \arg\max_{y \in Y} \hat{p}_{NB}(x, y)
\]

A Naive Bayes classifier needs much fewer examples for 'training' than one based on a full probability table.

**Remark**

Even for \( n \to \infty \), we likely won’t have \( \hat{p}_{NB}(x, y) \not\to p(x, y) \! \)

So, most likely, the NB model is **wrong** as a density estimate.

But that doesn’t mean it doesn’t work for making decisions!

In fact, NB is *very successful*, e.g. in Spam filtering (text classification).
**Definition**

The *Naive Bayes* classifier is given by

\[ c(x) := \arg \max_{y \in Y} \hat{p}_{NB}(x, y) \]

A Naive Bayes classifier needs much fewer examples for ‘training’ than one based on a full probability table.

**Remark**

Even for \( n \to \infty \), we likely won’t have \( \hat{p}_{NB}(x, y) \to p(x, y) \)!

So, most likely, the NB model is wrong as a density estimate.

But that doesn’t mean it doesn’t work for making decisions!

In fact, NB is very successful, e.g. in Spam filtering (text classification).

"All models are wrong, but some are useful." (George E. P. Box, 1979)
Both models we saw so far are parametric:

For finite $z \in \mathcal{Z}$, $p(z)$ is multinomial distribution:
- $|\mathcal{Z}|$ parameters: $\theta_z$ for $z \in \mathcal{Z}$ with $p(Z = z) = \theta_z$
- parameters fulfill
  - $\theta_z \geq 0$
  - $\sum_z \theta_z = 1$

Similar for Naive Bayes model:
- $\hat{p}(y)$ is multinomial for $y \in \mathcal{Y}$, parameter $\theta_y \in \mathbb{R}^{|\mathcal{Y}|}$, with $\hat{p}(y) = \theta_y$ and $\sum_y \theta_y = 1$,
- $\hat{p}(x_j|y)$ is multinomial for $x_j \in \mathcal{X}_j$, parameters $\theta^j_{x_j}$
  - $\hat{p}(x_j|y) = \theta^y_{x_j}$ with $\theta^y_{x_j} \geq 0$, $\sum_{x_j \in \mathcal{X}_j} \theta^y_{x_j} = 1$, for all $y \in \mathcal{Y}$

We set parameters as $\theta_z = \frac{1}{n} \sum_{i=1}^{n} \left[ z^i = z \right]$? Why?
Let $\hat{p}(z; \theta)$ be a parametric model with parameter $\theta \in \Theta$. 
Let $D = \{z^1, \ldots, z^n\}$ be i.i.d. samples from $p(z)$.

**Definition (Parameter estimation)**

There’s (at least) two main approaches to set $\theta$:

**Maximum Likelihood (ML) Estimation:**
Which parameter value makes it most likely that we observed $D$?

$$
\theta_{ML} = \operatorname{argmax}_{\theta \in \Theta} \hat{p}(z^1, \ldots, z^n; \theta) = \operatorname{argmax}_{\theta \in \Theta} \prod_i \hat{p}(z^i; \theta)
$$

**Maximum-A-Posteriori (MAP) Parameter Estimation:**
Treat $\theta$ as a random variable itself. What’s its most likely value given $D$?

$$
\theta_{MAP} = \operatorname{argmax}_{\theta \in \Theta} p(\theta \mid z^1, \ldots, z^n)
$$

$$
= \operatorname{argmax}_{\theta \in \Theta} p(\theta)p(z^1, \ldots, z^n \mid \theta) = \operatorname{argmax}_{\theta \in \Theta} p(\theta) \prod_i \hat{p}(z^i; \theta)
$$

where $p(\theta)$ is a *prior* distribution over the possible parameter values.
Remark

In practice, one almost always uses the log-likelihood, which gives the same $\theta$ (because $\log$ is a monotonous function):

$$\theta_{\text{ML}} = \arg\max_{\theta \in \Theta} \log \prod_{i=1}^{n} \hat{p}(x^i; \theta) = \arg\max_{\theta \in \Theta} \sum_{i=1}^{n} \log \hat{p}(x^i; \theta)$$

and

$$\theta_{\text{MAP}} = \arg\max_{\theta \in \Theta} \log \left[ \hat{p}(\theta) \prod_{i} \hat{p}(z^i; \theta) \right]$$

$$= \arg\max_{\theta \in \Theta} \log \hat{p}(\theta) + \sum_{i} \log \hat{p}(z^i; \theta)$$

Example in exercises: $z \in \{0, 1\}$, $\hat{p}(z = 1; \theta) = \theta$, $\hat{p}(z = 0; \theta) = 1 - \theta$. 