Boolean Logic
Resolution Method

Stepan Kuznetsov

Discrete Math Bridging Course, HSE University

Course Outline

- The aim of this course is to provide a background of discrete mathematics and computational complexity ideas useful for data science.

Course Outline

- The aim of this course is to provide a background of discrete mathematics and computational complexity ideas useful for data science.
- Given a very limited time for the course, we have to choose a simple central topic to use as a running example.

Course Outline

- The aim of this course is to provide a background of discrete mathematics and computational complexity ideas useful for data science.
- Given a very limited time for the course, we have to choose a simple central topic to use as a running example.
- And this topic is going to be **Boolean logic.**

Course Outline

- The aim of this course is to provide a background of discrete mathematics and computational complexity ideas useful for data science.
- Given a very limited time for the course, we have to choose a simple central topic to use as a running example.
- And this topic is going to be **Boolean logic.**
- Let us first remind the basics of it.

Boolean Functions

- Boolean functions operate on the two-element set $\{0, 1\}$ (the simplest non-trivial set).

Boolean Functions

- Boolean functions operate on the two-element set $\{0, 1\}$ (the simplest non-trivial set).
- Formally, an n-ary Boolean function is a function

$$f\colon \underbrace{\{0,1\} \times \ldots \times \{0,1\}}_{n \text{ times}} \to \{0,1\}.$$

Boolean Functions

- Boolean functions operate on the two-element set $\{0, 1\}$ (the simplest non-trivial set).
- Formally, an n-ary Boolean function is a function

$$f\colon \underbrace{\{0,1\} \times \ldots \times \{0,1\}}_{n \text{ times}} \to \{0,1\}.$$

- A Boolean function is a *finite* object: it can be represented by a table (so-called *truth table*) of 2^n rows.

Boolean Functions

- The total number of n-ary Boolean functions is 2^{2^n}.

Boolean Functions

- The total number of n-ary Boolean functions is 2^{2^n}.
- For example, we have 4 unary Boolean functions and $16 = 2^{2^2}$ binary ones.

Boolean Functions

- The total number of n-ary Boolean functions is 2^{2^n}.
- For example, we have 4 unary Boolean functions and $16 = 2^{2^2}$ binary ones.
- The only interesting unary Boolean function is *negation*, defined by the following truth table:

x	$\neg x$
0	1
1	0

Boolean Functions

- As for binary functions, among 16 possible there are several interesting ones: $\wedge$ (*conjunction*, "and"), $\vee$ (*disjunction*, "or"), $\rightarrow$ (*implication*, "if … then").

Boolean Functions

- As for binary functions, among 16 possible there are several interesting ones: $\wedge$ (*conjunction*, "and"), $\vee$ (*disjunction*, "or"), $\rightarrow$ (*implication*, "if … then").
- The truth tables for them are as follows:

x	y	$x \wedge y$	$x \vee y$	$x \rightarrow y$
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	1

Boolean Functions

- $\neg$, $\wedge$, $\vee$, and $\rightarrow$ form a **complete system** of Boolean functions in the following sense.

Boolean Functions

- $\neg$, $\wedge$, $\vee$, and $\rightarrow$ form a **complete system** of Boolean functions in the following sense.

Theorem

Any Boolean function can be represented as a composition of $\neg$, $\wedge$, $\vee$, $\rightarrow$.

Boolean Functions

- $\neg$, $\wedge$, $\vee$, and $\rightarrow$ form a **complete system** of Boolean functions in the following sense.

Theorem

Any Boolean function can be represented as a composition of $\neg$, $\wedge$, $\vee$, $\rightarrow$.

- For example, the **majority function** of three elements, which gives 1 iff at least two of its arguments are 1, has the following representation:

$$\mathrm{MAJ}_3(x, y, z) = (x \wedge y) \vee (x \wedge z) \vee (y \wedge z).$$

Boolean Formulae

- Such representations are formalized by **Boolean formulae**.

Boolean Formulae

- Such representations are formalized by **Boolean formulae**.
- The set Fm of Boolean formulae over a set of *variables* Var is defined as the minimal set obeying the following:
  - $\mathrm{Var} \subseteq \mathrm{Fm}$
  - $\bot, \top \in \mathrm{Fm}$ (these are *constants* for 0 and 1)
  - if $A, B \in \mathrm{Fm}$, then $(A \wedge B), (A \vee B), (A \to B), \neg A \in \mathrm{Fm}$

Tautologies

- We shall consider Boolean formulae as **logical formulae,** which represent logical truth.

Tautologies

- We shall consider Boolean formulae as **logical formulae,** which represent logical truth.
    - A classic example. If it is raining, then there are clouds in the sky. There are no clouds in the sky. Thus, it is not raining.

Tautologies

- We shall consider Boolean formulae as **logical formulae**, which represent logical truth.
    - A classic example. If it is raining, then there are clouds in the sky. There are no clouds in the sky. Thus, it is not raining.
    - $((r \to c) \land \neg c) \to \neg r$

Tautologies

- We shall consider Boolean formulae as **logical formulae**, which represent logical truth.
    - A classic example. If it is raining, then there are clouds in the sky. There are no clouds in the sky. Thus, it is not raining.
    - $((r \to c) \land \neg c) \to \neg r$
- This formula is true for **any** values of r, c.

Tautologies

- We shall consider Boolean formulae as **logical formulae,** which represent logical truth.
  - A classic example. If it is raining, then there are clouds in the sky. There are no clouds in the sky. Thus, it is not raining.
  - $((r \to c) \wedge \neg c) \to \neg r$
- This formula is true for **any** values of r, c.
- Such formulae are called **tautologies.**

Tautologies

- Checking a formula for being a tautology is an **algorithmically decidable** question.

Tautologies

- Checking a formula for being a tautology is an **algorithmically decidable** question.
- Indeed, the algorithm can just substitute all possible values of 0 and 1 for variables and compute the value of the formula.

Tautologies

- Checking a formula for being a tautology is an **algorithmically decidable** question.
- Indeed, the algorithm can just substitute all possible values of 0 and 1 for variables and compute the value of the formula.
- However, this requires **exponential time** (checking 2^n possible **assignments**).

Tautologies

- Checking a formula for being a tautology is an **algorithmically decidable** question.
- Indeed, the algorithm can just substitute all possible values of 0 and 1 for variables and compute the value of the formula.
- However, this requires **exponential time** (checking 2^n possible **assignments**).
- Is there a faster algorithm?..

Satisfiability

- It will be more convenient for us to consider a **dual** notion of *satisfiable formula.*

Satisfiability

- It will be more convenient for us to consider a **dual** notion of *satisfiable formula.*
- A Boolean formula is satisfiable, if it is true for **at least one** assignment.

Satisfiability

- It will be more convenient for us to consider a **dual** notion of *satisfiable formula.*
- A Boolean formula is satisfiable, if it is true for **at least one** assignment.
- Such an assignment is called a **satisfying assignment.**

Satisfiability

- Satisfiability is indeed dual to being a tautology:

  A is a tautology $\iff$ $\neg A$ is not satisfiable.

Satisfiability

- Satisfiability is indeed dual to being a tautology:

  A is a tautology $\iff \neg A$ is not satisfiable.

- And actually satisfiability is a very general model example of situations where we seek for existence of an object (here: satisfying assignment) with given properties (here: the given formula A).

DNF and CNF

- A **literal** is either a variable (x) or its negation ($\neg x$, written as $\overline{x}$).

DNF and CNF

- A **literal** is either a variable (x) or its negation ($\neg x$, written as $\overline{x}$).
- An **elementary conjunction** is a conjunction of literals, e.g., $x \wedge \overline{y} \wedge z$.

DNF and CNF

- A **literal** is either a variable (x) or its negation ($\neg x$, written as $\overline{x}$).
- An **elementary conjunction** is a conjunction of literals, e.g., $x \wedge \overline{y} \wedge z$.
- A **DNF** (disjunctive normal form) is a disjunction of elementary conjunctions.

DNF and CNF

- A **literal** is either a variable (x) or its negation ($\neg x$, written as $\overline{x}$).
- An **elementary conjunction** is a conjunction of literals, e.g., $x \wedge \overline{y} \wedge z$.
- A **DNF** (disjunctive normal form) is a disjunction of elementary conjunctions.
- Dually, a **CNF** (conjunctive n.f.) is a conjunction of elementary disjunctions, e.g., $(x \vee y) \wedge (y \vee \overline{z}) \wedge (x \vee \overline{z})$.

DNF and CNF

- A **literal** is either a variable (x) or its negation ($\neg x$, written as $\overline{x}$).
- An **elementary conjunction** is a conjunction of literals, e.g., $x \wedge \overline{y} \wedge z$.
- A **DNF** (disjunctive normal form) is a disjunction of elementary conjunctions.
- Dually, a **CNF** (conjunctive n.f.) is a conjunction of elementary disjunctions, e.g., $(x \vee y) \wedge (y \vee \overline{z}) \wedge (x \vee \overline{z})$.
- The elementary dis- / conjunctions are called **clauses.**

Trivial Cases

- The degenerate DNF with zero clauses is constant "false," $\perp$.

Trivial Cases

- The degenerate DNF with zero clauses is constant “false,” $\bot$.
- Dually, the empty CNF is $\top$, “true.”

Trivial Cases

- The degenerate DNF with zero clauses is constant “false,” $\bot$.
- Dually, the empty CNF is $\top$, “true.”
- Indeed, DNF clauses **add possibilities,** while CNF ones **impose constraints.**

Full DNF

- Any Boolean function can be represented by a **full DNF**, in which each clause contains all variables.

Full DNF

- Any Boolean function can be represented by a **full DNF**, in which each clause contains all variables.
- The full DNF can be obtained from the truth table:

Full DNF

- Any Boolean function can be represented by a **full DNF**, in which each clause contains all variables.
- The full DNF can be obtained from the truth table:

x	y	z	A
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Full DNF

- Any Boolean function can be represented by a **full DNF**, in which each clause contains all variables.
- The full DNF can be obtained from the truth table:

x	y	z	A	
0	0	0	1	$(\overline{x} \wedge \overline{y} \wedge \overline{z})$
0	0	1	0	
0	1	0	0	
0	1	1	0	
1	0	0	1	$(x \wedge \overline{y} \wedge \overline{z})$
1	0	1	1	$(x \wedge \overline{y} \wedge z)$
1	1	0	0	
1	1	1	1	$(x \wedge y \wedge z)$

Full DNF

- Any Boolean function can be represented by a **full DNF,** in which each clause contains all variables.
- The full DNF can be obtained from the truth table:

x	y	z	A	
0	0	0	1	$(\overline{x} \wedge \overline{y} \wedge \overline{z})$
0	0	1	0	
0	1	0	0	
0	1	1	0	
1	0	0	1	$(x \wedge \overline{y} \wedge \overline{z})$
1	0	1	1	$(x \wedge \overline{y} \wedge z)$
1	1	0	0	
1	1	1	1	$(x \wedge y \wedge z)$

(clauses combined with $\bigvee$)

Full DNF

- The full DNF presented on the previous slide,

  $$(\overline{x}\wedge\overline{y}\wedge\overline{z})\vee(x\wedge\overline{y}\wedge\overline{z})\vee(x\wedge\overline{y}\wedge z)\vee(x\wedge y\wedge z),$$

  is not the optimal (shortest) one for the given function.

Full DNF

- The full DNF presented on the previous slide,

  $$(\overline{x} \wedge \overline{y} \wedge \overline{z}) \vee (x \wedge \overline{y} \wedge \overline{z}) \vee (x \wedge \overline{y} \wedge z) \vee (x \wedge y \wedge z),$$

  is not the optimal (shortest) one for the given function.
- The following DNFs are equivalent to it and are shorter:

$$(\overline{x} \wedge \overline{y} \wedge \overline{z}) \vee (x \wedge \overline{y}) \vee (x \wedge y \wedge z)$$

$$(\overline{x} \wedge \overline{y} \wedge \overline{z}) \vee (x \wedge \overline{y} \wedge \overline{z}) \vee (x \wedge z)$$

Completeness of $\neg, \wedge, \vee$

- However, the notion of full DNF is sufficient to prove the theorem that $\neg, \wedge, \vee$ form a complete system of Boolean functions.

Completeness of $\neg, \wedge, \vee$

- However, the notion of full DNF is sufficient to prove the theorem that $\neg, \wedge, \vee$ form a complete system of Boolean functions.
- Moreover, by De Morgan laws, $\neg(A \wedge B) \equiv (\neg A) \vee (\neg B)$, thus $A \wedge B \equiv \neg((\neg A) \vee (\neg B))$.

Completeness of $\neg, \wedge, \vee$

- However, the notion of full DNF is sufficient to prove the theorem that $\neg, \wedge, \vee$ form a complete system of Boolean functions.
- Moreover, by De Morgan laws, $\neg(A \wedge B) \equiv (\neg A) \vee (\neg B)$, thus $A \wedge B \equiv \neg((\neg A) \vee (\neg B))$.
- This means that already $\neg, \vee$ and, dually, $\neg, \wedge$ are complete systems.

Completeness of $\neg, \wedge, \vee$

- However, the notion of full DNF is sufficient to prove the theorem that $\neg, \wedge, \vee$ form a complete system of Boolean functions.
- Moreover, by De Morgan laws, $\neg(A \wedge B) \equiv (\neg A) \vee (\neg B)$, thus $A \wedge B \equiv \neg((\neg A) \vee (\neg B))$.
- This means that already $\neg, \vee$ and, dually, $\neg, \wedge$ are complete systems.
- In particular, $A \to B \equiv \neg A \vee B \equiv \neg(A \wedge \neg B)$.

Full CNF

- A DNF can be translated into a CNF by distributivity.

Full CNF

- A DNF can be translated into a CNF by distributivity.
- One can also construct a full CNF from the truth table by **excluding** 0-lines:

Full CNF

- A DNF can be translated into a CNF by distributivity.
- One can also construct a full CNF from the truth table by **excluding** 0-lines:

x	y	z	A
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Full CNF

- A DNF can be translated into a CNF by distributivity.
- One can also construct a full CNF from the truth table by **excluding** 0-lines:

x	y	z	A	
0	0	0	1	
0	0	1	0	$(x \vee y \vee \overline{z})$
0	1	0	0	$(x \vee \overline{y} \vee z)$
0	1	1	0	$(x \vee \overline{y} \vee \overline{z})$
1	0	0	1	
1	0	1	1	
1	1	0	0	$(\overline{x} \vee \overline{y} \vee z)$
1	1	1	1	

Full CNF

- A DNF can be translated into a CNF by distributivity.
- One can also construct a full CNF from the truth table by **excluding** 0-lines:

x	y	z	A	
0	0	0	1	
0	0	1	0	$(x \vee y \vee \overline{z})$
0	1	0	0	$(x \vee \overline{y} \vee z)$
0	1	1	0	$(x \vee \overline{y} \vee \overline{z})$
1	0	0	1	
1	0	1	1	
1	1	0	0	$(\overline{x} \vee \overline{y} \vee z)$
1	1	1	1	

$\Big\} \bigwedge$

Satisfiability for DNF and CNF

- If a formula is given in DNF, checking its satisfiability is trivial.

Satisfiability for DNF and CNF

- If a formula is given in DNF, checking its satisfiability is trivial.
- The algorithm just checks elementary conjunctions until it finds a consistent one, i.e., one not including both x and $\overline{x}$.

Satisfiability for DNF and CNF

- If a formula is given in DNF, checking its satisfiability is trivial.
- The algorithm just checks elementary conjunctions until it finds a consistent one, i.e., one not including both x and $\overline{x}$.
- This clause is satisfiable, and so is the whole DNF.

Satisfiability for DNF and CNF

- If a formula is given in DNF, checking its satisfiability is trivial.
- The algorithm just checks elementary conjunctions until it finds a consistent one, i.e., one not including both x and $\overline{x}$.
- This clause is satisfiable, and so is the whole DNF.
- For CNFs, satisfiability is a non-trivial question.

Satisfiability for DNF and CNF

- If a formula is given in DNF, checking its satisfiability is trivial.
- The algorithm just checks elementary conjunctions until it finds a consistent one, i.e., one not including both x and $\overline{x}$.
- This clause is satisfiable, and so is the whole DNF.
- For CNFs, satisfiability is a non-trivial question.
- Translating from CNF to DNF does not help: this could increase the size exponentially.

Resolution Method

- The fact that a formula is a tautology can be established (in contrast with checking all assignments) by **proving** it in the *classical propositional calculus.*

Resolution Method

- The fact that a formula is a tautology can be established (in contrast with checking all assignments) by **proving** it in the *classical propositional calculus.*
- In this course, we consider a dual situation: **disproving** satisfiability via **resolution method.**

Resolution Method

- The fact that a formula is a tautology can be established (in contrast with checking all assignments) by **proving** it in the *classical propositional calculus.*
- In this course, we consider a dual situation: **disproving** satisfiability via **resolution method.**
- Recall that, by duality, proving that A is a tautology is equivalent to disproving satisfiability of $\neg A$.

Resolution Method

- Resolution method is applied to formulae in CNF, presented as a list of clauses.

Resolution Method

- Resolution method is applied to formulae in CNF, presented as a list of clauses.
- The one and only rule is **resolution,** which generates new clauses from already existing ones:

$$\frac{A \vee p \quad B \vee \overline{p}}{A \vee B}$$

Resolution Method

- Resolution method is applied to formulae in CNF, presented as a list of clauses.
- The one and only rule is **resolution,** which generates new clauses from already existing ones:

$$\frac{A \vee p \quad B \vee \overline{p}}{A \vee B}$$

- **Contradictive clause:** the empty one (obtained by resolution from p and $\overline{p}$).

Resolution Method

Theorem (Soundness and Completeness)

A CNF is not satisfiable if and only if one can obtain the empty clause by applying resolutions, starting from the given CNF.

Resolution Method

Theorem (Soundness and Completeness)

A CNF is not satisfiable if and only if one can obtain the empty clause by applying resolutions, starting from the given CNF.

- The "if" part (soundness) is easy: if an assignment satisfies $A \vee p$ and $B \vee \overline{p}$, it also satisfies $A \vee B$. The empty clause is not satisfiable.

Resolution Method

Theorem (Soundness and Completeness)

A CNF is not satisfiable if and only if one can obtain the empty clause by applying resolutions, starting from the given CNF.

- The "if" part (soundness) is easy: if an assignment satisfies $A \vee p$ and $B \vee \overline{p}$, it also satisfies $A \vee B$. The empty clause is not satisfiable.
- The "only if" part (completeness) will be proved next time.

Saturation

- The soundness and completeness theorem validates the following **algorithm** for checking satisfiability of CNFs.

Saturation

- The soundness and completeness theorem validates the following **algorithm** for checking satisfiability of CNFs.
- Given a CNF (as a set of clause), let us **saturate** it by exhaustively applying resolutions until they stop generating new clauses.

Saturation

- The soundness and completeness theorem validates the following **algorithm** for checking satisfiability of CNFs.
- Given a CNF (as a set of clause), let us **saturate** it by exhaustively applying resolutions until they stop generating new clauses.
- The CNF is satisfiable if and only if its saturation does not include the empty clause.

Translating into CNF

- The resolution method works only with CNFs.

Translating into CNF

- The resolution method works only with CNFs.
- When checking a formula A for being a tautology, it is convenient for A to be in DNF, since then $\neg A$ is easily transformed into CNF by De Morgan.

Translating into CNF

- The resolution method works only with CNFs.
- When checking a formula A for being a tautology, it is convenient for A to be in DNF, since then $\neg A$ is easily transformed into CNF by De Morgan.
- For implications, keep in mind the following equivalences:

$$A \to B \equiv \neg A \vee B \qquad\qquad \neg(A \to B) \equiv A \wedge \neg B$$

Example

- Let us check whether the following formula is a tautology:

$$A = (p \to (q \to r)) \to ((p \to q) \to (p \to r))$$

Example

- Let us check whether the following formula is a tautology:

$$A = (p \to (q \to r)) \to ((p \to q) \to (p \to r)$$

- Let us negate A and check whether $\neg A$ is satisfiable

$$\neg A = (\overline{p} \vee \overline{q} \vee r) \wedge (\overline{p} \vee q) \wedge p \wedge \overline{r}$$

Example

$\overline{p} \vee \overline{q} \vee r$

$\overline{p} \vee q$

p

$\overline{r}$

Example

$\overline{p} \vee \overline{q} \vee r$ $\quad\quad$ $\overline{q} \vee r$

$\overline{p} \vee q$

p

$\overline{r}$

Example

$\overline{p} \vee \overline{q} \vee r$

$\overline{p} \vee q$

p

$\overline{r}$

$\overline{q} \vee r$

$\overline{p} \vee r$

Example

$\overline{p} \vee \overline{q} \vee r$

$\overline{p} \vee q$

p

$\overline{r}$

$\overline{q} \vee r$

$\overline{p} \vee r$

r

Example

$\overline{p} \vee \overline{q} \vee r$	$\overline{q} \vee r$
$\overline{p} \vee q$	$\overline{p} \vee r$
p	r
$\overline{r}$	$\bot$

Example

$\overline{p} \vee \overline{q} \vee r$	$\overline{q} \vee r$	
$\overline{p} \vee q$	$\overline{p} \vee r$	
p	r	
$\overline{r}$	$\bot$	$\Rightarrow$ NOT SATISFIABLE

2-CNF

- Unfortunately, in the general case saturation can be exponential.

2-CNF

- Unfortunately, in the general case saturation can be exponential.
- However, if each clause has no more than 2 literals (this is called a `2-CNF`), resolution method works really fast.

2-CNF

- Unfortunately, in the general case saturation can be exponential.
- However, if each clause has no more than 2 literals (this is called a 2-CNF), resolution method works really fast.
- Indeed, applying resolution to 2-bounded clauses also yields a 2-bounded clause.

2-CNF

- Unfortunately, in the general case saturation can be exponential.
- However, if each clause has no more than 2 literals (this is called a 2-CNF), resolution method works really fast.
- Indeed, applying resolution to 2-bounded clauses also yields a 2-bounded clause.
- And the total number of 2-bounded clauses is $\leq 4n^2 + 2n + 1$.

2-CNF

- Unfortunately, in the general case saturation can be exponential.
- However, if each clause has no more than 2 literals (this is called a **2-CNF**), resolution method works really fast.
- Indeed, applying resolution to 2-bounded clauses also yields a 2-bounded clause.
- And the total number of 2-bounded clauses is $\leq 4n^2 + 2n + 1$.
- Thus, checking satisfiability for 2-CNF **can be performed in polynomial time.**

Polynomiality

- Traditionally, an algorithmic problem is considered “practically solvable,” if there exists a polynomially bounded algorithm for it (that is, the number of steps, even in the worst case, is $\leq p(|x|)$, where p is a fixed polynomial and $|x|$ is the input length).

Polynomiality

- Traditionally, an algorithmic problem is considered “practically solvable,” if there exists a polynomially bounded algorithm for it (that is, the number of steps, even in the worst case, is $\leq p(|x|)$, where p is a fixed polynomial and $|x|$ is the input length).
- This is, of course, a gross approximation: let, say, $p(n) = n^{100}$.

Polynomiality

- In real practice people usually wish better complexity bounds, e.g., $n \log n$.

Polynomiality

- In real practice people usually wish better complexity bounds, e.g., $n \log n$.
- However, polynomiality is **robust:** it is independent from details of implementation and even from the computational model.

Polynomiality

- In real practice people usually wish better complexity bounds, e.g., $n \log n$.
- However, polynomiality is **robust:** it is independent from details of implementation and even from the computational model.
    - A problem is polynomially solvable on a “real” computer iff it is polynomially solvable on a 1-tape Turing machine.

Polynomiality

- In real practice people usually wish better complexity bounds, e.g., $n \log n$.
- However, polynomiality is **robust:** it is independent from details of implementation and even from the computational model.
    - A problem is polynomially solvable on a “real” computer iff it is polynomially solvable on a 1-tape Turing machine.
    - … but with a different degree of p.

Polynomiality

- As we've seen, satisfiability for DNF and for 2-CNF is polynomially decidable.

Polynomiality

- As we’ve seen, satisfiability for DNF and for 2-CNF is polynomially decidable.
    - In short, these problems belong to class P.

Polynomiality

- As we've seen, satisfiability for DNF and for 2-CNF is polynomially decidable.
  - In short, these problems belong to class P.
- For satisfiability of CNFs, the situation is different.

Polynomiality

- As we've seen, satisfiability for DNF and for 2-CNF is polynomially decidable.
    - In short, these problems belong to class P.
- For satisfiability of CNFs, the situation is different.
    - By now, it is unknown whether it is in P.

Polynomiality

- As we’ve seen, satisfiability for DNF and for 2-CNF is polynomially decidable.
  - In short, these problems belong to class P.
- For satisfiability of CNFs, the situation is different.
  - By now, it is unknown whether it is in P.
  - However, this is highly unlikely, because then a large class of similar problems, called NP, would be also in P.

Polynomiality

- As we've seen, satisfiability for DNF and for 2-CNF is polynomially decidable.
    - In short, these problems belong to class P.
- For satisfiability of CNFs, the situation is different.
    - By now, it is unknown whether it is in P.
    - However, this is highly unlikely, because then a large class of similar problems, called NP, would be also in P.
    - These problems include, e.g., subgraph isomorphism, knapsack problem, subset sum problem, ...

What Next?

- In the course, we shall develop the theory of NP problems and NP-completeness, and related topics.

What Next?

- In the course, we shall develop the theory of NP problems and NP-completeness, and related topics.
- The running examples will be connected to Boolean logic and graph theory.

What Next?

- In the course, we shall develop the theory of NP problems and NP-completeness, and related topics.
- The running examples will be connected to Boolean logic and graph theory.
- During the course, we'll highlight possible connections and applications in data analysis.