.

České vysoké učení technické v Praze
fakulta Elektrotechnická

Czech Technical University in Prague
faculty of Electrical Engineering

doc. Ing. Filip Železný, Ph.D.

Relační strojové učení a jeho bioinformatické aplikace

Relational machine learning with applications in bioinformatics

Summary

This script accompanies a short lecture I am presenting within the qualification process for full professorship at the Czech Technical University in Prague, Faculty of Electrical Engineering. I am taking this opportunity as a challenge to explain the concepts, ideas, and goals of my research field to a general intelligent audience with elementary background in mathematics and technology but no special background in the particular disciplines underlying my work. These disciplines are computational logic, machine learning and bioinformatics.

The lecture is structured as a story of a quest to design an algorithm which could solve a particular problem in bioinformatics. The narrative aligns nicely with the conventional Greek drama pattern [1]. In Chapter 1, the main actors (logic, machine learning and bioinformatics) are exposed briefly through examples that illustrate their driving ideas. The actors collide in Chapter 2, where I show how an important bioinformatics problem in the area of structural proteomics calls naturally for a machine learning algorithm that encodes data (protein structures) and working hypotheses in the language of first-order predicate logic. Such a system learns and reasons on the basis of attributes of the objects in question but also on the basis of their mutual relationships, hence the term relational machine learning. Chapter 3 addresses a crisis resting in the observation that such a synergic system, if built with state-of-the-art ingredients, just won't scale up to the complexity of the problem. The two main culprits for this bad news are identified. In Chapter 4, I review a few peripeteias in the efforts to combat these two sources of complexity and I focus on those I took part in. The last Chapter 5 tells how the experience acquired lead to clarifying the design principles of the method and, finally, cracking the original challenge.

Souhrn

Tento materiál doprovází mou krátkou přednášku v rámci profesorského jmenovacího řízení na fakultě elektrotechnické Českého vysokého učení technického v Praze. Této příležitosti využívám jako testu, zda jsem schopen vysvětlit pojmy, myšlenky a cíle mého oboru inteligentním posluchačům se základními znalostmi matematiky a techniky, ale bez zvláštních znalostí speciálních disciplín, na nichž je má práce založena. Těmito disciplínami jsou výpočetní logika, strojové učení a bioinformatika.

Přednáška je podána jako příběh o snaze vyvinout algoritmus, který by uměl vyřešit konkrétní bioinformatický problém. Vyprávění se přirozeně člení do klasické osnovy řeckého dramatu [1]. V kapitole 1 se krátce představí hlavní postavy (logika, strojové učení a bioinformatika), a to prostřednictvím příkladů ilustrujících jejich nosné myšlenky. Ke srážce postav dojde v kapitole 2, kde ukazuji, jak jeden z důležitých bioinformatických problémů z oblasti strukturní proteomiky přirozeně volá po algoritmu strojového učení, který by byl schopen rozumět datům (strukturám bílkovin) popsaným jazykem predikátové logiky prvního řádu a vytvářet hypotézy ve stejném jazyce. Takový systém se učí a usuzuje na základě vlastností objektů, ale také na základě vztahů (relací) mezi těmito objekty. Odtud plyne termín relační strojové učení. V kapitole 3 dojde ke krizi, neboť se ukáže, že takový synergický systém, pokud je sestaven z běžných (state of the art) součástí, nebude schopen řešit úlohy požadovaných rozměrů. Kapitola objasňuje, jací jsou hlavní dva viníci tohoto problému. V kapitole 4 zmíním několik peripetií úsilí o překonání těchto příčin nezdaru, a zejména se zaměřuji na směry, kterých jsem se účastnil. Závěrečná kapitola 5 vysvětluje, jak zkušenosti z těchto peripetií vedly k objasnění principů, podle nichž je třeba navrhnout vytouženou metodu, a konečně i ke zdolání původního problému.

Klíčová slova

Umělá inteligence, relační strojové učení, induktivní logické programování, analýza dat, prediktivní klasifikace, bioinformatika, molekulární genomika, strukturní proteomika, genová exprese, transkripční faktory

Keywords

Artificial intelligence, relational machine learning, inductive logic programming, data analysis, predictive classification, bioinformatics, molecular genomics, structural proteomics, gene expression, transcription factors

Contents

1 Exposition *(Logic, Learning, and Bioinformatics)*

The three leads of the story are mathematical logic, machine learning, and bioinformatics.

Mathematical logic studies the way humans think rationally. To do so, it first sets grammatical rules by which 'reasonable' statements can be formed. A reasonable statement is one we can interpret; so *lorem ipsum* is not reasonable but *all men are mortal* is, whether or not it is true. One of the famous kinds of logic is called *first-order predicate logic*, which would encode the latter assertion as

$$\forall x\ \mathsf{man}(x) \to \mathsf{mortal}(x) \tag{1}$$

The meaning of the involved symbols is likely obvious. The arrow denotes implication and is one of several well-known symbols called *connectives* further including the conjunction $\wedge$, disjunction $\vee$ and negation $\neg$. The man and mortal are *predicates* which stipulate a property of the object in the parentheses. The object is represented by *variable* x which is *universally quantified* by $\forall x$; this means that the implication holds for every x. So for *some men are mortal* we would have and *existential quantifier* $\exists x$ instead. The formula above is of course more cryptic than *all men are mortal* but much easier to work with for a computer.

For formulas written in the correct syntax, we are naturally interested whether they hold true in the real world. Of course, mathematical logic cannot just tell whether a statement is true in the world unless we describe the world to it. We have to do this again in a precise formal language. For obvious reasons, we need to focus only on a particular finite domain, that is, a selected part of the world which we consider relevant. Probably the simplest way to describe a domain is to list all the facts that hold in it:

$$\begin{aligned}\{&\mathsf{man}(\mathsf{Sokrates}), \mathsf{man}(\mathsf{Aristotle}), \mathsf{activity}(\mathsf{philosophy}),\\ &\mathsf{mortal}(\mathsf{Sokrates}), \mathsf{mortal}(\mathsf{Aristotle})\}\end{aligned} \tag{2}$$

This simple way of describing the domain by a set of all true facts written again in the predicate syntax we already know, is called a *Herbrand interpretation* after the French mathematician Jacques Herbrand. Note that any fact belonging to the domain but not listed in the interpretation is considered false, so e.g. $\mathsf{mortal}(\mathsf{philosophy})$ or $\mathsf{activity}(\mathsf{Sokrates})$ are false.

A juicy part of mathematical logic comes into play when evaluating the truth of formulas such as (1) in light of an interpretation. A simple algorithm can check that (1) is true in the above interpretation just as a human can do by commonsense. Likewise easily, we see that a statement more general than (1)

$$\forall x\ \mathsf{mortal}(x) \tag{3}$$

i.e., *all is mortal* is false here due to the observed immortality of philosophy, although the statement could be true in a different interpretation. Any interpretation which makes a formula true is called a *model* of that formula.

It does not take much to imagine way more complex formulas where commonsense would fall short in judging their truthfulness, and that is just where the computational power of the computer becomes useful. But an even bigger challenge for logic arises when it comes to mutual relationships between different formulas. We see immediately that Formula 3 is stronger or *more general* than Formula 1. It does not matter that the former is in fact false in our current interpretation, the important thing is that any intepretation that makes Formula 3 true, i.e. is a model of it, must also be a model of Formula 1.

We capture this relationship graphically in Fig. 1. Sometimes the relation is phrased so that the top formula *entails* or *implies* the bottom one, although this kind of implication should not be confused with the one denoted by $\rightarrow$ and appearing inside formulas. Conversely, the bottom formula is said to be a *consequence* of the top one.

How would a computer go about verifying such a logical entailment? One option suggesting itself is to simply go by the definition, that is, look at every possible interpretation that is a model of the top formula to see if it is also a model of the bottom one. When saying 'every interpretation' we assume implicitly that there is only a finite number of them, otherwise the checking would never end. If we simply fix a finite set of objects symbols such as Aristotle and the same for predicates such as man then the number of possible interpretations we can make of them would indeed be finite, though possibly huge. The crucial problem is, however, that the first-order predicate logic language allows—in addition to what we have seen above—to form new objects out of other objects by using so called *function symbols*. So while we have not included Aristotle's father Nicomachus in our domain, the language can refer to him as fatherof(Aristotle) whenever the vocabulary contains the function symbol fatherof. And what is more, the very inclusion of a single function symbol suddenly renders our domain *infinite* because we can now refer to Aristotle's grandfather as fatherof(fatherof(Aristotle)), his great grandfather and so on! With an infinite domain, we can obviously produce an infinite number of infinite interpretations and so the brute force approach to entailment checking is doomed.

This is where logicians came to rescue with algorithms which can do the job without looking at all possible interpretations. The bottom line of such algorithms is akin to human rational argumentation. We take the premise and modify it very slightly according to some editing pattern that is correct in the sense that whatever it produces is a consequence of the premise. Then we

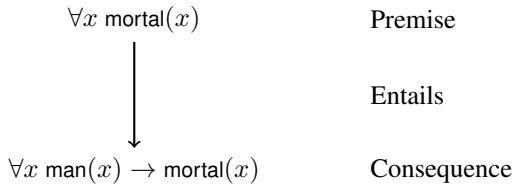

Fig. 1: The premise *entails* the consequence in the sense that in any world where the premise is true, the consequence is also true. That is to say, any model of the premise is also a model of the consequence.

apply a correct editing pattern on the consequence itself and continue doing so until, hopefully, we arrive to the suspected consequence. What are such correct editing patterns? For example, from $\forall x$ man(x) we can always draw the consequence man(sokrates) and this of course applies to any predicate and its object, not just man and sokrates. A more sophisticated pattern is known as *modus ponens*: from $(\forall x$ man$(x) \rightarrow$ mortal$(x)) \wedge$ man(sokrates) one derives mortal(Sokrates), and that again applies generally, not just for Sokrates and the predicates man and mortal. For the particular formulas in Fig. 1, yet another rewriting pattern has to be used and we leave this to the reader's imagination.

These rewriting steps seem quite easy and understandable. The big question is though, which edits should be applied and in which order so that we reach the bottom formula as soon as possible? And when do we know that it cannot be achieved this way, i.e. the entailment does not hold? These questions are tackled by so called *proving* algorithms which have been under perpetual development to date.

The adjective *proving* is apt since the sequence of steps leading to the consequence indeed forms what mathematicians call a *proof*. In fact, all mathematics consists of theorems and their proofs. While the latter are usually written in natural language for the reader's convenience, all math could in principle be encoded as formulas in second-order predicate logic which is just an extension of her first-order sister we are exposing here. It may feel somewhat disturbing that the field of logic thus studies how mathematics is built while itself being a part of mathematics. Probably the only way to deal with that uncomfortable thought is to get used to it.

We now leave logic for a while to visit the second actor, machine learning. This is a subfield of artificial intelligence [5] studying how machines (usually computer programs) can improve their performance by experience. While this is a very broad definition, a significant part of machine-learning research boils down to a more narrow and more clearly defined task of *classification*

No.	sepal length	sepal width	petal length	petal width	class
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
...					
51	7.0	3.2	4.7	1.4	versicolor
52	6.4	3.2	4.5	1.5	versicolor
...					

Tab. 1: Part of the Iris data set. This is a textbook example of data from which a machine-learning algorithm can learn to classify. In this case, a learned classifier would determine the taxonomic classes of flowers (right-most column) from their dimensions (other columns).

learning. The latter is very easily exemplified by the following toy exercise. A machine-learning algorithm receives data such as those shown in Table 1, which describe the sepal and petal dimensions of multiple Iris flowers and also their taxonomic class. From these specific examples which are called *training data*, the algorithm is supposed to discover a general pattern determining the plant's class from its dimensions. Here, the data are well fitted for instance by the following classifier

$$\begin{aligned} &\text{petal length} < 3 \rightarrow \text{class} = \text{setosa} \\ &\text{petal length} \geq 3 \rightarrow \text{class} = \text{versicolor} \end{aligned} \tag{4}$$

Why do we need such a classifier if we can simply retrieve the class straight from the table? There are at least two reasons. First, the classifier is *more general* than the data table. It can be used for example to classify a newly discovered plant as long as we can measure its petal length. Second, the classifier conveys a piece of information which was implicit in the data and oftentimes it is interesting or useful to obtain such information explicitly.

In the Iris example, the learning samples are organized nicely as rows in a table whose columns correspond to *attributes*. This form is referred to as the *attribute-value* representation of data and, while probably the most often in practice, it is certainly not the only one used in machine learning. In many domains, learning examples can be represented more naturally by structures that just do not fit well in a table. We will visit such a domain in the next section.

Similarly, the forms of learned classifiers are also very diverse, depending on the machine-learning algorithm applied. To give just one example of a

possible alternative to the two simple rules in (4), consider the equations

$$\begin{aligned} \text{setosa} =& 29.99 - 9.96 \cdot \text{petallength} - 5.71 \cdot \text{petalwidth} \\ \text{versicolor} =& -6.15 + 1.67 \cdot \text{sepallength} + 0.82 \cdot \text{sepalwidth} \\ & - 0.74 \cdot \text{petallength} - 1.28 \cdot \text{petalwidth} \end{aligned} \tag{5}$$

How is this a classifier? One simply does the math on the right-hand sides, plugging in the dimensions of the classified plant. If the setosa equation yields number larger than the versicolor equation, we predict the setosa class, otherwise we predict versicolor.

One of the big challenges of machine-learning research is to determine which kinds of classifiers are good for which data. Intuitively, the kind of classifier shown in (4) has less 'freedom' than the one in (5). Learning the former one essentially means we just pick a single critical attribute and tune a threshold value for it. In the latter, we have much more room for tuning as any attribute can be involved on the right-hand side with an arbitrary coefficient. If follows naturally that the latter classifier type will have a better chance of fitting the input data. That is to say, one can 'fiddle' with the coefficients until the predictions agree perfectly with the known classes. But is that always a good thing?

It turns out that too much perfection on the training data, if achieved at the price of an overly complex classifier, often results in poor generalization, i.e. low accuracy on data not used for the tuning. While this phenomenon known as *overfitting* has an air of paradox, it does have a convincing statistical explanation in machine-learning theory. Its more intuitive counterpart characterized by the other extreme, i.e. inadequate fit on training data is called (surprise!) *underfitting*. So in a way, the mentioned question *which kind of classifier for which data* can be viewed as the art of balancing between underfitting and overfitting, if we abstract from technical aspects such as that some kinds of classifiers can only be applied on some data types (e.g. numeric).

A little while ago we said 'fiddle with the coefficient until the predictions agree.' This is obviously much easier to say than to do. Indeed, the second big question in machine learning is how to adapt a classifier to the training data. This may mean more than just tuning the numeric parameters in equations such as (5). Many machine-learning algorithms also work out automatically the very structure of classifiers while respecting some prescribed constraints. It is beyond the scope of this lecture to explain how all this is done but it will come at little surprise that learning algorithms usually take the trial-and-error way, probing a vast number of parameter values or candidate structures until one works satisfactorily on the training data. In so doing, a more intelligent learning algorithm will make more informed guesses guided by the errors made so far.

Finally, we jump from machine learning over to our last field of interest, which is bioinformatics. In a broad view, bioinformatics deals with algorithms for processing biological data. In a narrower yet more widely accepted view which we also adhere to, bioinformatics is concerned with biological data only at the *molecular level* such as DNA sequences or protein structures. On the other hand, processing of ECG signals or brain images for instance, would not be considered bioinformatics. Is this division not artificial? What changes so abruptly between data on the molecular and, say, organ level? Interestingly, biology is split in a similar way as physics. In the macro-world, i.e. in the scales accessible to our senses, the laws of physic are deterministic and crisp (arbitrarily accurate) following the Newtonian paradigm. But when it comes to the molecular and finer levels, quantum physics kicks in and things become stochastic. Biology seems to follow exactly the reverse pattern. Structures and processes observed on the organ and organism levels can hardly be described through deterministic rules and measurements are almost never crisp. On the other hand, processes inside the cells tend to follow crisp rules and the structures that take part in these processes can usually be described using a finite discrete alphabet. So the biological micro world bears traits very natural for symbolic computer processing (thus earning the suffix 'informatics') and indeed differs from macro-biological data processing.

Let us visit this cellular micro world to see what these main structures and processes are, with a special focus on the genomic aspects. It is well known that the hereditary information prescribing the construction of an organism is stored in a *deoxyribonucleic acid* (DNA). Eucaryotes (organisms with cells more complex than bacteria) store one copy of the same DNA content in the nucleus of each of its cells. From the information-theoretic viewpoint, the DNA is a sequence of symbols drawn from a 4-symbol alphabet. In humans, it is about 3.10^9 symbols long. The symbols are called *bases* and physically they are represented by the respective molecules guanin, cytosin, thymin and uracil. Most of the time, the DNA in fact consists of two parallel sequences (*strands*, follow the left panel in Fig. 2) of the said length, which are however *complementary* in that a symbol at a position of one strand uniquely determines the symbol at the same position of the other strand. This parallelism serves DNA-replication purposes in processes such as cell multiplication.

One of the most important areas where computer science helped biologists is *sequencing*, that is the reading of the content of polymers such as the DNA. A prime example of sequencing was the Human Genome project concluded in 2003, which lasted 10 years and determined the string of bases forming the entire DNA of a single human. No laboratory equipment exists able to read such a long polymer in one shot. Instead, the experimenters have to cut multiple copies of the same DNA into tiny fragments (10s to 100s of bases)

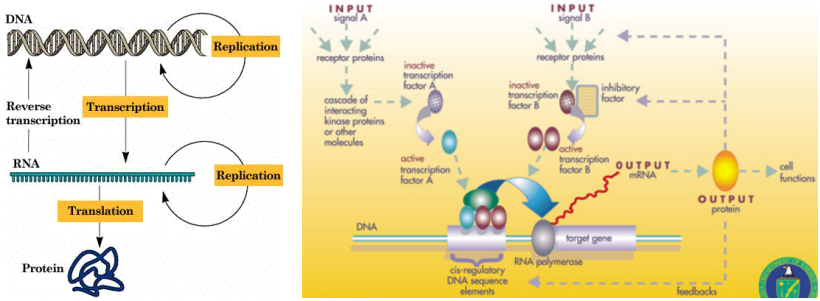

Fig. 2: Left: The flow of information from DNA to proteins often referred to as the *central dogma of molecular biology*. In this talk we are not interested in the processes of reverse transcription or RNA replication. **Right:** An simple scheme of a gene expression regulatory network. (From Wikimedia Commons)

which can be read. As a result they are left with an immense number of (mutually overlapping) substrings. And now the job is handed over to the computer scientist who is asked to reconstruct the most likely original string from the fragments. This problem is probably the oldest and also the founding problem of bioinformatics.

The DNA contains distinguishable regions known as *genes* (about 20 thousand genes in human DNA). A gene contains the prescription for building a protein, a complex molecule made of so called *amino acids*, that performs important tasks in the cell. The building process is called *gene expression*. A gene is first transcribed into a *ribonucleic acid* (RNA), which also is a 4-symbol alphabet just as the DNA and serves for passing the information content from the cellular nucleus to the *cytoplasm*, which is the area bounded by the cellular membrane, outside the nucleus and other organelles. Here, the RNA is finally *translated* into the protein using rather simple translation rules we will return to shortly.

Nowadays, a technology called *DNA microarrays* is available that can measure the rate of expression of very many genes, even the entire genome, at the same time. It is tempting to use these microarrays to measure the genome-wide expression in, say, a few samples of a cancerous tissue and the same for a heathly one. Genes that are expressed only in the cancerous ones would then be the prime suspects as possible causes of the cancer. Moreover, one can create a table such as Tab. 1, in which the flower dimensions would be replaced by the expression rates of genes (corresponding to columns) and the class attribute would carry the status of the tissue. Then a machine-learning algorithm can be used to learn a classifier distinguising cancerous tissues according to the gene expressions. And indeed, the computational analysis of

gene expression data is one of the most popular challenges of current bioinformatics.

We know already that gene expression means the construction of a protein according the gene's content. But how can we describe a protein from the bioinformatics viewpoint? We consider two perspectives. In a *primary structure* perspective, a protein is also a sequence of symbols, just like DNA, expect these are drawn from a different alphabet that has about 20 symbols which correspond to various amino acids. From an expressed gene, a protein primary structure is formed by following the gene's sequence; each three consecutive DNA bases (commonly called a *codon*) determine which residue to attach to the protein under construction. Since $3^4 = 81 > 20$, multiple codons may map to a single amino acid. Codons mapping to the same amino acid usually have similar base sequence and this contributes to resistance against translation errors.

From a *higher-order structure* perspective (secondary, ternary, and quaternary structures are distinguished in biology but here we treat them collectively) a protein folds into a spatial form uniquely defined by its primary structure and determining the protein's physiological function; a protein may act as a building block, be an *enzyme* which catalyzes reactions in the cell, and so on. In effect, the genes expressed into proteins in a tissue determine the structure and function of the tissue up to external influences.

We have seen so far that the gene's content determines uniquely the primary sequential structure of corresponding protein, and that in turn determines (up to marginal exceptions) its higher-order spatial structure. So in principle we should be able to predict the entire protein's shape knowing the content of its coding gene. Unfortunately, this is so far possible only in theory as the computations involved are daunting, and the task remains one of the exemplary hard problems of bioinformatics.

The crucial role of the protein's spatial structure for its function and properties is emphasized by the hyperbole *biology is applied geometry.* The role of the geometry is visible nicely in Fig. 3 which shows two small molecules (so called ligands) that happen to 'dock' into pockets on the surface of a protein that are just complementary to their respective shapes. If the pockets were not so compatible in shape, the ligands would bounce. But when docked, they usually change the shape of the protein and in turn change its function within the cell. For example, the shape change may 'activate' the protein in that it becomes able to bind to yet another molecule, for example the DNA. So this ligand-docking can be seen as ON/OFF switching, where the switch is specific in that it controls only those proteins, which have a compatible docking pocket. One significant facet of bioinformatics addresses exactly the docking problem, where the main goal is to predict whether and where the

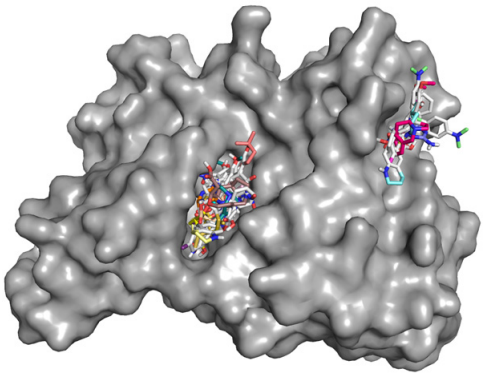

Fig. 3: Left: A computer simulation of the binding ('docking') of two ligands (small molecules shown as wire-models) into a protein (large grey body). The docking is possible only if the ligand's shape matches perfectly one of the protein's surface pockets. (From biochemlabsolutions.com)

docking will occur, given the structure of the protein and the ligand.

Cell functioning depends on which genes get expressed in what situations. The expression of genes is controlled by proteins known as *transcription factors* (TF). As we have learned above, an activated TF is able to physically *bind* to DNA. More specifically, it binds to a region called the *promoter region* (PR) of a gene, located in the vicinity of that gene. This binding is *specific* in that each TF binds to the PR's of certain *target* genes only, although multiple TF's can bind to a single gene's PR. A TF may *catalyze* the expression of a gene by 'dragging-and-dropping' it to the transcription machinery, or *inhibit* it by merely binding to the gene's PR and thus blocking the access of any catalyzing TF. Through TF's, a cell is thus able to react to external stimuli mediated by the ligand molecules which enter the cell through its membrane from the outside and activate the appropriate TF's.

Since TF's are themselves proteins, they are also regulated by other TF's or even by themselves. This gives rise to an extremely complex network of regulatory interactions including omnipresent feedback loops (see the right panel of Fig. 2). Consequently, the concentrations of proteins in a cell can be seen as a state-space vector in a mass-dimensional non-linear dynamic system. Steady states of the cell have been shown to correspond to attractors in this system, and their transitions are a result of external perturbations combined with intrinsic stochastic fluctuations. Modeling and simulation of such networks and predicting their reactions to external perturbations is also at heart of present-day bioinformatics.

2 Collision *(Learning bioinformatics concepts in logic)*

We opened the previous chapter by showing how mathematical logic simulates human rational reasoning. The reasoning consisted of drawing correct consequences from premises, that is, inferring special cases from a general pattern. This kind of reasoning is often called *deductive*. Then we visited machine learning which also mimics human reasoning but quite in the reverse way: given some specific examples (remember the Iris plants and their classes) we inferred a general pattern (a classification rule). As opposed to deductive inference, the specific-to-general reasoning employed in learning is termed *inductive*. But could we not reason inductively in logic as well? This is a tantalizing thought since if this was possible, we would harness the sophisticated logical language for encoding observations about the world and at the same time we would be able to learn hypotheses from these observations by inductive reasoning.

Returning to Fig. 1, induction would mean going against the direction of the arrow, i.e. inferring the general statement from the more specific one. There is an obvious catch though: while going down the arrow is correct (any consequence of a true premise is also true), going up against the arrow may not be correct. The example in Fig. 1 shows this convincingly as the top formula just happens to be incorrect, at least in the interpretation (2). But potential incorrectness is simply at heart of any inductive reasoning even outside logic. Note that for the classification patterns (4) or (5) induced from the specific examples in Tab. 1, we had no guarantee either that these patterns were generally correct even if they had been consistent with these specific examples. However, we tend to believe the more in the validity of the patterns the more examples they are consistent with. So, the justification of induction is merely *quantitative*. Indeed, entire theories exist that formalize such justifications on statistical grounds.

In Fig. 4 we show formulas connected by arrows, each representing an entailment relation. So this is like in Fig. 1 except we involved more formulas and their relationships. The underlined formulas contain specific observations which translate to natural language as statements such as *whenever Plato is a man, he is mortal*. Expressing the observation this way seems overly cumbersome: why don't we just say *Plato is mortal*? While the latter would be true in the example world described by interpretation (2), it might not be true in different worlds where Plato would be e.g. the name of an immortal biographic book. So we do need the seemingly redundant condition in the observation formulas. The graph now makes it clear that multiple such observations (two more philosophers along Plato) have a joint possible cause *all men are mortal*, which could rightfully be termed a *hypothesis*. We could make an even

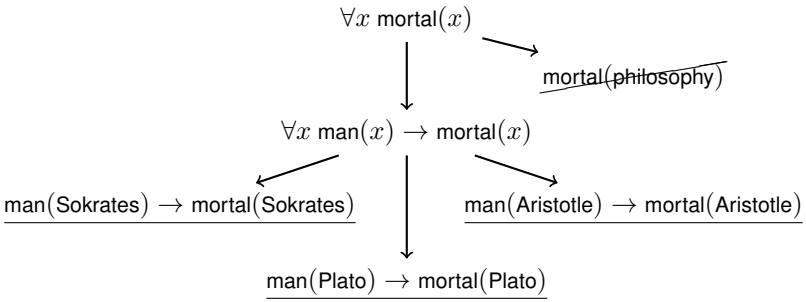

Fig. 4: Graph of entailment relationships among statements. The goal of induction is to find a hypothesis that entails all statements known to be true (underlined) but no statement known to be false (struck-out).

bolder step and generalize towards the top hypothesis *all is mortal*. The latter would be also supported by the three mortal philosophers so it seems we have an equally good justification for it. However, the top hypothesis entails the specific fact *philosophy is mortal* which we know *not* to be true and thus the fact contradicts and invalidates the hypothesis.

To summarize, the process of logical induction assumes that, to begin with, we have a set of examples which are true statements (underlined in Fig. 4) called *positive examples*, and another set of statements which are known not to be true (struck-out in the figure) called *negative examples*. To goal is to find a joint generalization of the positive examples which does not contradict the negative ones.

There is a prominent analogy with the machine-learning world: the two categories of examples (positive, negative) correspond to the two classes (setosa, versicolor) in the data shown in Tab. 1, and the sought hypothesis correspond to the classifier (4) or (5). But the machine-learning context also gives us an important lesson we should translate back into the logical world. In particular, we noted that perfect consistency on the training examples, if achieved at the price of a very complex hypothesis, tends to result in overfitting, that is, poor performance on future examples. So the more reasonable goal is to find a formula which entails *sufficiently many* positive examples and *sufficiently few* negative ones. And it is mainly statistics that determines what is sufficient in which conditions.

All in all, induction in logic may be framed by the equation

$$\text{induction} = \text{logical generalization} + \text{statistical justification}$$

How is the marriage of logic and machine learning relevant to bioinformatics? A great way to demonstrate this is a task of protein classification.

We have already learned that a protein is a sequence of amino acids which folds into a complex spatial form such as that shown in Fig. 3. We also know that protein functions are determined exactly by this fine geometrical conformation. One protein function we have elaborated already was the ability to bind to the DNA, which is a necessary condition for the protein to act as a transcription factor. But what exactly makes a protein able to bind to the DNA? One way to answer this would be to start from 'first principles', i.e. all relevant biochemical laws and the composition and form of the DNA, and infer deductively the conformation and properties of molecules in the protein required to enable the binding. Currently, this approach is simply impossible due to our limited knowledge of biochemistry and also due to limited available computational power.

A more viable option lends itself: collect a sufficiently large sample of DNA-binding and non-DNA-binding proteins, determine their spatial conformation, and have a machine-learning algorithms discover what properties or what parts of the protein are responsible for the DNA-binding ability. More precisely, the learning algorithm receives the protein conformation descriptions as examples of the positive (DNA-binding) and negative (others) classes and will learn a classifier to discriminate between these classes. A simple machine-learning exercise, right? Wrong.

We would face a problem straight from the beginning. A standard machine-learning algorithm will assume that input data are stored just like in the Iris example, that is, as rows in a table such as Tab. 1. But how on Earth do we fit the protein spatial shape into a row of values? In principle we could have three columns per one amino acid, corresponding to their x, y, z coordinates. We could even add more columns for amino acid properties such as hydrophobicity, polarity, etc. But this approach is lame at least for two reasons. First, the number of columns is obviously the same in all rows but different proteins contain a different number of amino acids. Second, it would be just arbitrary to choose which amino acid belongs to which column. A classifier that would work well for one possible cast would not work for a different one.

Logic, however, gives us a language to describe the protein examples that is much more elegant than the tabular representation. To express that a particular protein binds to the DNA as a result of its conformation is just like expressing that Sokrates is mortal as a result of being a man, except more complex:

$$\begin{aligned} \mathsf{binds}(\mathsf{protein1}) \leftarrow & \mathsf{amino}(\mathsf{protein1}, \mathsf{a1}) \wedge \mathsf{type}(\mathsf{a1}, \mathsf{histidine}) \wedge \\ & \mathsf{amino}(\mathsf{protein1}, \mathsf{a2}) \wedge \mathsf{type}(\mathsf{a2}, \mathsf{lysine}) \\ & \wedge \mathsf{distance}(\mathsf{a1}, \mathsf{a2}, 10) \wedge \ldots \end{aligned} \tag{6}$$

The shown part of the description only mentions the presence of two amino

acids (a1,a2) in the protein, their type (histidine and lysine) and their mutual distance but a much longer formula would of course be needed to describe the protein in full. Note that in contrary to the previous examples of logical formulas, we write the implication right-to-left, which is just for better readability. Also note that we now work with predicates which have multiple arguments, such as $\mathsf{distance}(\mathsf{a1}, \mathsf{a2}, 10)$. Such a predicate asserts a property of a tuple of objects rather than a single one, which is just like saying that it asserts a particular relationship among the tuple. Here, the relationship is the mutual distance (here 10, say the units are Angstroms) between the two amino acids represented as a1 and a2. Again, this is in contrast with the tabular representation (Tab. 1) where properties (numbers) are assigned to individual objects (dimension variables) but mutual properties are not expressed. In summary, the logical language also allows us to reason on the basis of relations and hence the name *relational machine learning* included in the title of this lecture.

Once we have overcome the representation problem, all that is left to do is to collect a sufficient number of positive examples such as (6) and analogical negative ones, and then find a suitable hypothesis by searching an entailment graph similar to that in Fig. 4 we used in the philosophers' domain. So we seem bound to happy ending.

3 Crisis *(Too slow, too crisp)*

But the road is thorny as the computations involved in induction contain two difficult problems.

Remind that induction can informally be viewed as search in an entailment graph such as that in Fig. 4. That, however, acquires gigantic sizes for real-life induction tasks. Indeed, if we replace the simple observations which are underlined in Fig. 4 by observations such as (6), the number of possible generalizations will become huge. This is obvious if one realizes that e.g. dropping a single condition such as $\mathsf{type}(\mathsf{a1}, \mathsf{histidine})$ (these fragments of formulas are called *literals*) in Formula (6) yields a possible generalization of the formula. And there are many literals to choose from; a real description of a protein would involve hundreds to thousands of literals. Besides, the number of example formulas forming the 'bottom' of the graph has to be much larger than shown in Fig. 4. This is dictated by machine-learning statistics as we remarked a while ago: to attain a reliable hypothesis, it has to be supported by a significant number of supporting specific observations. So in summary, the first big pitfall on the road is the complexity of the combinatorial search in the entailment graph.

The second big pitfall comes in disguise and is all the more pressing. Be-

fore we even think of searching an entailment graph, we need to know where the edges are, that is, which formulas entail which other formulas. In the first chapter we learned that entailment between two given formulas cannot be verified by brute force, i.e. by checking validity of formulas in all Herbrand intepretations (possible worlds). We also learned that algorithms exist which, in principle, can determine the entailment by constructing a logical proof of it. However, such algorithms are not always successful in that they may fail to find such a proof even if the entailment does hold. Interestingly, this is not due to some design flaws of the algorithms that would be rectified in the future. In fact, the entailment relationship between two formulas is in general *undecidable*, which means that no algorithm exists which would say yes or no correctly for any pair of formulas in finite time. Again, it does not just mean that no-one has yet invented such an algorithm, it means that no-one will be able to do so even in the future. Admittedly, this problem is somewhat theoretical as this undecidability only takes place if the logical vocabulary involves function symbols (remember from the first chapter that these render the discourse domain infinite). Looking at Formula (6), we do not seem to need any function symbols to describe a protein. Still, entailment checking remains a very complex subproblem of logical learning.

The two problems we just considered pertain to complexity and they limit the scalability of logic-based algorithms towards serious real-life learning problems. Another, more practical trouble arises when confronting the logic-based algorithm with real-life data, and that trouble is due to the crisp symbolic nature of the algorithm. The problem shows quite clearly with the protein classification problem we have considered. Assume for a while the simplistic idea that proteins bind to DNA whenever they contain a lysin and a histidine (two specific kinds of amino acids) in the mutual distance of 10.5 Angstroms. We would therefore expect the learning algorithm to produce the hypothesis

$$\begin{aligned} \forall x\ \mathsf{binds}(x) \leftarrow & \mathsf{amino}(x,y) \wedge \mathsf{type}(y,\mathsf{histidine}) \wedge \\ & \mathsf{amino}(x,w) \wedge \mathsf{type}(w,\mathsf{lysine}) \\ & \wedge\ \mathsf{distance}(y,w,10.5) \wedge \ldots \end{aligned} \tag{7}$$

Now the trouble is that the positive example (6) would not support this hypothesis, i.e. would not be entailed by it. The only reason for that is that the number 10 in the example and the number 10.5 in the hypothesis are simply different symbols, although they seem to us very close, say well within the range of measurement errors. Conventional predicate logic does not have means to project disparities in the arguments of literals into 'fuzzy' levels of entailment such as 'almost entails,' let alone to interpret such disparities in light of statistical distributions of measurement errors.

4 Peripeteia *(Trade-offs, middle-grounds, hacks)*

Induction in predicate logic has been studied for over thirty years (interestingly, mainly in Europe and Japan, less so in the United States) and the field was established under the title *inductive logic programming* [4]. Why programming? Computer programs written in the programming language called *Prolog* are in fact logical formulas and the execution of such a program is really proving the entailment of a user-supplied *query* formula by the program formula. As non-orthodox as it seems, any conventional computer program, i.e. a series of commands with conditional branching and looping, has its counterpart in Prolog. Now, inductive logic programming frames the scenario in which the user supplies examples of the program's behavior, that is, a set of query formulas which should be entailed by the program formula and a set of those which should not be entailed, and the inductive system automatically finds a suitable program formula. This is exactly the same scenario as we considered in Chapter 2 except the target hypothesis is viewed as a Prolog program and the search process is viewed as automatic programming.

The emphasis on the programming aspect did not last long in the developments of inductive logic programming. This is mainly becuase writing non-trivial computer programs in Prolog simply will not do without function symbols. As we have seen already, these make the entailment relation undecidable, and even if decidability is achieved through some ad-hoc constraints, the goal of automatic induction of full-fledged programs remains too complex and overambitious. So the inductive logic programming community shifted quite early the attention to the machine-learning classification scenarios we have already illustrated. And over the tens of years, numerous strategies have been proposed to tackle the critical problems outlined in the previous chapter. Here I want to explain a few examples of these but I mainly focus on those I have contributed to myself.

An obvious hack to prevent the explosion of the search graph in Fig. 4 simply drops all formulas over a certain size and considers only the small simple ones. Indeed, restriction to small formulas (typically up to 10 literals) is present in many implemented systems and is supported by an 'Occam's razor' kind of argument: we prefer simpler hypotheses over the more complex ones. However convincing, such argumentation will not help in tasks necessarily complex examples giving rise to complex hypotheses. An example of such a task is the protein classification problem we have been concerned with; to describe a protein by a formula such as (6), we need hundreds to thousands of literals! That is just beyond the reach of conventional systems of inductive logic programming.

As an example of a more sensitive strategy to accelerate the search in

the entailment graph, we will look at a research stream which I was part of and which explored *randomized* local search strategies [9]. The conventional, non-randomized strategy to search a graph as in Fig. 4 visits a formula (node in the graph) and if it is not acceptable then it moves on to one of its neighbors. It also keeps track of the visited formulas so that none of them is explored twice. This is essentially the trial-and-error approach we mentioned in the first chapter. Here, each error (unacceptable formula) gives us a clue on where to go next. For example, if we first try the top formula in Fig. 4, it turns out over-general as it entails a negative example. So whichever formula we try next, we know it should not be a formula even more general than the current one. A *randomized local-search version* of the strategy work just like the conventional one except for one change. If an acceptable formula is not found within constant time (or constant number of nodes explored), we simply abandon the entire search neighborhood and jump over to a completely different random node of the graph and start the local exploration again.

So the design of the randomized algorithm is simple and not all that amusing. What is more interesting are the reasons why it outperformed the conventional algorithms in most of our experiments. Intuitively, the random jumps seem appropriate for very large graphs. In these, the conventional neighbor-to-neighbor searcher will necessarily remain stuck in a relatively tiny region of it. Here, the random jumps prevent the algorithm from this unjustified bias towards a particular small region of the graph. A more mathematically founded argument for the superior performance of the randomized version follows a statistical consideration. Consider an experiment in which the conventional algorithm is run very many times and each time we measure the run time spent until a good formula was found. Interestingly, the statistical distribution of such run times follows a *power-law*, which is popularly known in economics as the distribution of people's incomes follows this law as well. For the run-time distributions, the power law means there is a significant proportion of very quick ('lucky') runs but also a high proportion of extremely long ones. The mean run time is then also very high (in some of the theoretical power-law distributions, they are even infinite). This is in contrast with a more usual *normal* (exponential) distribution where the extremely long runs would also have an extremely small probability and the mean run-time would be smaller. Interestingly, the randomized restarted strategy turns the originally power-law run-time distribution into an exponential one and thus the mean run-time is reduced.

Unfortunately, the gains achieved by randomized search were far from sufficient to work efficiently with the protein description such as (6). Part of the reason is that the randomized search strategy does not change anything about the second source of complexity we explained in the previous chapter,

that is the complexity of verifying whether a candidate formula entails an example formula. Naturally, this complexity is also aggravated if the formulas in question become large. This source of complexity is indeed an important factor since the entailment verification is embedded in the graph search and is permanently iterated in it. Again, a popular hack is in place that mitigates this problem somewhat. Normally, the entailment relation is confirmed by constructing a proof, as we explained in the first chapter. Many inductive logic programming algorithms do not create such a proof, instead they just compare the two formulas in question to see if one of them syntactically *subsumes* the other. We do not need to bother with the exact details of subsumption, the important points are that the subsumption check does not need to construct a proof, and whenever a formula subsumes another one, it also entails it. However, the reverse way is not always true: a formula may entail another one without subsuming it syntactically. So subsumption is only an aproximation of entailment but usually is faster to check. The speed gains are not dramatic though, subsumption checking still belongs among hard problems known in computer science as *NP-complete*. Furthermore, subsumption is only applicable to a restricted form of formulas (so called *clauses*) although this restriction is usually not overly limiting in practice; in fact all the formulas we have seen so far were clauses. Also in the case of subsumption, we were able to implement randomized variants which lead to certain speed-ups of subsumption checking [2], although again not sufficient to handle formulas with hundreds or thousands of literals.

It is interesting to mention one more approach that deals with the problems we identified in the last chapter, by finding a middle-ground between the principles of inductive logic programming on one hand, and conventional machine learning as was illustrated through the Iris classification example on the other hand. Put differently, the approach aims to convert the task of learning a logical formula into a dual task of standard machine learning. Recall that the latter assumes training samples organized as rows in a table such as Tab. 1. We have already argued why the structural descriptions of proteins just cannot fit naturally in such a flat table. Well, that was not entirely true. Imagine that in the protein case, the table's attributes (column) would correspond to some structural features expressed in our familiar logical language. For example, one column could correspond to the feature of containing a pair of lysine and hystidine in the mutual distance of 10 Anstroms, so it would be encoded as the conjunction of the five literals right of the arrow in (7). Each protein (row) which has this feature would have a "Yes" in that columns and other protein would have a "No" there. We can generate a large number of such features as long as they are relatively small and thus it is tractable to check whether they occur in the sample proteins (this is usually done by the

subsumption algorithm) and thus whether we should fill in a "YES" or "NO" in the appropriate cell in the table.

As a result, we have a tabular description of the training data which can be presented to one of a plethora of fast machine-learning algorithms. The latter could e.g. learn a classifier resembling our familiar Iris classifier (4):

$$\mathsf{f}_2 = \mathsf{Yes} \wedge \mathsf{f}_3 = \mathsf{Yes} \rightarrow \mathsf{class} = \mathsf{binding}$$

This rule conditions the binding function on the presence of the two predefined features f_2 and f_3. We can view the combination of the two (or generally a number of) features as one more complex feature. Thus, in principle the feature-based approach allows us to discover complex causes while eliminating the daunting search for a complex formula as in Fig. 4. Note, however, what we are trading off in this approach. The part of learning outsourced to the standard machine learning algorithm (i.e. the composition of simpler features into complexes) no longer accounts for any structural relationships between these features. So the rule above will apply whenever the two features are present in the protein, no matter what their mutual position is, whether they overlap, etc.

The feature-based approach outlined is termed quite cryptically *propositionalization* [8] as it converts the original predicate-logic descriptions of data into a table of simple Yes/No propositions. We have seen it is not a magical cure since we pay a price for the efficiency gained, in particular we lose some expressiveness of classification. To add to the credit side though, note that propositionalization can partially rectify also the crispness problem we have commented on at the end of the previous chapter. This is because the of Yes/No values in the produced table can be as well replaced by the respective numerical 0/1 values. Then a machine-learning algorithm can be used to produce a classifier such as (5) where these feature values would be weighted by real coefficients. So if one feature is significant yet less important than another feature, the learning algorithm can reflect this naturally by a suitable choice of the coefficients.

I have participated in research in propositionalization and its applications in bioinformatics tasks beyond proteomics. For example, relational features proved very useful to capture automatically discovered groups of co-regulated genes [7].

5 Catharsis *(Structural tractability)*

Experience gained with the hacks outlined in the previous chapter was valuable. Randomized local search reduced run-times of learning significantly but not just enough to learn from data as complex as the protein descriptions. Propositionalization using small tractable features had an even more

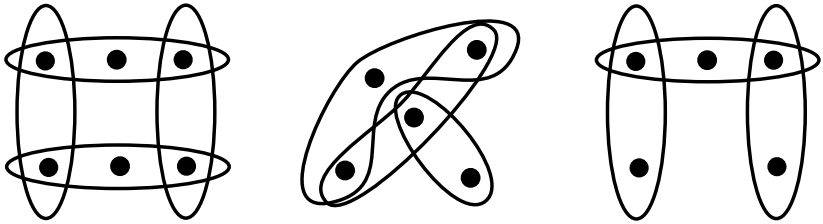

Fig. 5: Examples of hypergraphs, each showing the structure of a clause. "Edges" (the closed curves) correspond to predicate symbols in the clause. Nodes correspond to variables or object names. A predicate-edge encompasses all nodes representing the predicate's arguments. A single variable or object name may be inside multiple edges as it may occur as an argument in multiple predicates.

dramatic impact on run-times but it traded off what we wanted to maintain, i.e. the ability to express large structural motifs.

These peripeteias made us realize more clearly what ingredients will probably lead to the design of an algorithm that would succeed with such complex data. We realized that the language we use to express formulas would need to be restricted for sakes of efficiency but clearly we cannot just limit ourselves to small formulas as this would be a dead end in tasks such as the protein classification problem. Instead, we took inspiration in the *structural tractability* approach known in other fields of computer science. In general, the approach aims to identify some contraints on the graphical structure of the involved objects, which if satisfied, enable to work with these objects efficiently. In our case, the objects are formulas and their structures can be presented as graphs. We have already seen a graph in Fig. 4 in which nodes are formulas and edges represent entailment relations. This time, however, we mean a graph depicting the structure of a single formula, and we focus only on the special case of formulas called clauses (remind from the preceeding chapter that all the formula examples so far were in fact clauses) and we also assume there are no function symbols. In the clause graph, edges correspond to predicate symbols appearing in the clause and nodes represent the arguments in the parentheses, be it variables of the concrete names of objects. The situation is trickier now because a predicate corresponding to an edge may have an arbitrary number of arguments. This means that an edge will no longer be an arrow connecting two nodes but rather a closed curve encompassing a set of nodes. Graphs with such exotic nodes are called *hypergraphs* and three examples of them are shown in Fig. 5.

A popular subclass of standard graphs are so called *trees* which are connected (there are no groups of nodes isolated from the rest of the graph) and have no loops. Fig. 4 is a nice example of a tree. It is well known that many computational problems on graphs become much easier when all the graphs in question are trees. For hypergraphs, we can also distinguish trees (Fig. 5, right-most) although their visual character is less intuitive. The important point is, however, that for clauses whose hypergraphs are trees, we were able to design a learning algorithm [3] that combats successfully both of the crucial complexity sources we are concerned with. That is, it efficiently verifies entailment and it also efficiently searches for a good clause. The entailment relation is approximated via subsumption in the same way we have already elaborated. However, as a result of the assumptions of the algorithm (function-free tree-like clauses), subsumption in fact becomes equivalent to entailment. The search for a good clause differs from the standard neighbor-to-neighbor search we are familiar with already. It is rather based on progressive composition of smaller blocks ('subclauses') into larger blocks, and their combination into still larger blocks, and so on. In follow-up research, we (well, mainly my Ph.D. student Ondřej Kuželka) were able to relax the assumption of tree-likeness towards a softer constraint of *bounded tree-width*, which is a concept out of the scope of this lecture.

With this fancy algorithm, we were finally able to work efficiently with protein descriptions translating to very large yet tree-like clauses. As a cherry on top, we implemented this approach within the propositionalization framework. This means that the algorithm does not produce a single classifier formula but rather a large number of (possibly complex) clauses expressing structural features of the protein. A nice property is that all these features are guaranteed to be 'relevant' in a statistical sense; for example, each of them is found in significantly more samples of one class (binding proteins) and in significantly fewer samples of the other class (non-binding). Then a conventional machine-learning algorithm is employed to explore yet more complex combinations of these base features. As we have remarked earlier, such a combination need not be just a crisp logical conjunction, it may be a finer feature 'ensemble' where features are e.g. weighted according to their reliability. This is an additional benefit of the two-stage procedure consisting of propositionalization and subsequent attribute-value learning.

And only with this arsenal of methods, we succeeded in learning to classify DNA-binding proteins with accuracy matching the best state-of-the-art methods [6]. Unlike in these methods, based on long-tuned hand-crafted physico-chemical features, our approach discovers such features automatically and represents them in a way straightforwardly interpretable as protein structural motifs.

References

[1] G. Freytag. *Die Technik des Dramas*. S. Hirzel, Leipzig, 1863.

[2] O. Kuzelka and F. Zelezny. A restarted strategy for efficient subsumption testing. *Fundamenta Informaticae*, 89:95–109, 2008.

[3] O. Kuzelka and F. Zelezny. Block-wise construction of tree-like relational features with monotone reducibility and redundancy. *Machine Learning*, 83:163–92, 2011.

[4] S.-H. Nienhuys-Cheng and R. de Wolf. *Foundations of Inductive Logic Programming*. Springer-Verlag, New York, 1997.

[5] S. J. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2002.

[6] A. Szaboova, O. Kuzelka, F. Zelezny, and J. Tolar. Prediction of DNA-binding proteins from relational features. *Proteome Science*, 10, 2012.

[7] I. Trajkovski, F. Zelezny, N. Lavrac, and J. Tolar. Learning relational descriptions of differentially expressed gene groups. *IEEE Trans. Sys Man Cyb C*, 38(1):16–25, 2008.

[8] F. Zelezny and N. Lavrac. Propositionalization-based relational subgroup discovery with RSD. *Machine Learning*, 62:33–63, 2006.

[9] F. Zelezny, A. Srinivasan, and D. Page. Randomized restarted search in ILP. *Machine Learning*, 64:183–208, 2006.

doc. Ing. Filip Železný, Ph.D. (born on March 5, 1974)

9/2013 - today	*chair*, Dept. of Computer Science, CVUT FEL
11/2010 - today	*associate professor*, CVUT FEL
4/2006 - today	*head of the Intelligent Data Analysis research lab*, CVUT FEL
3/2004 - 11/2010	*assistant professor*, CVUT FEL
9/2004 - 11/2004	*visiting professor*, State Univ. of New York in Binghamton
3/2003 - 3/2004	*postdoctoral researcher*, Univ. of Wisconsin in Madison
3/2003	*Ph.D.*, Czech Tech. Univ. in Prague
6/1998	*Ing. ($\sim$ MSc.)*, Czech Tech. Univ. in Prague
6/1996	*BSc.*, Czech Tech. Univ. in Prague

Journal papers

- Kuzelka O., Szaboova A., Zelezny F.: A Method for Reduction of Examples in Relational Learning. *Journal of Intelligent Information Systems*, 42(2):255-281, 2014
- Wohlfahrtova M., Brabcova I., Zelezny F., Balaz P., Janousek L., Lodererova A., Honsova E., Wohlfahrt P., and Viklicky O.: Tubular atrophy and low netrin-1 expression levels are risk factors associated with delayed kidney allograft function. *Transplantation*, 97(2):176-183, 2014
- Bartak R., Cernoch R., Kuzelka O., Zelezny F.: Formulating the Template ILP Consistency Problem as a Constraint Satisfaction Problem. *Constraints* 18(2):144-165, 2013
- Szaboova A., Kuzelka O., Zelezny F., Tolar J.: Prediction of DNA-binding proteins from relational features. *Proteome Science* 10:66, 2012
- Szaboova A., Kuzelka O., Zelezny F., Tolar J.: Prediction of DNA-binding Propensity of Proteins by the Ball-Histogram Method using Automatic Template Search. *BMC Bioinformatics* 13(Suppl 10):S3, 2012
- Holec M., Klema J., Zelezny F., Tolar J.: Comparative Evaluation of Set-Level Techniques in Predictive Classification of Gene Expression Samples. *BMC Bioinformatics* 13(Suppl 10):S15, 2012
- Urbanova M., Brabcova I., Girmanova E., Zelezny F., Viklicky O..: Differential Regulation of the Nuclear Factor-κ-B Pathway by Rabbit Antithymocyte Globulins in Kidney Transplantation. *Transplantation* 93(6):589-96, 2012
- Kuzelka O., Zelezny F.: Block-Wise Construction of Tree-like Relational Features with Monotone Reducibility and Redundancy. *Machine Learning* 83(2):163-192, 2011
- Zahalka J., Zelezny F.: An Experimental Test of Occam's Razor in Classification (Technical Note). *Machine Learning* 82(3):475-481, 2011
- Zakova M., Kremen P., Zelezny F., Lavrac N.: Automatic Knowledge Discovery Workflow Composition through Ontology-Based Planning. *IEEE Trans. Automation Science and Engineering* 8(2):253-264, 2011
- Kuzelka O., Zelezny F.: A Restarted Strategy for Efficient Subsumption Testing. *Fundamenta Informaticae* 89(1):95-109, 2008
- Trajkovski I., Zelezny F., Lavrac N., Tolar J.: Learning Relational Descriptions of Differentially Expressed Gene Groups. *IEEE Trans. Sys Man Cyb C* 38(1):16-25, 2008.
- Klema J., Novakova L., Karel F., Stepankova O., Zelezny F.: Sequential Medical Data Mining: A Case Study. *IEEE Trans. Sys Man Cyb C*, 38(1):3-15, 2008.
- Zelezny F., Srinivasan A., Page D.: Randomized Restarted Search in ILP. *Machine Learning* 64(1-2):183–208, 2006.

- Zelezny F., Lavrac N.: Propositionalization-Based Relational Subgroup Discovery with RSD. *Machine Learning* 62(1-2):33-63, 2006.
- Gamberger D., Lavrac N., Zelezny F., Tolar J.: Induction of comprehensible models for gene expression datasets by subgroup discovery methodology. *Journal of Biomedical Informatics* 37(4):269–284, 2004
- Zelezny F.: Efficiency-conscious Propositionalization for Relational Learning. *Kybernetika 4(3):275–292*, 2004

Selected Conference Papers

- Kuzelka O., Szaboova A., Holec M., Zelezny F.: Gaussian Logic for Predictive Classification. *ECML/PKDD 2011: 22th European Conference on Machine Learning / 15th European Conference on Principles and Practice of Knowledge Discovery*
- Kuzelka O., Zelezny F.: Block-Wise Construction of Acyclic Relational Features with Monotone Irreducibility and Relevancy Properties. *ICML 2009: the 26th International Conference on Machine Learning*
- Kuzelka O., Zelezny F.: Fast Estimation of First-Order Clause Coverage through Randomization and Maximum Likelihood. *ICML 2008: the 25th International Conference on Machine Learning*
- Zakova M., Zelezny F.: Exploiting Term, Predicate, and Feature Taxonomies in Propositionalization and Propositional Rule Learning. *ECML/PKDD 2007: 18th European Conference on Machine Learning / 11th European Conference on Principles and Practice of Knowledge Discovery*

Editorials

- Riguzzi F., Zelezny F.: Guest editors' introduction: special issue on Inductive Logic Programming. *Machine Learning* 94:1-2, 2014
- Blockeel H., Kersting K., Nijssen S., Zelezny F.: Guest editor's introduction: special issue of the ECML PKDD 2013 journal track. *Data Min. Knowl. Discov.* 27(3): 291-293, 2013
- Blockeel H., Kersting K., Nijssen S., Zelezny F.: Guest editor's introduction: special issue of the ECML PKDD 2013 journal track. *Machine Learning* 93(1): 1-3, 2013
- Blockeel H., Kersting K., Nijssen S., Zelezny F.: (Eds.): Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Parts I, II, III. *Lecture Notes in Computer Science* 8188, Springer 2013
- Riguzzi F., Zelezny F. (Eds.): Inductive Logic Programming - 22nd International Conference, ILP 2012, Dubrovnik, Croatia, September 17-19, 2012, Revised Selected Papers. *Lecture Notes in Computer Science* 7842, Springer 2013
- Riguzzi F., Zelezny F. (Eds.): Late Breaking Papers of the 22nd International Conference on Inductive Logic Programming, Dubrovnik, Croatia, September 17-19, 2012. *CEUR Workshop Proceedings* 975, CEUR-WS.org 2012
- Zelezny F., Lavrac N.: Guest editors' introduction: Special issue on Inductive Logic Programming. *Machine Learning* 76(1):1-2, 2009
- Berendt B., Mladenic D., de Gemmis M., Semeraro G., Spiliopoulou M., Stumme G., Svatek V., Zelezny F. (editors): Knowledge Discovery Enhanced with Semantic and Social Information, Springer 2009
- Zelezny F., Lavrac N. (editors): Proceedings of the 18th International Conference on Inductive Logic Programming (ILP-2008), *Lecture Notes in Computer Science* Springer 2008