[Music]

Welcome to the third lecture of the fifth week of the course in machine learning this lecture will be about Inductive Logic Programming. Inductive logic programming abbreviated ILP is an efficient attempt to marry together the area of logic programming with techniques from machine learning establishing what one can call learning with logic. One of the big advantage here with this mission is to create that kind of framework in which you can easily express both deductive and inductive impresses in a very uniform way. So if we look at the two main forms of inference that first deduction, when we actually apply rules to facts and then can infer other facts it's a very traditional way and this kind of inference it is a cornerstone of what you do in logic program so actually we do not have to worry about that. The new thing with inductive logic programming is to see to that we can also make inferences the opposite way that we can start with just the collection of facts and from those facts infer some rules given of course as you hopefully have understood that it will only have some credibility, if we handle a reasonably large amount of instances or data items to work from. Inductive logic programming, the goal of ILP is to find hypotheses expressed as logic programming clauses from a set of positive and negative examples and in the presence of a domain theory where the latter two also are expressed in the same logic programming formalism. So what we have given is a set of positive and negative examples expressed in this observation language we call LE, domain theory T, hypothesis language LH that delimits the clauses that allowed in the hypothesis space H and our relation covers, covers is important because covers is the means for us to decide whether a certain hypothesis covers a certain entails a certain example E, considering also the background theory. So given these pieces, logic programming has the aim to find hypothesis H that covers all positive examples, no negative examples. So we can say that the learning goals of inductive logic programming expressed in this way is extremely much aligned with the learning goals for all other kinds of inductive logic programming. What is not covered in this is of course, the basic capabilities of logic programming as such, which from the start enables deductive reasoning, all executions of a pure logic programming can be regarded as an instance of deduction. So I say this because if you only define inductive logic program as inductive, you forget the deductive side and one of the big positive things with this area that the combination is made possible.

Let us now talk about some of the main advantages of ILP, actually the most important advantage is that by using the same framework the language of logic of logic programming. It's possible to express examples, hypotheses and domain theory or background theory in the

same language. Also the second thing important thing is that one can handle deductive inference and inductive inference within the same framework. Because of the ability for logic programming to support structured data types and multiple relations in a convenient way, it also makes ILP by inheritance from logic programming, also very suited to handles domains where there is a need to conveniently describe complex structures. One example of that is chemistry and the complex chemical structures that many applications demand. One thing that maybe is not the biggest advantage because of course many other systems and approaches of similar features is that some ILP systems also have the possibility to invent new predicates and add them to the domain theory, so this is this dynamic property that you also can introduce here.

As have been said, the nice property of inductive logic programming is that everything is represented in the same way. So this means that hypotheses or examples, facts everything in in a way are in the end represented at logic programming clauses. So at the core of inductive logic programming are the rules of specialization, relation and generalization. And these two rules or relations connect hypotheses and facts. So one can say that hypothesis let's call it G is more general thank hypothesis S, if an only G entails S. And then S is also said to be more specific than G. So there are two then kinds of other relations that are used in different way, so let's start with a so-called specialization rule or specialization relation. So if you have a deductive inference rule R, that map's a conjunction of clauses G onto a conjunction clauses S such as G entails S, this is a specialization rule. And the typical rule in inductive logic programming of that kind is called Theta subsumption. If we set up hypothesis space and structure that space using the Theta subsumption specialization relation, this search space will be a lattice, and for every pair of clauses under the specific conditions of the Theta subsumption that will exist at least upper bound and the greatest lower bound with respect to the Theta subsumption relation. So it's a pretty well ordered structure. In this context what we talk about is inductive process that starts where we start the algorithms from the top, so we start with the most general hypothesis and this hypothesis can then be specialized in order to finally reach the facts. The opposite direction is the generalization, where we essentially start bottom-up and that so there we have a similar generalization rule. The typical ILP rule of general size and type is called inverse resolution. There's also some variants of that something inversion entailment, but essentially for that for given how much we time we spent on this topic and it's fine without one example of a top-down and one of the bottom-up operation. And in the same fashion a typical generalization rule like inverse resolution

establish a least generalization entity from positive examples. It's important to realize that that in this context of inductive logic program induction is viewed as the inverse of deduction as far as possible. So that the basis for the way of setting up algorithm for designing the algorithms and finally one comment is that in inductive logic programming the view of the process of induction is a search process. So this means that what happens in an inductive logic programming, induction process is that we search for the appropriate hypotheses but in a search space structured through the generalization and specialization relations, so therefore we when we design an algorithm we have to marry together the properties of the search algorithm and the prompt and specific properties of these relations.

We will now look at what is termed here the Generic ILP algorithm so first it should be repeated again, that inductive logic programming views thus the establishment of hypotheses as a search process. So there we of course always have our standard search process could be depth first, it could be breadth first, it could be best first, it could be anything. Okay, so that's the baseline, so all algorithms will be somehow phrased in a search process context. Secondly I already said that we have two kinds of main generalizations that we use for structuring hypothesis space. So we have a generalization and we have a specialization relation, and the generalization is more relevant to use in a bottom-up perspective, in a bottom-up process when the specialization is more relevant to use in a top-down setting. This algorithm that we're going to look now should potentially be able to work in both settings. So in both settings we somehow initialize the set of possible relevant hypotheses. Naturally if we have a top-down approach we would probably instantiate to the most general hypothesis as possible, while we have a bottom-up approach we will instantiate hypothesis with actually the data items of the case we handle. But this algorithm describes very simple iteration where in each stage of the algorithm we take an take hypothesis from this list of hypotheses and we apply the inference rules, that we have chosen to and as already said for top-down is typically Theta subsumption and for bottom-up is typically inverse resolution, so those are the typical choices in inductive logic programming, But anyway independently of what we choose we transform this list of current hypothesis through systematic apply and the shows and inference rules in one of the other direction, but in each step after we generated a new set hypothesis it may be so that we invented - too many or redundant or irrelevant hypotheses, so therefore in every cycle there is a proven step when we're using some criteria design to prove some of their processes in the list. And actually then this list goes on until some stop criterion is satisfied. On this line we seem clear elaborate a few of the items mentioned in the algorithm, so as

already stated there are a few dimensions here first is the direction of the process whether it's top-down or bottom-up and that of course influence how we initialize and hypothesis differently if we start top down and in bottom up. Also that influence the choice of operation inference rules or operations. So furthermore there is also another distinction for every algorithm we have to decide on which kind of basic search strategy we apply. So we can go for depth first we can go for breadth first we can go for best first and etc. And actually it turns out that as this algorithm is designed or written, it's actually two places in the algorithm where our actions or choices decide controls, the search strategy that we actually plot. So actually by the design of how what we do when we take away hypothesis delete and I processes to expand it and also what we do when we actually do exactly when we proven something, that combined choice we control what search strategy we will have so with so with certain actions in these places in the algorithm we will get a depth-first search strategy, otherwise we will get the breadth-first and so on. So final comment on this line is at the stop criteria of course your state when we should finish and actually the most natural thing we start when we the main thing main condition under which we stop is that we achieved at least one hypothesis in the hypothesis layer style that is good enough.

There is one point in the algorithm after the point where we actually transformed the hypothesis space in each iterations based on an expansion based on generalization and specialization operate, so we've got a new set of hypothesis. So than actually at that point we are supposed to look into whether it's possible to prove the hypothesis set. So there are two main criteria we could use to decide on what to prove. So the first case have to do with this space we have hypotheses that actually do not cover all instances, which means that it's too special, it's not general enough. So each hypotheses we have is not general enough and but in our set we have other hypotheses that are specializations of this one, then it's not meaningful to pursue them because the first time positives we talked about it's not general enough. So more special hypotheses then that one is not meaningful to keep. And similarly if we have the situation that the hypothesis we have is too general, which means that it also covers some negative instances, then it's already too general. So therefore if we have a generalization of this hypothesis, it's even worse so it's not meaningful to keep either. There will be also other criteria but these I think are the most primary ones to consider at this point. And we also had a stop or termination condition in this algorithm, so actually the question is when do we naturally stop the iteration. So one attempt which I will be in this slide to define something correct hypothesis. So then given that when I've defined that one can say that a natural stop

condition for finding in hypotheses is whether if you have established one correct hypothesis among your hypotheses. So a definition of correct hypothesis here is that it satisfies four requirements, so first such hypotheses should be sufficient in the sense that it covers all the positive examples we have at hand. It should also satisfy the requirement of necessity which means that if we take it away not all positive examples are covered by any other hypothesis so therefore it's needed we will not have coverage without it. The third requirement is weak consistency. It means that the hypotheses do not contradict any element of the domain theory. The last part is strong consistency, which means that their poses are not consistent with the negative examples. So if I follow this satisfies all these requirements we will call it correct and then it could be a candidate for being that the combined criteria could be candidate for using for termination criteria.

Let's now look at a few examples so the first pretty short example is about that kind of algorithm because it's based on generalization and this means that means bottom-up process where we use inverse resolution as the main generalization operation. So in this case we start from the facts, so we start from the facts that we have in the case. So father, adam. Adam is the father of Kain, Adam is male, Adam is the parent of Kain and so on. So we use these facts to infer the new rule father is implicated by male and so and as you can see it's natural to see that as an inverse resolution because at the top you can see the example of a revolution where father simply by a male, male adam will make us conclude that Adam is the father of Kain. Okay, if we look in the other direction now the top-down direction, the specialization operation is the Theta subsumption typically, so what is exemplified, here first we take it for propositional logic is that you can see that the hypothesis space naturally for a very simple case in propositional logic, the subsumption operation generates a lattice and also you can see that every man every element in this that is every pair of elements in this lattice has a least upper bound and it a grade is lower. So this is a very neat structure, as you can see in the next slide it's the same case, this also works for predicate logic, so for logic programming we will end with structures that look approximately like this.

To compare the top-down and bottom-up approaches let's introduce another example. So here we have a simple example we have an hypothesis which is bird X, we have some positive examples like that penguin is a bird, eagle is a bird, sparrow is a bird,and then we have some background knowledge saying the penguin, a sparrow and eagle lay eggs as all birds do, however only the sparrow and the eagle flies and only the Eagle has talents. So then if we first look at the hypothesis space, for this example from a bottom-up perspective where we

have inverse resolution as the relation that defines the structure, so you can see here that an eagle being a bird is defined by the close at the bottom. Bird X is lays eggs, flies X and has talons. While bird sparrow need to be an instance of the more general concept part of eggs with lays eggs and flies and so on. So by static by the facts the general algorithm would construct hypotheses within this space if we the same fashion look at the top-down search on the Theta subsumption,it will just be reversed. So hopefully this simple example can give you a flavor of the basic structures that always have to underlie the inductive processes an inductive logic programming, independently of whether they are top-down or bottom-up.

I want to wrap up this lecture by first giving you some examples of ILP systems and then gave you two examples of important application areas. So first ILP systems, so you get four examples here FOIL, PROGOL, Golem, and Marvin. Actually two top-down approaches and two bottom-up approaches and as you also say they represent different kinds of search strategies, so Foil and Golem are hill climbing while PROGOL have implemented the best first search. Probably the most well-known systems of all these are FOIL and PROGOL.

When we talk about application areas those two strongest application they have an inductive logic programming so far, is actually in natural language processing and bioinformatics. Maybe not surprisingly so because these areas we have high demand for being able to represent complex structures. So therefore logic programming provide a very good basis for both describing the examples, hypothesis and the domain theory, and want to say there have been few success stories in this areas over the years. This was the last part of this lecture thanks for your attention, the next lecture five point four will be on the topic of reinforcement learning thank you.