[Music]

Welcome back to the third lecture on the fourth week the course in machine learning. We will continue our discussion about decision tree learning algorithms. We will spend most of the time to talk about the ID3 algorithm. So ID3 which is short for Iterative Dichotomiser 3 is a TDIDT (Top-Down Induction of Decision Tree) algorithm, invented by Ross Quinlan in 1986. The TDIDT algorithm returns just one single consistent hypothesis and considers all examples as a batch. This kind of algorithm employs a greedy search algorithm, which means that it performs local optimizations without backtracking through the space of all possible decision trees. Obviously it's susceptible to the usual risk of hill climbing without backtracking and as a consequence finds tree with short path lengths typically but not necessarily in the best tree. This kind of algorithm selects and orders features recursively according to a statistical measure called Information Gain, which we described earlier in part 1 of this lecture, and until each training example can be classified unambiguously. Some kind of inductive bias is built into this algorithm. On one hand it's prioritized simplicity, which means this applies Occam’s razor by always choosing the simplest tree structure possible. But also as already mentioned it's systemically prioritize for high Information Gain when selecting features to discriminate among instances.

I assume you all know William Occam, a monk in the 15th century started to study logic, and who for was the first to phrase the principle of choosing always the simplest solution to any problem. So when there is a choice of alternatives, always use the simplest variant. This is relevant for discussion we are now but as you hopefully already know, is a very useful principle in a number of situations.

The ID3 algorithm starts with the original data set as associated with a root node. On each iteration of the algorithm the algorithm, it considers every unused feature and calculates the Information Gain of that feature. It then selects the feature which has the largest Information Gain value. The data set is then partition by the selected feature to produce subsets of the data that is then associated with the branched out nodes corresponding to the values of the chosen feature. The algorithm continues to recur on each subset considering only attributes never selected before. Recursion on a subset may stop in one of these cases, case 1 if every element in the subset belongs to the same class, the node is turning to a leaf node and labelled with a class of these examples. If there are no examples in the subset it's an empty set a leaf node is great in a label with the most common class of the example in the parents nodes set. If there

are no more attributes to be selected, but the example still do not belong to the same class, the node is made a leaf node and the label with common class of examples there in the subset.

So throughout algorithm the decision tree is constructed with each non-terminal node representing selected feature, on which the data is split and terminal representing the class label they're suited for the final subset of this branch. Let us now take a look short look at the pseudo-code for the ID3 algorithm. I already can formally characterized how the algorithm work but hopefully by going through this pseudo code shortly you need consolidate the understanding of how it works. So actually this algorithm takes as input a set of instances, set of classes and a set of features and what it does is that it returns a tree actually. It says return node in the end but actually the node well due to the function of the procedure the algorithm returned actually a tree. So it creates the root node in the first iteration and then it checks if only instances that it I belongs to the same class. So then you have actually node with maximum purity, then you return a single node tree with class label belonging with a class label belonging to those instances. If the feature set is empty, there are no more features to choose from in order to split the tree, then you also have to stop the algorithm and what you return is this single node with a label that is the most common label class label still in the set of instances that you have. Otherwise you still have instances, you still have classes, you still have feature, so you select the feature you select a feature by first calculating the maximum Information Gain again for each feature and then you select the feature with high as the information gain. And for each value of that feature, you add a new branch belonging to that feature, then you create a new subset of the instances that satisfies that feature value, if that instances is empty then you just create a leaf node with the most common class label you found among those instances. Otherwise you make a recursive call to the ID3 algorithm again, using the arguments now the instances but only those instances to belong to this branch of course you return the also have a parameter of the classes, but then you also include the features but of course you have to remove the feature already used because putting this algorithm never reused in in this process. And finally the algorithm stops by returning the node. So that's the functionality.

Let us now reverse it the simple example again that we introduced in part 1 of this lecture, so this was the example where we look at this 14 instances which are characterized by a number of features outlook, wind, humidity or temperature and so on. And in this slide it's depicted which kind of decision tree is produced by feeding those examples to the ID3 algorithm. So as you see we had 49 data items to start with 9 positive 6 negatives obviously the information

gained test decided that outlook was the most relevant feature to use in the first place to discriminate among the instances and recursively down the way humidity was chosen, eventually wind was chosen, so and as you can see given the decision or considering those features in that order, the dataset post partitioned, so that 5 data items was associated with the outlook sunny case, 4 with these outlook overcast case rain, 5 items with the rain outlook case and so on. As you see we ended up with actually 5 leaves of this tree, and it turns out that in this case only 3 of the four features were used to produce the decision tree. I should the comment on the phenomena of noise. Non-systematic errors in the values of features or class labels are usually referred to as Noise. Typically two modifications of the basic algorithmic are required if the tree building should be able to operate with the noise affected training set. The algorithms must be able to work with inadequate features, because noise can cause even the most comprehensive set of features to appear inadequate. And secondly the algorithm may be able to detect if testing further attributes will not improve the predictive accuracy of the decision tree but rather result in overfitting and the algorithm must as a consequence be able to take some measures for that like pruning.

Now we turn to the problem of overfitting. So one serious problem for decision trees is the risk of overfitting, the practical significant practical difficulty and for decisions is many other predictive models. Overfitting happens when the learning algorithm continues to develop hypothesis that reduce training set error at the cost of an increased test set error. So let's introduce a few definition so let's say that the average error for in our processes the training data is called ET and the corresponding average area for the training data plus the data in kind of all data, all together is ED. So we define overfitting in the following way, so if we have an hypothesis H, we say that this hypothesis is overfitting training data if the ET(h), which means the training data error for this hypothesis is less than some other than ET for some other hypothesis ET(h ‘) but the ED the error for test data of h is larger than the ED for tests for some other hypothesis. So this means actually that there are better other hypothesis now with respect to the accuracy for test data. So I said you can see in the example to the right and where you can see on the x-axis the size of the tree so when the tree grows the accuracy becomes better for the training data but it's actually more like a goes down for the test data.

So for decision tree the most important approach to handling overfitting is through pruning. So pruning is the major approach in this case and the idea with pruning is of course reduce the size of the decision tree without reducing predictive accuracy as measured by a test set or

cross-validation set. So there are two kinds of pruning, so on one hand we can have pre pruning where you stop to grow the tree earlier before it perfectly classifies the training data set, which means that no longer data split is statistically significant. So criteria for stopping are usually based on just the statistical tests like the chi-square tests and of course the aim of this test is to give a ground for decisions whether to expand particular node or not. The problem with this test even if they those tests exist is it's still not a trivial problem. So the risk of stopping too early is imminent. The alternative which is essentially a much more popular approach is called Post-Pruning, where we allow the tree to grow so it perfectly classifies the training set and then afterwards we post-prune the tree by removal of sub trees. I mean common approach here is to set aside some part of the data set that we could call a validation set, evaluate and use that validation set as a basis for the decisions in the post pruning phase. One variant, simple variant of post pruning is called Reduced Error Pruning and the scheme for that looks roughly as follows. So actually the data set is split into a training and a validation set. And the validation set is used as a basis for the pruning procedure. All nodes are iteratively considered for pruning. The node is remove if the resulting tree performs no worse than the original on the validation set. Pruning means removing not only the node but the whole subtree for which no one is the root making it a leaf and assign the most common class of the associate instances. Pruning continues until further pruning is considered as deteriorating accuracy. So this is a pretty simple procedure but it's pretty freely reflects many schemes for post pruning. Before we leave the talk down induction of decision tree algorithms, I will say want to say something about the some other algorithms. So actually ID3 I would say is the prototypical algorithm and therefore I've been using it for exemplification, however there were much earlier systems of this kind one of those called CLS and actually ID3 is an extension of CLS. And also Quinlan who developed ID3 fold up this work and there are actually more current versions that are in practical use that's called C 4.5 and there is also something called C.5 and actually C4.5 was at a time consider the default machine learning algorithm, that means a very popular tool to use for machine learning. Then there are others ACLS, assistant, CART etc. So there are many alternatives in general one can say that the later system extends ID3 in various ways but the primary ways are actually that it extends the kind of data types that are allowed for the features because ID3 was pretty restrictive here. Also the later algorithms are much better with respect to pruning and they're also much better with respect to noise handling.

So to further emphasize what I said for the last slide you can see on this line a subset of these systems are compared, so let's just look at ID3 and C4.5. And then you can see that on the point where C4.5 is working a better performance than ID3 because it can handle more wider range of feature values, it can handle missing values with more or less me is equivalent that it can also handle noise and obviously it has a more sophisticated pruning techniques. Still there are certain things that are not handled in C4.5, like also handling the case of Outliers which is also a common phenomenon, but as you can see another system a CART has mechanism for handling that. So far we have talked primarily about approaches to building decision trees which have the purpose to build one single tree. As you remember most of these methods performs in a greedy fashion which means that they take local decisions to form the tree therefore cannot be guaranteed to give to result in a a globally optimal tree. So there is another category of approach is called Ensemble Approaches, where assemble methods which construct more than one decision tree and use the set of trees for your classification. So there are two kinds of approaches relevant which are not only relevant for decision trees but for different kinds of classifiers. So we have boosting approaches where boosting means that we take a sequential approach where a sequence of average performing classifiers, it can be one decision tree, second decision tree, third decision tree, can give a boost performance by feeding experience from one classifier to the next. So we use experience on the first to improve the performance of the second. One example of such a system is called Ada Boost which is not only applicable to decision trees but to many ML algorithms. The other variant is called Baaging approaches where bagging is a parallel approach, where a set of classifiers together can produce partial results fully in parallel that then can be handled, can be used to form the basis for a total negotiated result. So one example of that is called Random Forest algorithm which combines random decision trees with bagging methodology to achieve very high classification accuracy. So to take a concrete example of an ensemble method we can look at random forests or random decision forests which are techniques can be used for classification, regression and other tasks. So random forests operates by constructing a multitude of decision trees at training time and outputs the class that is the most common of the classes or mean predictions produced as results from the individual trees. So the random forest approach is also an alternative remedy for the decision tree problems of overfitting, as you can see and you can see to the left you can see illustrated how the outcome of the individual trees are used as a basis for some kind of majority voting to decide upon a final class or a final regression value, and to the right you can also see in the strategy that in many cases the accuracy becomes better, so when using the random forest approach you can get a

much more smoother result that is closer to the desired output. So this was the end of lecture 4.3 and thanks for you attention. The next lecture 4.4 will be on the topic of instance based learning thank you and good bye.