[Music]

Welcome to the second part of the lecture 4.4 on instance-based learning. I will now talk about something called Distance Weighted Nearest Neighbor Algorithm which is actually a pretty straightforward extension of the k nearest neighbor classifier, and if we consider that k nearest neighbor classifier can be viewed as assigning the k nearest neighbors a weight of 1 and all others await all 0, the natural extension is to also weight the nearest neighbors, that is that the *ith* nearest neighbor is assigned weight *wi* from I = 1 to k, where k the number of neighbors we consider. So normally one have to define a specific function that computes this weight, one where are the common choice for that function is to take the inverse of the square of the distance between the instance to be classified and the training instance in question, but this function could also be some of those kernel functions that we will described a little later in this lecture. So then there are two cases of course as for the k nearest neighbor algorithm, so we have the classification version where the output is a class membership, so actually then what happens is that the query instances assigned the class label most common, amongst k-nearest neighbors but where the vote of the each neighbour i is weighted with *wi*. So it's not an equal vote for the weighted vote. So in k-NN regression where the result should be property value, so we calculate this value as the weighted sum over property values divided by the sum of the weights. So it's pretty straightforward extension from the normal k, nearest neighbor case. It's also possible when you weight that you consider to extend the k nearest neighbor from k to all data items, I mean this means that you would consider all data items in this process. So one can say that two versions here of the nearest neighbor that you keep to a normal k number like in the original algorithm, we call that a local weighted method or we can extend this to all data items and we call this method the global weighted method. So the normal weighted nearest neighbor algorithm can handle both classification and regression, however in both cases the algorithm approximated target function for one specific query instance, that means for one point in space. So we have another concept here Locally Weighted Regression which is actually where we extend this approach of regression from one point to a surrounding around a query instance q, and if we look at the term so there is a reason why it's called local, because we do regression not over the whole space but we do regression based on data near to xq. And of course we still keep the word weighted here and that the contribution of the training instance are weighted based on the distance from xq. So the weights are defined typically by a function and as you will see we will call this kind of function a kernel function, but actually generally it's a function where the weight of an

instance the contribution of an instance is weighted and the weight is higher close to the new instance and it's further away, and so and one can say that the kernel function moderates the original distance measure that's why we're looking at it. Regression of course is natural because we still aim at approximating a real valued function. And this function could be how any kind it could be linear, it could be quadratic etc. and you see in the picture below an illustration how this will look like where we have a certain point and then when we have some kind of weighted curve that determines the importance of the surrounding items. So a kernel function is a concept imported from nonparametric statistics and when you say kernel is a window function that it's defined over a certain window when the argument to the function is a distance measure one can say that the kernel function is a moderation of the original distance measure which means that the effect of the distance is moderated. A kernel is normally a non-negative real-valued integrable function and for most applications it's common to define the function, also to satisfy some other constraints, for example that the integral over the whole area beneath the function is 1, so the area is always 1 and also that the function is symmetric with respect to 0. And there are a lot of these functions and they are exemplified in this graph and I can say that for all of them apart from the uniform one, we can see the decreasing weight or importance for points or for distances further away from the center. I also want to mention another approach to instance-based learning there's that connect to what we talked about here kernel functions. So kernel functions are also useful in a kind of weighted instance based learners called Kernel methods. The kernel function here serves as a similarity function, this kind of method typical computer classification has a weighted sum of similarities, actually here with this approach the class label are modelled as +1 and numerically as plus 1 and minus 1, both for the class label of the training instances and for that output down labelled instance xq. So in this approach it is also assumed that there is a the weight allocated to his training example on that and you have this function k that measures a similarity between any pair of instances, and in this kind of approach this kind of function is called the kernel function. And so what we do is we with sum number of terms where term it is the weight of an instance a certain instance, the classification has certain instance which plus minus 1 and this kernel function between the target instance and the selected training instance. And then we take the sign of that sum and that sign and the sign of that sum becomes the class label of the target instance. So this is a pretty old method it was the first instance of it was invented many years ago in the 1960 and was termed the kernel person drop.(10:56)

Now we will switch to a few other but related topics as you may have already understood machine learning is a pretty complex field and there are many concepts many approaches many kinds of algorithms and there is no self-evident trivial characterization of all these things. So what I try to show you on this slide is how I want to talk about a few concepts in the in the rest of this lecture. So actually the focus for the rest of the lecture with something called a support vector machine. A support vector machine is a kind of scheme or type of algorithms which are an instance of something called binary linear classifier and essentially the common theme for all of these is to look at the instance space and try to find lines or surfaces that distinguish between different classes and for the word binary of course signifies could we talk about two classes yeah in the basic case. So this is a special kind of algorithm , however I find it natural to discuss it here because typically the support vector machines work in a work in an instance based learning fashion in the sense that this kind of algorithm does not build up explicit hypothesis structure. So rather it computes indirectly some kind of surfaces in the feature space which corresponds one can say to the hypothesis. So therefore support vector machine in my mind is very much inspired by instance-based learning and the case is though that support vector machine in its basic form handles only the linear case. So if we want this technology to be able to handle non-linear case, we need two answers of something and actually what we then do is that we the trick is to map the instance based on to another space. So that in the new space we create it's possible to handle it with linear methods and the technology for doing that is termed kernel methods. So that's logic of the rest of the lecture.

A few words about Binary linear classifiers. Such classifiers are both binary in the respect that they distinguish only between two categories or classes. They are linear in the respect that the classification is based on a linear function of the inputs. So each data item is characterized by a vector x some features 1 to n, refer to the features of it of in instances x as ai(x). Associate with each instance is also binary a valued class label c(x). So the goal of a binary linear class is to find a hyperplane that line or plane the separate or something else in higher dimensions and suppress the instances of the two categories. The property of the instance space required for such a hyperplane to be found is called linear separability. Obviously the linear classification can only handle linear separable cases. So you can see a very simple example of this to the right. The lender (15:54) case which is easy to find and applying in (15:52) two dimension and the case where the limiting area clearly is not linear. So techniques that can handle non-linear situations we call nonlinear classifier techniques.

So this slide simply shows another two other examples of hyperplanes. So I mean what's important to understand is that for an n-dimensional Euclidean space and hyperplane is of n minus 1 dimensional subsets. So this means, that for if we have a two dimensional space which we have to the left hyperplane is a line, if we have a three dimensional space the hyperplane is a plane and so on. It's difficult a little more tricky to illustrate how an hyper plane actually looks in that in higher dimensions. The support vector machine is a model for machine learning which operates in a supervised mode with pre classified examples. It can also brighten an incremental model. It is an instance of a non probabilistic binary linear classifier system where binary means that it classifies instances into two classes and linear means that the instance space have to be linear etcetera. A vector machine can be used to handle nonlinear problems, if the original instance space is transformed into a linear separable one. It can also manage your flexible representation of the class boundary, contains mechanist to handle overfitting, it has also a single global minima which can be found in polynomial time. So it has many nice properties, you easy to use, it often has a good generalization performance and the same algorithm solves a variety of problems with very little tuning.

The support vector machine or SVM performs binary linear classification by finding the optimal hyperplane that separates the two classes of instance. So it's a typical instance of a binary linear classifier. A hyperplane in an n-dimensional Euclidean space is a flat n minus 1 dimensional subset of that space, the divides the space into two disconnected part. The two-dimensional space the hyperplane is a line dividing the plane in two parts as an example. So new instance are mapped into the instance space and predicted to belong to one of the classes based on which side of the hyperplane there are position. One should observe here that the hyperplane is not directly stored or concretely stored as an hypothesis, it's rather computed on demand which is typical for instance based systems. The support vector machine can even full case only be applied to linearly separable instance spaces. So in the basic form it has limited coverage.

So I want to give you an informal outline on the support vector machine algorithm, so it works like follows. So a small subset of instances in the borderline region between the main instance space regions for the two classes are chosen as corner stones for the analysis. These instances are called “support vectors”. So as you understand these have also given name to the algorithm. So the SVM algorithm aims at maximizing the margins around the separating hyperplane we look for, the maximizing the distance between the target hyperplane and the

chosen support vectors. So as you can see in in the picture to the right depending on we select the hyperplane or line in this simple case, the margins becomes more or less broad. So the album's aims at finding the maximum margin. One can say that the optimal maximum margin hyperplane is fully specified by the chosen support vectors. Over there is an optimization problem had to be solved but this problem can be expressed as a quadratic programming problem that can be solved by standard methods.

SVM's are very straightforward to apply in linear cases, but we also want to look into how we can apply SVMs in nonlinear cases and also in the cases of non-feature vector data. So as the SVM only handles linearly several instance spaces with instance expressed in fixed length real value feature form, all problems that we want to approach SVM have to be transformed into such kind of linear separable spaces, typically most cases when we create a new space in order to achieve the separablity we want that space will typically have more dimensions than the space we start from. So the simplest case the most straightforward case is that we still have instances expressed as feature vectors but the instances are so configured that they are nonlinear separate, and this is where we will focus the rest of this discussion. The other problem which is also important in all cases is the problem of being able to apply a method design for handling feature vectors, for instance spaces where the instances are expressed in other forms. And I already illustrated for you an example of that in the context of the cosine distance measure similarity measure where one have to ,we have instances there are texts when I have to map those texts, the features of those elements of those texts into feature vectors. And this goes for other things it's so you can see to the right examples of complex molecular structures, if those are the row instances those also have to be transformed into a feature vector form in order to use this tiny standard kind of method. A key concept in our approach to transforming a nonlinear feature space to an equivalent linear space is to make the new space more high dimensional, a problem with that is that many times computations of instance coordinates in the high dimensional target space is often more complex and costly. Luckily enough the SVM algorithm only needs distance/similarity measure between instances position in a linear separable space. So this means if you create a new map or reduce space on to a new space which is linearly separable, that's of course a requirement, if we do that then we can see to that we define a similarity measure in that new space, such that the similarity measure is expressed in terms of the coordinates from the original space, which is more easy to compute. So to summarize we define a similarity measure that is applicable and meaningful for a new high dimensional space, the SVM algorithm can use those

similarity measure however the detailed calculation of the similarity measures are defined by a function which called the kernel function. So that they are expressed in terms of coordinates from the original space. So then of course is very much up to how we choose this function so that can work that way and the clever choice actually is to define the kernel function in terms of the inner product of the mappings of the coordinates in the original space. So this scheme is what is called the Kernel Trick.

Let's look at an example so consider a two-dimensional input space and a three-dimensional target space with the following feature mapping onto the linear separable three-dimensional space. To the right you can see what we're trying to do, we have a two dimensional space where we have instances that are not linearly separable what we now try to find is a mapping that map's the space into a three dimensional space, where the instances is more clearly linearly separated. So what we do is to issues a mapping, so actually we map at an instance x1, x2 on to three new coordinates, with the first coordinate in the three dimension is a square, and of the x coordinate in the first, and the second coordinates the second and the third is a square root of two times, one times larger so okay. So this is how we define it, so if we then look at two vectors x and z ,and we can consider them of course in both spaces and we look at the similarity function in the three-dimensional space which we choose to be the inner product of x and z in that space, and then we expand that through a normal calculation using the definition of the mapping, and then it turns out that when we done that calculation it turns out that becomes equivalent, of course depending on our initial choice of mapping. As the square of the inner product between the corresponding vectors in the two dimensions, so this means that we fulfilled what I've said in an earlier slide that we want to define a similarity measure in a higher dimensional space but still in being able to compute these distances in terms of the coordinates in the original space. So this is very simple an example of this can be achieved.

So apart from extending the use of support vector machines from the linear case to the nonlinear case which we now discuss extensively, it's also interesting to see how it can be extended to two other classification scenarios. So one natural case is to see whether we can extend the use of this technology from binary classification to multiple class problems and the strategy here is then to reduce the single multi-class problem in to multiple binary classification problems, because for each of the second week we can then apply this rehab technique. It's also possible but maybe no not as exactly a straight forward but possible to modify the algorithm, so it can handle regression keeping most of the key properties of the

algorithm. So this was the end of lecture 4.4, on instance-based learning. So thanks for your attention. The next lecture for point 4.5 will be on the topic cluster analysis, thank you