Transcripts (This lecture was given by Joakim using Abode Connect a few years ago).
Please, read carefully some remarks at the end of the transcripts!

Nearest Neighbor Classification

kNN is widely used learning algorithm. Based on an old idea:
1) recollection of an earlier case: you must store in memory
2) resemblance: if the symptoms are symilar, the diagnosis should be similars
So the key components are: storage of old instances AND similarity-based reasoning (if we have seen similar cases, we should have similar classes)

k-Nearest Neighbour

Some people might say that there is no learning here.
Here learning is basically storing all the training instances.
Classification, that's where the whole work is. The way we do this is:
Given a new instance x, what we have to do is to compare it with all the stored instances. We then compute the similarity or better the distance between the new instance and the stored instances. And then we keep track of the nearest k-closest instances, or the k-most similar instances. And these are what we call the nearest neighbours. And then we just assign to x the majority class (this happens in the simplest versions). If k is 3 and 2 of the nearest neighbours are red and one is blue, we say that the new instance is red.
This is very different of what we saw in DTs. Essentially this is a geometric learning, where we use some notion of proximity in space. And proximinty in this space (which is a feature space) tends to imply that we should have the same class. This is sometimes called the *smoothness assumption*. *Smooth in the sense that if we look at the different regions of the spaces, all the instances in a region should have the same class*. So basically instances will cluster in this space rathen than being randomly distributed. Of course this depends on having the right feature space. If we pick up the attributes/features that are not indicative and informative, this might happen. But we assume that our features are indicative and informative.

Eager and Lazy Learning

The k-neighbors are characterized by what is called lazy learning or memory-based learning
The opposite of lazy learning is eager learning.
DTs would be an example of **eager learning** where we have a learning phase *where we use the training data to induce a abstract model*. A model that exactly covers the exact training data. We abstract over the training data. It is true not only for DTs, but for most of the approaches in this course. When you create a DT and throw away the training data, you can no loger infer the training data, becase the same DT might model different training sets. At validation time we apply the model to new data, but we do no longer need the training data (this is abstraction).
In **lazy learning**, instead, there is no abstraction going on. Essentialy *it is a comparison of the new data with data stored in memory*. In that sense, a lazy learning algorithm retains all the information in the training set. This means that it can accomodate very

complex hypotheses. We tend to apply the smoothness hypothesis and make disjctions among regions. Each region is smooth internally, but regions are not connected among them. Some people claims that it is what we find naturally for natural languages: we have rules and then we have exceptions to rules, and then we have exceptions of exceptions, etc. so we end up with very complex spaces. Some have argued that the inductive bias for these learners are well suited for learning categories in Natural Language. A drawback of this approach is that classification can be very inefficient. Sorting an instance through a DT via a number of tests can be very efficient and fast in this respect since it is not dependent on how large the training set was (once we have a tree, it is usually very efficient [of course the size of the tree matters]). Lazy larning (at least the simple version) are inefficient because they have to search the whole training dataset anyway, and this can take time if the training dataset is large.

Dimensions of a *k*-NN Classifier

One other thing that we need to consider if we want to create a k-nn.
One is obviously the distance metrics or the converse, ie the similarity. This determines the layout of the space or the neighborhood
The other important thing is the parameter k. we look at the k-nearest neighbors. How many 1 neareste neighbor, or 2 or 3 or…
What is an appropriate value of k? Given that we fix the layout of the space, the k parameter allows us to determine what neighborhood we should consider and apply the smothness assumption too. The largest the value k, the less complex is the hypothesis

Distance Metric 1

Attention! "y" should be "z"
There is a number of distance metric.
One metric that has been widely used in NLP is the overlap distance where the big delta is the distance betw 2 instances. Bold x is an instance and bold z is another instance. This is defined in terms of the distance of features. If we have m features that classifies this, the simplest way is to measure the distance betw the feature values. This is the value x for the ith features and this is the value z for ith features. We assume that we have some simpler distance function, small delta, that can measure this and then we sum over all the features. The sum of the distance over all features define the distance betw 2 instances.
And then this small delta function can be defined in a variety of ways. If we have categorical features, then we are simply checking: do they have the same value or not? If x I equals z I the distance is 0 (if the part of speech is verb in both cases, the distance is 0). If they are different, the distance is 1. If we have categorical features, the sum over here will be only the number of mismatching features. Suppose we have 10 features, and we have identical value for 3 of them, then the distance will be 7. We get a penalty point for each feature that does not match.
If we also include numeric features, the standard approach is to compute the numeric difference. Ex: If a word has 5 characters and another word has 3 characters, then the distance will be 2. Then we normalize this by the maximum possible distance. If the longest word we have seen is 30 ch and the shorted word is one ch, then we divide by 29 and we get the normalized value.

Distance Metric 2

This is another way, widely used the so called modified value difference metric
Again, the distance betw 2 instances is the sum of distance over all features. In order to decide the distance over 2 features, we do not just look at whether they are identical or not, but we consider the conditional class probabilities, that we also looked at for decision trees.
So for each class we ask: what is the probability of a class or the percentage of instances that belong to a class GIVEN that we have this value for the feature?
Then we compare that with the same kind of probability for another feature. In the case of identical features, the distance will be zero. If they are different, we should consider the distribution of classes to be similar. If this probability is .6 and this probability is .4, then we get a distance of .2 here. Whereas if this probability is 1 and this other is 0, then we get the maximum distance of 1.
In this case we quantify the difference betw no identical features wrt the distribution of classes over instances. And then we do this over all classes. We look at all classes we have and then compare the possible conditional class probability for the 2 features. And this gives us a more fine grained distance metric.
And for some tasks this works very well, better than the simple overlap metric.

The *k* parameter

A way of looking at the k parameter is as a way of tuning the complexity of the hypothesis space.
There 2 extreme values. If we set k to 1, then every instance has its own neighborhood. We can get 0 training error, of course, because every distance decides its own classification. But it may not give good generalization because some of these might actually be noise.
On the other hand, if we set k to N (which is the size of the training set), then all the feature based on one neighborhood and the features will be classified with whatever happen to be the majority class. This is sometimes called the **majority baseline**, that is guessing the most frequent category. This is often used as baseline to see how the learning algorithm performs.
In the screen there are 3 distributions, 3 different data points (blue, green, red). On the LHS you see the regions you get if you set to 1 nearest neighbor. On the RHS, you see what you get if you set 15 nearest neighbors: here the regions are larger and smoother.
As for the green islands, it would be important to understand if they are linguistically important exceptions or noise of some kind. That's why tuning this parameter is very important. It might also be the k to 15 misses some important generalizations.
One way to find out is to measure the generalization error.

A Simple Example

Here is a very simple example.
The k parameter changes the hypothesis
Suppose we have 4 instances, we have 4 features and the capital letters are the classes
So the first class A etc.
How do we classify this new instance that has the features abba? It is not identical to any of the ones we have seen before

We use the simple overlap metric. The distance (capital delta) betw the first instance and the fifth (new) instance is 5 , etc.
We know the distances, but this still does not determine the classification, for which we have to fix the k value
Let k equal 1, the class we predict is B because the most single similar example belongs to class B.
If k=2 we get a tie. This is incidentally why we should avoid odd value, but we get ties any way if you get more than 2 classes, unless we have tie-breaking.
But if we go on to k= 3 and k=4, we converge to the class A
This is a made up example, so we do not know what is the real class, but it shows how to play around with parameters.

Further Variations on *k*-NN

There also additional dimensions that we can add to k-nn
In the overlap metric, all features have equal weight.
For ex, we can use feature weighting if we assume that some features are more important than others. We can add some weight with some entropy-based measures like IG and GR
We can also have a weighting voting scheme and we take the majority class of k neighbors. We might think that some instances are important than others, so instances that are closer in the space might be given more weight. The normal way to do this is to weigh the votes by some notion of inverse distance

Properties of k-NN

kNN like DTs can accomodate numerical and categorial features.
They can also cope with complex disjunctive descriptions.
They tolerant to noise.
They should be used when fast classification is not crucial (although there ways to make them faster, for ex through indexing)
The inductive bias is based on the smoothness assumption, so we think that the nearst neighor should have the same label.
We might assume that features are all equally important (this is different from DTs where we try to find the most informative features, and if we can ignore the other). kNN is more suitable when we have many feautures and are all mostly equally important.
In terms of complexity, we can tune the complexity using the k parameter
---end Joakim

REMARKS:

The k-nn classifier assign the input to the class having most examples among the k neighbors of the input. All the neighbours have equal vote, and the class having the maximum numbers of voters among the k neibours is chosen (Alpaydin, 2014: 193).

How to choosing the k parameter: Small values for k lead to complex decision boundary, increasing the number of nearest neighbour reduce the overfitting to the training dataset and results in smoothed decision boundaries (Rogers and Girolami, 2015: 182).

Att! The term "parametric" refers to parameters that define the distribution of the data. Since decision trees such as C4.5 don't make an assumption regarding the distribution of the data, they are nonparametric. Do you remember lect 3? The population is known and we do not make any assumption about the underlying population. The same is true for kNN. Decision Trees and K-Nearest Neighbors are called **non-parametric** methods. When a method makes assumptions about he underlying distribution, it is called **parametric**. For instance, the Naive Bayses classifiers assumes the population has a normal distribution.