# Certificate 2 Session 2 — Script

[slide 1]
Last session we built our first machine learning model. The bankruptcy dataset we were using was already given to us, and the features of the dataset were simple value encodings of *6* parameters: *1* means high value, *0* means average value, *-1* means low value.
Today we are going to work with *textual features*. How can we represent text in a machine-readable way, in the form of feature vectors? Which features can we use and how can we compute values for these features? In other words, we are going to enter the domain of Natural Language Processing, or NLP.

[slide 2]
And we will start with a concrete machine learning task called Language Identification: https://en.wikipedia.org/wiki/Language_identification
Suppose you have some text, a paragraph or just a phrase. The task is to name the language the text is written in. To make things a bit simpler, we are going to differentiate between three languages: Swedish, Norwegian and Dutch. All the three languages are from the same family (Norwegian and Swedish are North Germanic languages, and Dutch is West Germanic), so they are not too different from each other and not trivial to differentiate of you do not know them ( and hopefully you don't =). We are going to process relatively short phrases, which makes the task a bit more challenging since the information about the instance is limited. Here you can see an example of the input data: three phrases meaning "happy birthday" in the three languages.

[slide 3]
We have collected our own dataset of phrases from Wikitravel phrasebooks. You can download the dataset from github: https://github.com/smartypanty/AIgaming-course/blob/master/C2S2/phrases.txt. In total there are *970* phrases: *333* for Swedish, *330* for Norwegian and *307* for Dutch. The file is structured as follows: there are two type of lines that follow one another. The lines commented out by the % symbol contain translations into English, and they are followed by phrase lines. The phrase lines are separated by the ||| symbol into the phrases themselves and the labels, i.e., the languages codes *SWE* (for Swedish), *NOR* (for Norwegian) and *DUT* (for Dutch).

[slide 4]
In order to represent our data instances formally, i.e., to extract features from the foreign phrases, we need to learn about *language modelling.* Language models are probability distributions of certain sequences of text in a language, or rather, over some *corpus* of texts (collection of texts, in simple words). Sequences could be characters, character sequences, words, phrases etc. You can read more about language modelling here: https://en.wikipedia.org/wiki/Language_model.
Using a language model, we can predict what is the probability of an item *i* to appear in text (think in Norwegian vs Swedish text), and also what is the probability of several items to appear in text in a certain order. This brings us to one of the central notions of NLP — *ngrams*. An ngram (sometimes spelled n-gram) is a continuous sequence of *n* items appearing in text. The two most commonly used types of ngrams are character ngrams and word ngrams. Think about a window of length *n* that you move along a piece of text, collecting the content of the window every time you move it one step further. For example, if you are given the sentence "how to make an AI gaming bot", and a window of size *3*, you can get the following *word trigrams*: 'how to make', 'to make an', 'make an AI', 'an AI gaming', 'AI gaming bot'. In the similar fashion you can also collect character trigrams from the same phrase: 'how', 'ow ', 'w t', ' to', 'to ', 'o m', ' ma', 'mak', 'ake', 'ke ', 'e a', ' an', 'an ', 'n A', ' AI', 'AI ', 'I g', ' ga', 'gam', 'ami', 'min', 'ing', 'ng '. A technical note: when harvesting character ngrams and creating a character-based model, one can either keep or discard the spaces. If the spaces are kept, the ngrams are able to capture patterns of word order which might be of great use.

[slide 5]
In practice, people usually do not go beyond ngrams of length *5*. Surprisingly, these simple units can capture linguistic patterns pretty efficiently and on several levels. Consider a Swedish phrase “Grattis på födelsedagen” which means happy birthday. First of all, character ngrams can capture a probability distribution of different letters in a language, which comes very much in handy in case a language with unique symbols, e.g., a letter å. Also if the task was to differentiate Chinese from Japanese from Korean, which use different sets of hieroglyphs, character unigrams would be enough (but you might not even need ML for this task). Secondly, ngrams can capture typical words and word parts, e.g., roots of verbs and nouns, prepositions, which also include suffixes, prefixes and endings of words that reflex the morphology of the language etc. Lastly, if we include spaces into the analysis, character ngrams can capture typical sequences of words. For instance, on a corpus of English texts an ngram “ed by” would have a high probability, and this is because in English a verb in passive voice (which usually ends with *-ed*) is often followed by a preposition *by* (“caused by”, “divided by”, “created” by etc.)

[slide 6]
Now we are all equipped to perform feature extraction! Let’ start with extracting character ngrams. Our example string is “Kunt u mij dat tonen op de kaart” which means “Can you show me this on the map” in Dutch. We set the length of the ngrams in a separate variable *n*, which we set for *3* for now. Then by means of a *for*-loop you go through the sentence, from its first symbol until the symbol at position that is *n* steps away from the end (`for i in range(len(s) – n + 1)`), and builds an ngram of length 3 by means of slicing (`ngram = s[i:i+n]`, where n is *3*). Do not forget to save ngrams into the list of ngrams (`ngrams.append(ngram)`)!
You task is now to modify the code such that it generates all character ngrams *up to size n*, i.e., *1,2,…,n*. You can find the code in the *feature_extraction.py* file on github.

[slide 7]
What else can we do in language modelling? Ngrams are solid enough foundation for doing machine learning, but there is room for improvement:
(a) try running your classification model on a combined set of features, using both character and word ngrams;
(b) try playing with the size of the ngrams; does the performance of your model change when you go from *n=2* to *n=5*?
(c) do not forget to handle upper-case letters; do you want to differentiate “an” from “An”, or do you want it to be the same ngram?
(d) you might not need extra spaces that accidentally appeared in the text; for that you might want to use the `strip()` function over strings in Python;
(e) as in the case of upper case, you need to decide what to do punctuation signs, whether to keep them (or some of them) or not; on the one hand, full stops are signs of the end of a sentence, which might be a useful feature for some languages; on the other hand, commas might be pretty useless and might only create unnecessary noise in the data.

[slide 8]
As we discussed last session, there is a plethora of classification algorithms, even families of algorithms. Last time we were using k Nearest Neighbour, and today we are going to look at another classical supervised ML algorithm — *logistic regression*.