DAT410 - Assignment 3

Eric Johansson & Max Sonnelid

February 15, 2021

Group number: 70	
Module number: 3	
Eric Johansson	**Max Sonnelid**
970429-2854	960528-5379
MPDSC	MPDSC
johaeric@student.chalmers.se	sonneld@student.chalmers.se
35 hours spent	35 hours spent

We hereby declare that we have both actively participated in solving every exercise. All solutions are entirely our own work, without having taken part of other solutions.

Introduction

In this assignment, you will be investigating machine translation systems. You will be asked to read and reflect on an overview of different machine translation techniques, implement one of the most famous models for machine translation, and discuss some questions about the behavior of machine translation systems. This project has been executed by Eric Johansson and Max Sonnelid. Summaries of associated lectures toghether with personal reflections will be attached in the end of this document.

1 Reflections from article

1.1 Max Sonnelid

(a) As you can see, automatic translation has been one of the "holy grails" of AI research for several decades, and attempts at solutions have been proposed using many diverse approaches, based on radically different computational techniques. Can you think of other AI problems where we can see a similarly wide range of approaches?

An AI problem, which has got big attention in recent years, is playing the Chinese game Go. The game has similarities with chess, but the much bigger board makes the game an immense number of times more complex than chess and requires multi-dimensional strategic thinking. Previous rule-based attempts at creating Go-playing machines were able to beat amateurs, but not professional players. However, the company DeepMind with its AlphaGo machine eventually beat the one of the world's best Go players, Lee Sedol, in 2016. This was done by using a deep neural network technology, where AlphaGo was able to evolutionary improve its Go playing by reinforcement learning, both by matching moves of recorded, previous games and by playing games against other instances of itself. Similarly as the emergence of neural-based machine translation, the success of Go-playing machines was achieved when humans let the machine learn and encode the games by itself, compared to human programming of the logic of the game.

(b) Can you think of similarities between some rule-based translation systems and the state-of-the-art neural systems?

An early idea about machine translation, when RBMT was still dominating, was to be able to encode a text in one language to a so called interlingua (universal language), and then decode the interlingua text to another language. A strong advantage with this approach would be the ability to easily add more languages to this system and then enable translation between all possible combinations of languages. However, a rule-based, interlingual machine translation never succeeded.

Today, neural-based machine translating is the most successful method at a general-purpose-level. In its simplest form, it consists of one neuron that is responsible for encoding the text in one language to a set of specific, abstract features. Another neuron is responsible for decoding these features back into a text in another language. Basically, this is the same idea as in interlingual machine translation, where the interlingua is replaced with a set of specific, abstract features in the neural-based machine translation.

(c) Can you think of situations where it could be preferable to use an "old-school" rule-based solution instead of a modern (neural or statistical)?

As described in the article, some advantages with RBMT are morphological accuracy, reproducibility of results and ability to tune it to the subject area. A specific area where RBMT could still be applicable is translating medical information. For medical information it is not acceptable with incorrect translations as this could risk the health of people as well as lead to lawsuits against pharmaceutical companies. Still, a majority of the phrases in medical information are standardized and identical for several drugs. Using RBMT on these standardized and identical phrases, which have unambiguous translations in all of the training data, would save many hours of translators' time and thus reduce translating costs. Compared to e.g. using neural-based translation, an RBMT-based approach would always give the same translation to the same phrase, which is important in a pharmaceutical setting. Simultaneously, a RBMT-based system needs to excel at alerting the translator when there is an unknown phrase or a phrase with an ambiguous translation, which can then be translated manually by an expert.

1.2 Eric Johansson

(a) As you can see, automatic translation has been one of the "holy grails" of AI research for several decades, and attempts at solutions have been proposed using many diverse approaches, based on radically different computational techniques. Can you think of other AI problems where we can see a similarly wide range of approaches?

The first thing that comes to my mind when discussing "holy grail" of AI based solutions is in the financial world, where trading bots has been used since the eighties when Jim Simons started one of the first trading companies that completely lacked people with any financial background. Instead, the company built algorithms that detected patterns in different stocks using statistical models. As the company grew, their algorithms successively turned into black boxes, where no single person could determine why and how a decision was made. The company is still active and continues to use advanced algorithm that now trades completely by themselves. Since a lot of companies tries to earn money in the same way, there is a never ending strive for the optimal model and a lot of different approaches exists. A clear indication that the implementation of computers generated trades has been successful is that the companies biggest fund, Medallion, has the best return record in investing history. (**?**)

(b) Can you think of similarities between some rule-based translation systems and the state-of-the-art neural systems?

One thing that rule-based systems and neural systems have in common is that they both use grammatical rules in order to make accurate translation. The difference is that the rules are stated by linguists in the first case, whereas the neural model learns the grammatical rules by themselves. A more concrete example of similarities between the two models are that they are both using interligual translation, which could be described as all language can be translated into a meta-language which then can be translated into any other language. The clear benefit of this is that it is very easy to add a third language as well.

(c) Can you think of situations where it could be preferable to use an "old-school" rule-based solution instead of a modern (neural or statistical)?

Even though neural based translation models are the clear choice in most cases, I still think there exists occasions where the basic rule-based model can come of use. For example, lets say that we want to learn more about an ancient language that almost no alive person speak and lets assume that we have limited amount of data (texts) from the language. In this case, rule based could be a better option since it does not require as much data compared to modern approaches.

2 Implementing a Machine Translation System

This section will describe how the machine translation system was implemented by describing its individual parts step-by-step in the same order as the lab PM. Finally, strengths and weaknesses with the final translation model will be discussed.

2.1 Warmup

The initial task of this problem was to read the files. This was done by using `os` and by creating a method that reads the data and returns the file as a string. Since all words and special characters were separated by white space, the `split()` method was used, which returned an array where each element contains a word from the string. To find the most common words, a counter was created and then the method `most_common(10)` was used to get the ten most frequent words. The same procedure was repeated for all languages by looping through the different text files.

To calculate the probability of a certain word, we used the same method as above and then divided the occurrence of the word with the length of the array containing all the words. Here, we decided to combine all languages into a long array and calculate the probability using that array. The result is shown below:

Word	**Probability**
zebra	0.0
speaker	8.854e-6

2.2 Language modeling

This task was solved using three methods. The first method, `bigrams` takes a sentence as input and creates a two-dimensional array containing the bigrams from the sentence.

The second method called `probability` takes a bigram and a string as input. Then, with the help of `bigrams`, it creates all the bigrams from the string and calculates the probability of the bigram used as input by dividing the number of times the certain bigram occurs in the string divided by the number of times the first word of the bigram occurs in the string.

The third method, `finalPred` takes a sentence and a long text document (in this case notes from the European Parliament) as inputs. It then creates all bigrams from the sentence, and for each bigram, calculates the probability of this particular bigram with the `probability` method. By multiplying the probability for each bigram in a sentence, it was eventually possible to calculate the probability of a full sequence of words. In the cases where a bigram contains a word that is not present in the large text document or if the bigram itself it not present in the large text document, we decided to return a small number (10e-6) in order to prevent division by zero error and also to enable the final translation model to make predictions even if not all words in the sentence that are to be translated are present in the large text document.

2.3 Translation modeling

The first problem of this task was to enable comparison between an English sentence and the corresponding Swedish sentence. This was done by creating two arrays, one for each language, which takes a string of text as input and returns an array with the size corresponding to the number of sentences in the text document using `splitlines()`. These were later used as inputs in our EM algorithm together with the desired number of iterations.

To create our EM algorithm method, we based our idea on the pseudo code given in the assignment. Since there exists many loops in the algorithm and since each loop iterates through a large number of words, we wanted to use something that could handle words and probabilities in a more efficient way compared to regular lists. Hence, we used `defaultdict` from the `collections` library. This enables us to have words as keys, and when we wanted to store probabilities to a combination of a English and Swedish word, we could use a dict that had a English word as key and then returned another dict with the Swedish words as keys. This turned out to be a very efficient way of handling large set of words. The concept of dictionaries were used for all variables $(t(f|e), c(e, f), c(e))$ except for $\delta(k, i, j)$ which only needed to be a float since it was never stored. $c(e)$ were one-dimensional dicts whereas the rest were two-dimensional (since they depended on two languages). t (which is our target dictionary) was initialized with all probabilities set to 0.01. This could possibly have

been done in a different way (e.g., by setting the probability to $\frac{1}{nr\_of\_words\_in\_the\_string}$) but since our model managed to translate the words we tested and also yielded high probability for the correct word, we thought that this was the method that required the least calculations (since the number is fixed).

After running through the entire algorithm, our method return $t(f|e)$ in the form of a two-dimensional defaultdict.

Finally, we experimented with different numbers of T, i.e, number of iterations. The result is displayed below:

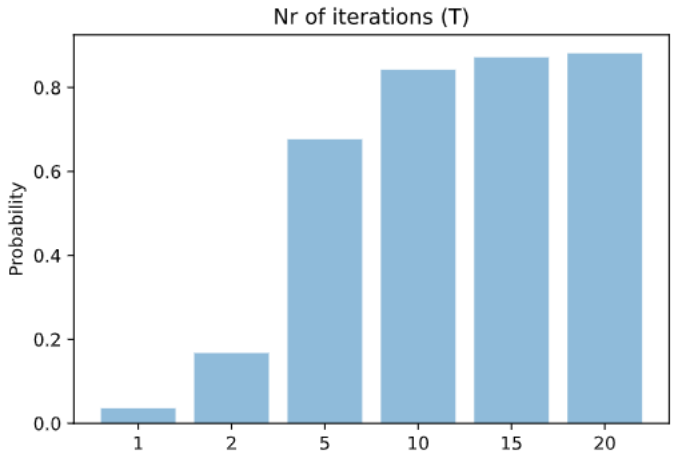

Figure 1: Probability of the most probable word using the translation model

By looking at the figure, one can clearly see the benefit of iterating through t multiple times. After only one iteration, the probability that the proposed word is the correct one is close to zero, whereas it is around 70% after five iterations. After that, the probability increases, but much more subtle than earlier. We stopped after 20 iterations, since the increase after that was insignificant. A conclusion that can be drawn out of this is the model yields better prediction if the initial t is better. This means that a very accurate method could be acquired if a lot of computing power were available (for example Google Translate).

2.4 Decoding

In order to connect both the language model and the translation model, we created a method called `translation`, which takes two input parameter: one sentence and one large string with sentences from the target language (used in `finalPred`). Three help methods were created for the `translation` method and below is described how they are connected.

`decoder` takes a sentence in the source language as input parameter and then splits this sentence into individual words. Each of the words from the source language are then translated by the `most_probable_translation` method, from the translation model created in step 3, and an array with all the translated words in the target language is returned.

`all_possible_sentences` takes the list of words in the target language (returned by `decoder`) as input parameters and uses the method `permutations` from the `itertools` library for creating all possible permutations of these words and thus form all possible sentences from the given words. All these possible sentences are then converted to strings and returned in an array.

`most_probable_sentence` takes the list of all possible sentences in the target language (returned by `all_possible_sentences`) as input. For each sentence, the method `finalPred` from the language model in step 2 is run. The sentence with the highest probability is returned together with its probability value, which is then used as the final output for `translation`.

2.5 Strengths and weaknesses

The biggest advantage with the model is that individual words are translated rather accurately by the language model. In most cases, the returned translations make sense. For example the words

"parliament", "dogs", and "friends" were translated into "parlamentet", "hundar", and "vänner", which are good translations. Another advantage is that the model always returns a sentence in the target language, which consists of individually correctly translated words. A third strength of our model is that it is very easy to translate into a new language. For example, to switch to France, we only have to read the file en_fr_file and use that to come up with a new target dict. However, it is important to note that one needs to have a file that is translated from English to France in order for this to work. Thus, this is not a model that translates into a interlingua (universal language).

A drawback is that the combination of many words to sentences mostly is incorrect and in many cases does not make any sense at all. One example of a good result is the English sentence "science is fun" which is translated to "vetenskap är roligt" which seems like an appropriate translation. However, when a longer sentence is translated, such as "this is a sentence from europe", it is translated into "från detta europa är en struktur". One can not expect the inflections to be correct, as the system has not implemented any functionality for this. However, the aim with the language model (step 2) is to return the most probable sentence and thus get a sentence with a correct word order. Apparently though, the language model does not succeed at this. A potential reason for this is that the data set is too small for being able to calculate accurate probabilities for sentences in the language model.

The biggest drawback with the final translation model is that the running time grows exponentially with the number of words in the sentence to be translated. This can mainly be explained by studying the formula for calculating the number of permutations (P), which is done in the method `all_possible_sentences`:

$$P = \frac{n!}{(n-r)!}$$

n represents here the total number of words and r represents the number of selected words. As the total number of words (n) is always equal to the number of selected words (r) when forming possible sentences, the denominator will always be 1 and the number of permutations will be equal to the faculty of the number of words in the sentence ($n!$). For example, "I love dogs" respectively "I love dogs with fluffy fur" thus have 6 respectively 720 permutations, which implies that `all_possible_sentences` will return 6 respectively 720 possible sentences with these input sentences.

The running time for "I love dogs with fluffy fur" compared to "I love dogs" will thus be much longer as the method `most_probable_sentence` will have to loop through and calculate the probability for 720 different sentences, compared to 6 sentences for "I love dogs".

For longer sentences, it is not reasonable to form every possible sentence from the translated words from the source sentence and then return the most probable of these. Instead, some more sophisticated method for forming a lower number of possible sentences must be created, which potentially could be done by, as a first attempt, try the exact word order from the source language for the translated sentence in the target language. If the probability for that translated sentence is above some kind of threshold, it is accepted as translation without the need of looking though all permutations. Another suggestion could be that longer sentences are divided into subordinate clauses and main clauses, which are regarded as separate sentences, which will lower the time complexity of the algorithm. Finally, it could also be possible to program grammatical rules that only sentences which follows the normal Subject-Verb-Object word order (normal word order in English) are created and other possible sentences disregarded.

3 Discussion

(a) Propose a number of different evaluation protocols for machine translation systems and discuss their advantages and disadvantages. What does it mean for a translation to be "good"? Minimally, you should think of one manual and one automatic procedure. (The point here is not that you should search the web but that you should try to come up with your own ideas.)

Manual procedure: One way of evaluating a translation model is to let several bilingual speakers count the number of perceived errors in translation. Then sum up all the errors for all senctences and all words. A faulty translated word yields a score of 1 and a correct word yields a score of 0.

$$totalScore = \sum_{s\_0}^{s\_i} \sum_{w\_0}^{w\_j} error$$

A similar way to approach the evaluation task could be to loop through each sentence and only look if the sentences as units are correctly translated.

$$totalScore = \sum_{s\_0}^{s\_i} error$$

Automatic procedure: An automatic way of approaching the task would be to use the first mentioned totalScore function, but compare each word with a thesaurus and thus automatically calculate the score.

A clear advantage of the automatic procedure is that it requires less man power and is much more time efficient. And the resulting score would probably be very similar to the one calculated manually. The optimal accuracy would probably be attained if the second version of the manual procedure were used. Here, it is up to the linguists to decide if each sentence is correctly translated. In this way, small error such as grammatical mistakes could be ignored (if grammar is not of high importance). And in the same fashion, grammatical error could be included in the verdict (if grammar is of high importance). Grammar is something that can not easily be picked up from methods that iterates through all words. Two drawbacks of the manual sentence method is that it is very time consuming, since it requires a human being to read all the translations. Moreover, human biases can lead to skewed results if multiple people are collaborating on scoring a translated text. Finally, there exist many challenges related to creating a automated scoring method that looks at one sentence at a time, one of them being that each sentence can be translated in multiple correct ways.

(b) The following example (Figure 2) shows a number of sentences automatically translated from Estonian into English. In Estonian, ta means either "he" or "she", depending on whom we're talking about. Please comment on the translated sentences: what do you think are the technical reasons we see this effect? Do you consider this to be a bug or a feature?

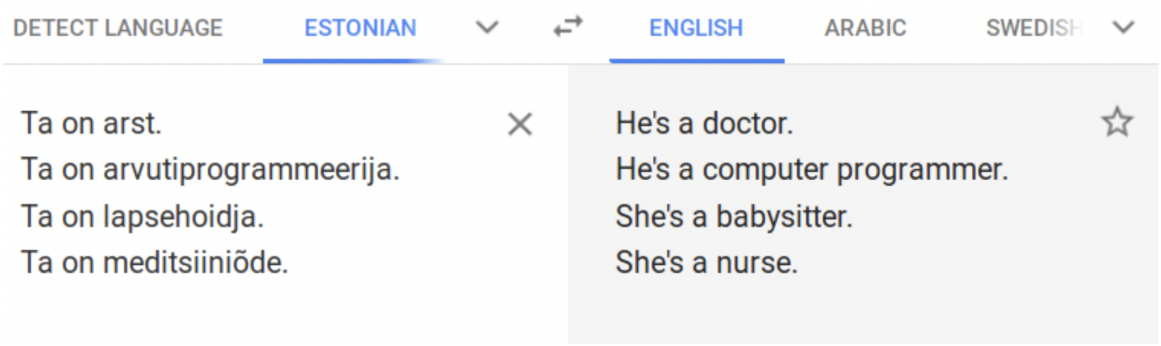

Figure 2:

The conclusion drawn from the example in the assignment PM is that the translation system tends to translate the Estonian gender-neutral pronoun "ta" to "he" when connected to traditionally male-dominated occupations, while it is translated to "she" when connected to traditionally female-dominated occupations. Because of the presence of male- respectively female-dominated occupations, the training data for the system can be expected to contain more sentences with for example "she" connected to "nurse" than "he" connected to "nurse".

From a probabilistic point of view the translation system has made a good decision when returning the translation "She's a nurse" as it, without further information, is more probable than "He's a nurse". However, such a probability-based decision reproduces stereotypes that many people wish to reduce and may cause annoyance.

In reality, this is however probably not causing any major issues as a sentence as "Ta on meditsiiniõde" very probably is included in a bigger context, where other gender-revealing details could be found and be used by the translating systems for returning the appropriate gender noun.

(c) Below, in Figure 3, we consider three sentences that include the English word bat and their automatic translation into Swedish by Google Translate. In the first example, bat is referring to a club for hitting a ball used in e.g. cricket or baseball, while in the second and third examples we are referring to a flying mammal.

The first two examples are translated correctly into Swedish, while the third translation is nonsensical: the automatic translation system seems to have come up with some sort of mix between the two

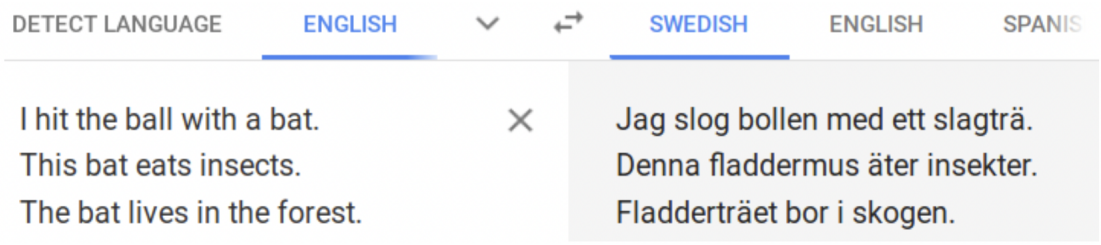

Figure 3:

Swedish translations of the word bat (the first half of the word Fladderträet comes from the flying animal, and the second from the baseball or cricket bat).

Why do you think the translation system has been able to select the correct translation of bat in the first two cases? What might be the reason that it has invented a new nonsense word in the third case?

The first sentence, "I hit the ball with a bat", probably only occurs in sports-related contexts and the translation systems have probably been exposed to numerous sports-related texts in both English and Swedish containing this sentence. In the same way, the second sentence, "This bat eats insects", is probably only found in animal-related contexts. From a probabilistic point of view, it is hard to translate "bat" incorrectly to Swedish in these cases as the context of both sentences are very clear.

However, in the third sentence, "The bat lives in the forest", the translation of "bat" resulted in an incorrect, made-up Swedish word. As can be read in the article reviewed in this assignment, some machine translators have the ability to decompose rare words and add translated word components together in order to form new words, which is very useful when e.g. translating a rare word as "Nordsjöoljeplattformsarbetare" to "North Sea oil platform worker". This feature has probably been activated on this sentence and in some strange way combined the both possible Swedish translations of "bat" into a made-up word. A possible reason for this could be that the probability to find "träet" and "skogen" in the same sentence is rather big, and that "fladder" was kept as the first word element instead of "slag" as "slagträ" is unlikely to be a predicate to the verb "bor".

4 Summary of lectures

4.1 Max Sonnelid

4.1.1 Lecture 4 - Natural Language Processing

- The main themes in this lecture are *natural language processing* (NLP) and *machine translation*, which is also the theme for the assignment this week.
- NLP is a central task in AI research and applications, and many philosophical questions in AI involves language understanding (e.g. Turing test and Chinese room).
- Common specific NLP tasks are for example *categorization tasks* (e.g. spam filters, topic classification), *tagging/sequence labeling tasks* (e.g. tag specific words for topic), *parsing/structured prediction* (identify relations between words) and *generation* (e.g. translation).
- Since the late 1980s, NLP research started to focus on data-driven techniques. Firstly, *probabilistic linear models* were mainly used, but today *neural networks* is the main technology for NLP.
- Neural networks are efficient and widely applicable in many NLP tasks, but two major drawbacks are the long training times and inability to explain the intermediate steps in the model.
- *Machine translation* is one of the most common NLP tasks and is defined as given a text in a source language, a text in a target language is generated.
- Neural machine translation systems are mainly based on a *encode-decoder architecture*, where the encoder summarizes the information in the source sentence, while the decoder generates the target-language output in a step-by-step method

- Another important architecture in machine translation is the division into a *language model* and a *translation model*, where the language model assigns a probability of the next word based on a sequence of words, while the translation model aligns a word in the source language with a word in the target language.

4.1.2 Reflection lecture

- *Distributional shift* is the phenomenon when the distribution differs between the training data and the test data, which might result in that the classifier trained on the training data might not perform particularly well on the test data. It is important to not by default assume that the training data and the test data share the same distribution.
- A *non-stationary distribution* caused by such a distributional shift can for example be identified to depend on a time parameter (this is often the case when data changes with the current season), which could be derived by comparing data over time.
- Changes in AI systems over time occur for two main reasons: (1) *changes to the model pipeline* (e.g. objective and hyperparameters), and (2) *changes to training data* (e.g. new customers). When versioning ML models, it is therefore important to version both training data, preprocessing pipeline and the hyperparameters beside the actual model code in order to keep the model *reproducible*.
- When developing AI systems, it is very frequent with changes in data and there are two main approaches for versioning the data: (1) *maintain releases* by regularly releasing new versions of the data and only train models on these releases, and (2) *maintain changes* by storing log queries executed on a database and only train models based on specific commits.
- An important concept when testing software is *unit testing*, which means that one does not test the whole software at a time, but only selected parts of it. This can be problematic with ML software as most parts are heavily integrated, but one solution approach is *testing gradient values* to model parts.
- Currently, big investments in AI hardware are focused on *GPU computation* because of the current rapid development of *deep learning* models. These models are mainly based on matrix multiplication, which is well-handled by GPUs. A strong advantage with deep learning is that such models may scale arbitrarily as it is a rather trivial task to create larger models.

4.2 Eric Johansson

4.2.1 Lecture 4 - Natural Language Processing

Natural language processing has been central in AI for a very long time (even at the time when Turing test were invented). There are many ways of using NLP, such as spam filters or spell checkers and grammars to machine translation system. Categorization is a common area where NLP is used. One example is - given a certain text, one or many classifications are to be made. The biggest difference between language data and other types are they often follow a power law distribution, i.e., most of the worlds are spotted very rarely. This means that it is hard to train our model.

Implementation of NLP Since the beginning of NLP history, there has been many different trends ranging from precursors in speech (before 1990) to neural models (after 2015). Another difference between machine learning in the beginning of 2000 and now is that the previous approach consisted of small steps that the creators understood, whereas it is common to use an advance neural network to solve the task directly in a black-box manner. Even though this seems disadvantageous doing this leads to better models that can solve entire problems by themselves. With this said, there still exists some drawbacks such as that new models are very complex and thus needs a lot of computing power to use (training one model made by google is equivalent to the entire lifetime of five cars with regards to CO2e emissions).

Fundamental idea of neural machine translation systems A neural machine translation system can be split up in two parts. The first part consists of an *encoder* which takes language as inputs and outputs vectors. Then, there is a *decoder* that generates output. One application where a system like this is useful is in language translation models.

Language- and translation models A common approach to translate word or sentences is to use Language- and translation models. Given English sentences, we should denote probability of what

the next word is. This could be further elaborated using the chain rule. However, when analyzing long sentences almost all words will have a probability of 0 (since there is a low probability that an identical sentence has been spotted previously). To get around this problem, Markov assumptions is commonly used. A Markov process is random process where the next action depends on the current state only. This a called a bigram language model. In this way, the model predicts the probability of a certain word only using the last spotted word in a sentence.

A translation model uses word alignments, which tags each English word with a French (if we are supposed to translate between these languages). In this model, there often exist a chicken-and-egg problem, regarding calculating the probability of a word in one language given another word in a second language. This can be solved using EM (expectation, maximization).

4.2.2 Summary of reflections lecture

5 Reflection on the previous module

5.1 Max Sonnelid

Our final implementation of the KNN-classifier gave very similar accuracy scores as the KNN-classifier in the Scikit-learn library. As the implementation in Scikit-learn can be expected to be optimal, reaching a similar accuracy shows that our implementation is well-functioning. It was a strong advantage in this task to have such a clear benchmark level for the performance of our implementation, which is probably not always the case in a real-life situation.

The biggest drawback of our implementation was its low time efficiency. Our implementation took around 25 minutes make predictions on the whole dataset, while Scikit-learn's KNN-classifier only took a few seconds to perform the same task. This shows the importance of not only providing a correct answer, but also to be able to provide that answer as efficient as possible. The low speed severely affected our ability to use and experiment with the model as we for each run had to think through very thoroughly what we wanted the model to do. In the following assignments, it will be important for me to consider the algorithm efficiency when implementing new projects and always try to minimize the number of loops and calculations. Especially in a real-life setting, clients do not accept slow programs.

Finally, it was interesting to be able to reflect about what evaluation metrics to use. Our first thought was to only look at the accuracy score. However, when we noticed that the DummyClassifier reached a higher accuracy score than Scikit-learn's KNN-classifier, we had to re-think about the evaluation metrics. Instead, we used a confusion matrix to be able to evaluate how good the model was at performing the actual task: correctly predicting dangerous pollution levels (1s). In future assignments, it will be important to keep in mind that there is no ultimate evaluation metrics. The choice of evaluation metrics depends on the objective of the task.

5.2 Eric Johansson

The purpose of the third assignment of this course was to learn more about a certain classifier, the k-NN classifier, by building it from scratch in the same manner as the one made by Scikit-learn. We have used the classifier several times before this and had a brief understanding about how it classified data points. With that said, we did not have enough knowledge to build a model ourselves right off the bat. After some research, we started to get an idea of how to proceed to create a proper model. This was challenging and at the same time very interesting. After multiple attempt, we managed to create a model with nearly identical accuracy as the one provided by Scikit-learn. However, the performance did not matched Scikit-learn's model with respect to running time. Our model generated predictions on the entire data set in 12 minutes, compared to a tenth of a second. In previous modules and courses, running time has never been an issue, due to small data sets or uncomplicated models. This time, computing power was the major bottle neck for us to develop our model. We really tried to understand what made the model so slow, but did not have time to figure it out.

Another lesson was the importance of building a small model and iteratively expand it, and not try to build a model that satisfies all the constraint all at once. This will be something that I will try to implement in the upcoming projects, as well as in real life scenarios.

Lastly, we both got a new perspective on how accuracy sometimes can be an insufficient way of measuring the performance of a prediction model. In this case we got an accuracy of around 70% with our model, compared to over 90% when using a dummyClassifier that predicts the same thing for all data points. Therefore, something like a confusion matrix could be useful in order to get a better understanding of the performance of a model.