DAT410: Group 60 Assignment 4

Cecilia Nyberg
Personal number: 990106
Program: MPDSC
Hours spent: 28
`cecnyb@chalmers.se`

Elin Stiebe
Personal number: 000210
Program: MPDSC
Hours spent: 28
`elinsti@chalmers.se`

February 20, 2024

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

1 Reading and reflection

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?

Image recognition is one such field with many approaches. Boesch, 2023 mention four main strategies of recognizing objects in images. SVMs, Bag of Features Models, Viola-Jones Algorithm, and deep learning models like CNNs. The most striking similarity between language translation and image recognition is that both applications try to match one item to another, a word or sentence in translation, and an object in image recognition.

Second, document classification is another area with a wide range of implementations. The text can be extracted to features in several ways such as using vectorizers or sentence transformers. Further, the extracted features can be utilized using a wide range of ML models such as k-clustering or using random forests. It can also be done using only statistics, such as collapsed Gibbs sampling for LDA.

Third, generating answers to user questions. This AI problem exists in many domains and levels of scope. Many companies use it in customer support and a version of this can be seen in ChatGPT.

Implementations can range from rule-based to neural and statistical methods. A similarity for this is that there has to be both an encoder and decoder part, first the question has to be understood, and then a answer has to be given.

Lastly a note on another scope of an AI problem with a wide range of solutions. Something that has gained a lot of interest in recent years is interpretable ML. ML might be used to generate a text description of a certain outcome or to visualize the result. The method of creating interpretation needs to be varied depending on the context and the user and thus, there are a large amount of solutions to creating interpretable systems.

After reviewing all the cases above, translation is still the area with the largest amount of solutions. This is likely due to the massive interest in translations described in “A history of machine translation from the Cold War to deep learning”, 2018. Translations are needed within every business and in people’s everyday lives. It will be interesting to see which ML fields will come close to the size of ML translation in the future.

b)

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

One similarity brought up in “A history of machine translation from the Cold War to deep learning”, 2018 is how there is a middle step of the translation. This middle step is visualized in figure 1. “A history of machine translation from the Cold War to deep learning”, 2018 writes that both neural systems and the Interlingua method use one model to encode the first text’s features and then have another model decode the encoded features independently.

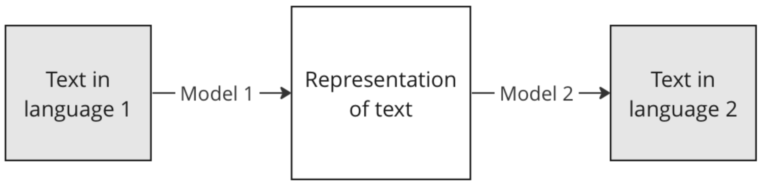

Figure 1: Two models are used to translate between languages.

Another similarity is related to the fields in which the models are used. Both rule-based and neural methods need to be trained and developed with their context in mind. For example, what format the output should be in and how the features should be preprocessed to work most efficiently. Both systems must also consider the changing world they’re part of. Their contexts might change and their performance will decrease if they do not evolve with it.

Finally, both models need to handle ambiguity. Rule-based systems demand rules to handle ambivalent data. On the other hand, neural networks mitigate ambiguity by generalizing over big amounts of data. Professionals implementing rule-based systems need to be aware of ambiguities and actively decide how they should be handled, while neural systems generalize without much human input.

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)?

Neural and statistical solutions are not always a better choice than rule-based ones. First, neural networks can become extremely big and complex. This is not something that all applications demand. High complexity should never be introduced if it is unnecessary to do so, thus a rule-based system might be enough in many cases with a low complexity of the issue at hand. Moreover, the company implementing the system must consider its capabilities and resources. According to “Choosing between a rule-based vs. machine learning system | TechTarget”, n.d., a neural or statistical system may demand a team of data scientists and thus has a higher demand for resources.

Second, the interpretability of rule-based systems is another reason to choose them before neural or statistical methods. Rule-based methods are easy to decipher and explain to people, including to those outside of the ML community. Transparency also has several other advantages such as giving a clear view of biases a system might have.

Third, a rule-based system might be necessary within some languages where there is less data to build the models on. Both neural and statistical methods demand large amounts of high-quality data which may not be accessible in all languages. In these instances, a rule-based system would be preferable.

2 Implementation

Now to the implementation of the IBM model 1. The translation algorithm is built step by step in each sub-chapter. According to Collins, n.d. IBM model 1 is a statistical method that aims to calculate the alignment probabilities between words in different languages. The best translation is found by using the relationship in equation 1. Here, E is an English sentence, and F a French one.

$$E^* = \text{arg max } P(E) * P(F|E) \tag{1}$$

P(E) is called a language model and P(F|E) is called the translation model and gives the probability of a French sentence given an English one. Collins, n.d. writes that this can seem to be "backward" though explains that it is retrieved using the Bayes rule.

a) Warmup

As a warmup, write code to collect statistics about word frequencies in the two languages. Print the 10 most frequent words in each language. If you're working with Python, using a Counter Links to an external site. is probably the easiest solution. Let's assume that we pick a word completely randomly from the European parliament proceedings. According to your estimate, what is the probability that it is speaker? What is the probability that it is zebra?

First, each language's top 10 most frequently used words are printed. The result can be seen in table 1. There were four files with English translations, they were concatenated and the most common words in English thus represent the most common words in the union of the four documents.

Rank	**English**	**French**	**German**	**Swedish**
1	the	die	att	de
2	of	der	och	la
3	to	und	i	et
4	and	in	det	l
5	in	zu	som	le
6	is	den	för	les
7	that	wir	av	à
8	a	daß	är	des
9	we	ich	en	que
10	this	das	vi	d

Table 1: Top 10 Most Common Words in Different Languages.

Now, if we pick a word at random, what is the probability of getting "speaker"? What about "zebra"? A method was created to get a better understanding of the probability of a word overall or in a specific language. The result for these two words can be seen below.

```
"Speaker" all languages 2.1349591802274315e-05
"Speaker" in English 4.23327120259538e-05
"Zebra" all languages 0.0
"Zebra" in English 0.0
```

Now to a very common English word. What is the probability of "The" in all languages, in English and German? Not so surprisingly, the probability of "The" occurring is highest in English, though it also occurs in other languages. Looking at the input data for the three languages it becomes evident that all instances of "the" come from English statements and are not a word included in the languages.

```
"The" all languages 0.038048854335616875
"The" in English 0.07541636787896436
"The" in French 4.799909238079862e-05
"The" in German 2.9430065755175486e-05
"The" in Swedish 3.927198467519887e-05
```

b) Language Modelling

We will now define a language model that lets us compute probabilities for individual English sentences. Implement a bigram language model as described in the lecture, and use it to compute the probability of a short sentence. What happens if you try to compute the probability of a sentence that contains a word that did not appear in the training texts? And what happens if your sentence is very long (e.g. 100 words or more)? Optionally, change your code so that it can handle these challenges.

A language model refers to P(E) which was discussed in section 2, it is the probability of a sentence within a certain language. Listing 1 demonstrates the implemented bigram and probability of a sentence model. The "count_bigrams" method shown in listing 1 calculates the probability of a certain bigram in the same way as shown in equation 2.

$$P_{MLE}(w_n|w_{n-1}) = \frac{\text{count}(w_{n-1}, w_n)}{\text{count}(w_{n-1})} \tag{2}$$

The method "probability_of_sentence" uses bigram to calculate the probability of a certain sentence. This is done the same way as shown in equation 3. The implemented strategy to calculate the probability of a sentence uses the Markov assumption, meaning the assumption that the next word only depends on the current word. In the implementations case <START> is denoted as NULL.

$$P(w_1, \ldots, w_n) = P(w_n|w_{n-1}) \cdot \ldots \cdot P(w_2|w_1) \cdot P(w_1|\text{<START>}) \tag{3}$$

```
 1 #implement a bigram model
 2 def count_bigrams(language_list):
 3     bigram_counter = Counter()
 4 
 5     for sentence in language_list:
 6         # Split sentence into words
 7         words = sentence.split(' ')
 8         words = [word.lower() for word in words]
 9 
10         # Update Counter with words
11         bigrams = [(words[i], words[i+1]) for i in range(len(words)-1)]
12         bigram_counter.update(bigrams)
13 
14     return bigram_counter
15 
16 # nr such bigrams/ all other bigrams starting with the same word
17 def probability_of_bigram_in_language(bigram, bigram_counter, word_counter):
18     return bigram_counter[bigram] / word_counter[bigram[0]] if
19         word_counter[bigram[0]] > 0 else 0
20 
21 #probability of a sentence is the product of the probabilities of the
22 bigrams and the probability of the word itself
23 def probability_of_sentence(sentence, bigram_counter, word_counter):
24     words = sentence.split()
25     words = [word.lower() for word in words]
26     bigrams = [(words[i], words[i+1]) for i in range(len(words)-1)]
27     probabilities = []
28     #probabilities.append(probability_of_word(words[0], word_counter))
29     for bigram in bigrams:
30         # If the word doesn't exist yet, we assume a very low probability
31         prob = probability_of_bigram_in_language(bigram, bigram_counter,
32             word_counter)
33         #probability_of_word(bigram[1], [word_counter]) is not included
34         probabilities.append(prob if prob > 0.000000001 else 0.000001)
35 
36     return np.prod(probabilities)
```

Listing 1: Implementation of a bigram model.

There are two potential problems when implementing a bigram model. First, if a new word is introduced the probability of such a sentence is 0. The implementation was therefore revised to return a very small probability of such a sentence, see line 34 to see the exact code. So if $P(w_n|w_{n-1}) = 0$, or is very close to zero (< 0.000000001) it is replaced with 0.000001. This change was implemented since there exist many words that were not represented in the training data. The decision to replace probabilities of 0 with a small probability was made since it could be too harsh to say that these words can never occur. Table 2 displays some sentences and their probabilities. One of these, "The speaker is speaking Gobbledygook" has a word that has not occurred in the training data ("Gobbledygook") and therefore receives a low probability but not 0.

Sentence	**Language**	**Probability**
"The speaker is speaking."	English	3.26×10^{-18}
"The speaker is speaking."	German	1.00×10^{-18}
"Of the speaker"	English	5.10×10^{-11}
"The speaker is speaking Gobbledygook"	English	3.26×10^{-24}
Long sentence in English	English	0.0

Table 2: Probabilities of a few sentences.

A second problem that might occur is that there is an issue with long sentences. The longer the sentence the lower the probability of it. As displayed in table 2, the probability of the last sentence, which was a very long sentence ($>$ 100 words), is 0. This was not something that was fixed in the implementation but discussed. Chiusano, 2022 writes about one way to deal with this problem; to normalize probabilities over sentence length by calculating the perplexity. Additionally, an end-of-sentence token could be introduced, similar to the NULL token at the beginning of sentences. This could result in shorter sentences not always having a lower probability than longer ones.

c) Translation Modelling

We will now estimate the parameters of the translation model P(f|e). Self-check: if our goal is to translate from some language into English, why does our conditional probability seem to be written backward? Why don't we estimate P(e|f) instead? Write code that implements the estimation algorithm for IBM model 1. Then print, for either Swedish, German, or French, the 10 words that the English word European is most likely to be translated into, according to your estimate. It can be interesting to look at this list of 10 words and see how it changes during the EM iterations.

Now to the translation model, P(f|e) which was described in section 2. The reason that the conditional translation probability seems to be written backward is because of the way that the most probable translated sentence is calculated. This is shown in equation 4. This is done for two reasons; only one language is needed as training data, and, there is a division of labor since the first part P(E) can handle fluency and P(F|E) can take care of the content.

$$E^* = \arg\max_E P(E|F) = \arg\max_E P(E)P(F|E) \tag{4}$$

Now to some definitions before the implementation begins. There are two counting variables seen in equations 5 and 6.

$$c(e) : \text{Count of occurrences of the word e in the English data.} \tag{5}$$

$$c(e, f) : \text{Count of every time a specific English and French word are aligned.} \tag{6}$$

Equation 7 is defined by Collins, n.d. as the conditional probability of generating the French word f from the English word E.

$$t_{ML} = \frac{c(e, f)}{c(e)}. \tag{7}$$

Delta is defined as equation 8 by Collins, n.d. for model 1. Its specific look is different from model 2 due to its prior distribution.

$$\delta = \frac{t_{ML}((f_i)^k|(e_j)^k)}{\sum_{j=0}^{l_k} t_{ML}(f_i^k|e_j^k)}. \tag{8}$$

The full implementation is shown in listing 2.

```
# Give t values only to those g and e which are in the same indexed sentences
# Initialize t(f|e) randomly
def initialize_parameters(source_list, target_list):
    t_parameters = {}
    for i in range(len(source_list)):
        sentence1 = source_list[i].split(' ')
        sentence2 = target_list[i].split(' ')
        for word1 in sentence1:
            if word1 not in t_parameters:
                t_parameters[word1] = {}
            for word2 in sentence2:
                if word2 not in t_parameters[word1]:
                    t_parameters[word1][word2] = np.random.rand()
    return t_parameters

#estimation algorithm for IBM model 1
def EM(list_language1, list_language2):

    #initialize t(e|f) uniformly
    t = initialize_parameters(list_language1, list_language2)
    print('initialization done')

    # 10 iterations of the EM algorithm
    for i in range(10):
        #set count(e|f) to 0 for all e,f
        count = {}
        for j in range(len(list_language1)):
            sentence1 = list_language1[j].split(' ')
            sentence2 = list_language2[j].split(' ')
            for word1 in sentence1:
                if word1 not in count:
                    count[word1] = {}
                for word2 in sentence2:
                    count[word1][word2] = 0

        #set count(f) to 0 for all f
        total = {}
        for sentence2 in list_language2:
            for word2 in sentence2.split(' '):
                total[word2] = 0

        #for all sentence pairs (e_s,f_s)
        for j in range(len(list_language1)):
            sentence1 = list_language1[j].split(' ')
            sentence2 = list_language2[j].split(' ')
            #for all words e in e_s
            for word1 in sentence1:
                denominator = 0
                temp_counts = {}
                #for all words f in f_s
                for word2 in sentence2:
                    denominator += t[word1][word2]
                    temp_counts[word2] = t[word1][word2]

                if denominator > 0:
                    for word2 in sentence2:
                        count[word1][word2] += temp_counts[word2]/denominator
                        total[word2] += temp_counts[word2]/denominator
        #for all f in F
        for j in range(len(list_language1)):
            sentence1 = list_language1[j].split(' ')
            sentence2 = list_language2[j].split(' ')
            #for all e in E
            for word1 in sentence1:
                #t(e|f) = count(e|f)/total(f)
                for word2 in sentence2:
                    if total[word2] > 0:
                        t[word1][word2] = count[word1][word2]/total[word2]

        #print for german the ten words 'european' is most likely to be
        #translated to
        word_to_translate = 'european'
        if word_to_translate in t:
            # Sort the target dictionary items based on values in descending order
            sorted_target_items = sorted(t[word_to_translate].items(),
                key=lambda x: x[1], reverse=True)

            # Print the top 10 words
            print(f"{word_to_translate}: {sorted_target_items[:10]}")
        else:
            print(f"{word_to_translate} not found in the dictionary.")
        print('iteration', i, 'done')
    return t
```

Listing 2: Implementation of the Em algorithm.

The implemented EM algorithm was run for 10 iterations. This could have been done differently, for example by checking for conversion by calculation when the difference between current probabilities and updated probabilities were small enough. This change would both allow for a higher amount of iterations if needed but also smaller which would decrease the complexity of the model.

The ten most likely words to translate European to were calculated and printed in each iteration. These are shown in table 3. It can be seen that the translation finds increasingly accurate words and becomes increasingly sure of the words throughout the iterations. The correct word "Europäischer" is found already in iteration 2. For this specific word, very few iterations were needed in order to find the correct translation, though the model builds its confidence in its prediction as the iterations continue.

Iteration	Top Translations
1	'zwei-klassen-gesellschaft', 0.18 | 'fbi', 0.17 | 'flugsicherung', 0.16 | 'grenzenlos', 0.15 | 'csu-europaabgeordneten', 0.15 | 'aufrechterhalten', 0.15 | 'funktionsfähigkeit', 0.14 | 'zentraler', 0.14 | 'kulturraumes', 0.13 | 'botschaften', 0.13
2	'europäischer', 0.18 | 'mitteleuropas', 0.18 | 'funktionsfähigkeit', 0.16 | 'nordeuropäischer', 0.15 | 'fbi', 0.15 | 'zentraler', 0.14 | 'aufrechterhalten', 0.14 | 'grenzenlos', 0.14 | 'rechtshängig', 0.13 | 'flugsicherung', 0.13
3	'europäischer', 0.40 | 'europäisches', 0.24 | 'europäischen', 0.23 | 'mitteleuropas', 0.23 | 'nordeuropäischer', 0.20 | 'bekennen', 0.18 | 'union', 0.18 | 'totale', 0.17 | 'europäische', 0.17 | 'europawahlen', 0.17
4	'europäischer', 0.57 | 'europäisches', 0.44 | 'europäischen', 0.44 | 'europäische', 0.44 | 'mitteleuropas', 0.27 | 'nordeuropäischer', 0.25 | 'europawahlen', 0.25 | 'totale', 0.25 | 'bekennen', 0.21 | 'union', 0.21
5	'europäische', 0.68 | 'europäische', 0.65 | 'europäischen', 0.62 | 'europäisches', 0.59 | 'europäisch', 0.43 | 'europawahlen', 0.37 | 'totale', 0.35 | 'nordeuropäischer', 0.31 | 'mitteleuropas', 0.30 | 'europäischsten', 0.27
6	'europäische', 0.77 | 'europäischen', 0.80 | 'europäischer', 0.79 | 'europäisches', 0.68 | 'europäisch', 0.50 | 'europawahlen', 0.40 | 'totale', 0.37 | 'nordeuropäischer', 0.32 | 'mitteleuropas', 0.30 | 'europäischsten', 0.29
7	'europäische', 0.84 | 'europäischen', 0.85 | 'europäischer', 0.82 | 'europäisches', 0.75 | 'europäisch', 0.55 | 'europawahlen', 0.41 | 'totale', 0.36 | 'nordeuropäischer', 0.32 | 'europäischsten', 0.30 | 'mitteleuropas', 0.30
8	'europäische', 0.91 | 'europäischen', 0.88 | 'europäischer', 0.83 | 'europäisches', 0.77 | 'europäisch', 0.57 | 'europawahlen', 0.42 | 'totale', 0.35 | 'nordeuropäischer', 0.33 | 'europäischsten', 0.31 | 'c5-0121', 0.30
9	'europäische', 0.93 | 'europäischen', 0.90 | 'europäischer', 0.84 | 'europäisches', 0.78 | 'europäisch', 0.59 | 'europawahlen', 0.42 | 'totale', 0.33 | 'nordeuropäischer', 0.33 | 'unionsbürgerschaft', 0.33 | 'c5-0121', 0.32
10	'europäische', 0.93 | 'europäischen', 0.90 | 'europäischer', 0.84 | 'europäisches', 0.77 | 'europäisch', 0.59 | 'europawahlen', 0.42 | 'totale', 0.33 | 'nordeuropäischer', 0.33 | 'europäischsten', 0.31 | 'unionsbürgerschaft', 0.31

Table 3: Top 10 most probable translations of the English word "European" to the German word.

d) Decoding

Define and implement an algorithm to find a translation, given a sentence in the source language. That is, you should try to find E = argmaxE P(E|F) In plain words, for a given source-language sentence F, we want to find the English-language sentence E that has the highest probability according to the probabilistic model we have discussed. Using machine translation jargon, we call this algorithm the "decoder." In practice, you can't solve this problem exactly and you'll have to come up with some sort of approximation. Exemplify how this algorithm works by showing the result of applying your translation system to a short sentence from the source language. As mentioned, it is expected that you will need to introduce a number of assumptions to make this at all feasible. Please explain all simplifying assumptions that you have made, and the impact you think that they will have on the quality of translations. But why is it an algorithmically difficult problem to find the English sentence that has the highest probability in our model?*

The last part of the implementation is to find E^* defined in equation 1 in section 2 and thus end up with the translated sentence. A few assumptions and simplifications were made to calculate E*. First, because of the Markov assumptions made in P(E) and P(F|E), no consideration was given to words after every bigram in the sentence. This is a big downside when translating since words at the end of a sentence could have provided important context.

Additionally, finding the highest probable sentence P(E) is algorithmically difficult since the search space becomes extremely large. It is not possible to search through all possibilities. Therefore, a simplification was made by only using ten potential translation words, given by the top ten words with the highest P(F|E). The translation method works by multiplying the probabilities for these ten words with the probability for the already translated part of the sentence having this potential word last. This could exclude some alternative translations where the probability of the sentence is so high that it would outweigh a low translation probability. The code on rows 10-23 in listing 3 demonstrates how the simplification was implemented.

Even though the P(E) was only used for the top ten translation words, it still had a large impact on the result. No penalization was given for repetition or usage of non-informative words, which was reflected in the results. The P(E) is usually higher for non-informative common words such as 'the', 'and' or 'of' since these words occur many times in the training data and therefore also in many bigrams and in many combinations of language sentences. Some kind of penalization could have been given if a suggested word had already occurred in the so-far translated sentence since such repetitions are unlikely to the extent they occurred.

Another way to deal with this problem could have been to give the P(E) part of E^* less impact than the P(F|E). When translating by only using the P(F|E) and not P(E), the translations turned out much better. This indicates that the two factors may not have equal weights. The sentence probability P(E) could be normalized in some way so that the values don't differ as much.

Another important downside is that the translator can only handle words that were present in the training data. This could have been handled by for example keeping the word that is not recognized as it is, which some translators do. Another idea could be to split the word and hopefully, parts of the word could have been present in the training data. This could be useful for e.g. verb conjugations, but could also lead to incorrect translations if the subparts of a word do not correlate with the meaning of the whole word.

The full implementation is shown in listing 3.

```python
#translation of a sentence is calculated by product of sentence
#probability p(e) and the translation probability p(f|e)
def translate_sentence(sentence, t, word_counter_bigrams, word_counter):
    words = sentence.split(' ')
    words = [word.lower() for word in words]

    translated_sentence = []
    translated_sentence.append('NULL')

    for word in words:
        probabilities = []
        top_ten_translations = sorted(
            [(key, inner_dict[word]) for key, inner_dict in t.items() if word
                in inner_dict],
            key=lambda x: x[1],  # Sorting key based on the value
                                #(second element of the tuple)
            reverse=True
            )[:10]  # Selecting the top ten

        for word1, t_value in top_ten_translations:
            temp_sentence = ' '.join(translated_sentence)
            probabilities.append(probability_of_sentence(temp_sentence + ' '
                    + word1, word_counter_bigrams, word_counter)*t_value)

        translated_sentence.append(top_ten_translations
                                [np.argmax(probabilities)][0])

    return translated_sentence[1:]
```

Listing 3: Implementation of the EM algorithm.

The translator was used for two different sentences in English and the result in German is shown in table 4. The result is not completely incorrect but it is not completely correct either.

To see if the result would be different if the sentence probabilities were more similar, the method for calculating sentence probabilities was slightly updated. The line:

Source Language (English)	**Target Language (German)**
I am a European speaking here	ich bin eine europäische werte die
A cat was here today and spoke about some things	eine wirtschaft und hier heute und von den von so

Table 4: Translated sentences from English to German using the implementation in listing 3.

```
probabilities.append(prob if prob > 0.000000001 else 0.000001)
```

was changed to:

```
probabilities.append(prob if prob > 0.1 else 0.1)
```

The new results are presented in table 5. The translations are better in some ways, but 'I am' which was correctly translated in the first example is now incorrectly translated to 'ich ich'. The second sentence is much better in the second example though. This highlights the importance of the sentence probability since that was probably much higher for 'ich bin' than for 'ich ich'. But also demonstrates the importance of the translation probability since that generated the correct translation of more words, like translating 'about some things' to 'über einige dinge'.

Source Language (English)	**Target Language (German)**
I am a European speaking here	ich ich eine europäischen spreche hier
A cat was here today and spoke about some things	eine ausgeht wurde hier heute und gesprochen über einige dinge

Table 5: Translated sentences from English to German using the described adaptation.

3 Discussion

Section 3 discussed how to evaluate ML translations. The next section touches on biases in Google Translate's statistical algorithm and lastly section 3 discusses a faulty translation by Google Translate and discusses some reasons why the specific result was yielded.

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.)

First, what is a good translation? According to Popplewell, 2023 at "Institute of Translation and Interpreting" a good translation should cover six elements. First, *fitness of purpose*, the translated text should fulfill the same objective that the original text does. Second, *linguistic accuracy*, the translation must convey the same nuances as the original text. Third, *excellent writing*, the translation should be good text in its own right. Fourth, *cultural awareness*, the translation needs to represent the cultural reality of the target language's region. Fifth, *subject expertise*, expert knowledge needs to be conveyed convincingly. Sixth, *quality assurance*, the translation needs to be quality-checked.

The points above elude to the fact that there are many requirements a good translation needs to reach. Popplewell, 2023 talks about human translation and thus talks about the importance of education and knowledge of the translator. Humans need to encode rules or objectives in the ML translation model such as the six above to give satisfactory translations. This can be complex.

Just through reasoning, the two most important aspects of a good translation could be that it semantically conveys the same information as the source text, and secondly, that the translated text is linguistically accurate, i.e., using proper grammar and language conventions.

Manual Evaluation

One manual procedure would be to have educated translators to evaluate the accuracy of translations. A very specific method would be to measure the number of changes a professional translator would make to a certain translation. The maximum number of points would be,

$$\text{max\_score} = \text{number\_of\_words} + 1 + 6 \tag{9}$$

A point would be deducted for every changed word and the added point would be subtracted if the grammar of the order of the words were to change. The last $+6$ points relate to the six points indicating if a translation is good or not. A point should be deducted for every objective that the ML model's translation fails to reach.

Below is an example where the professional made one change in a 4-worded sentence. The score is thus 10/11 since one change was made. The additional points besides the number of letters were described above. The score would become 0 if the translator changed everything in the translation.

Model output: He is a doctor.
Translator changed to: She is a doctor.
Score: 10/11

Another approach could be to have people knowing the source language and target language grade the translations both regarding the fluency of the text and the content. Whether these two aspects would be given equal importance depends on the situation. In some cases, the it is more important to convey the same message than having fluency in the language, and in some cases it might even be the other way around. For example when reading a novel, some people might be more bothered with incorrect language than if the information differed slightly from another language. A downside with this approach is the subjectivity when grading texts if there is not a very detailed protocol to follow, which can be difficult to create.

An advantage of the manual proposal is the expert knowledge in the loop. The biggest drawback is the demand for manpower and resources it would take. This evaluation strategy scales badly.

Automatic Evaluation

An automatic way of evaluating would be to compare the translated sentence to a big "bag of sentences". If the translated sentence exists among the bag of sentences then the sentence is assessed to be accurate. The bag of sentences could be built by loading big amounts of literature or other data that the creator of the model trusts are written in the correct language. Data sources such as twitter should be avoided. An example is inserted below

Translation: I good feel
Bag of sentences:
...
I am happy
My name is
I feel good
...

The translation would be labeled as a bad translation as it does not exist in the bag of sentences. A big drawback of this method is that it demands HUGE amounts of data to cover the full language.

Moreover, it labels the translation as binary as either "good" or "bad". In reality, there are more nuances to whether a translation is accurate or not. Another drawback is that it does not consider if the sentence is translated correctly just if the translation exists in the language. The biggest pro of this method is that it most probably would get the grammar correct.

Furthermore, an idea could also be to compare translations with already translated texts by calculating different scores. For example, word usage similarities like the F1 score, which calculates the occurrences of the exact same words in two texts, could be used together with a semantic similarity score. That way, both the phrasing and the meaning of the translated text would be taken into consideration. A downside to this is the need for translated texts as training data. It could potentially also be done by comparing translations given my the model with translations given by a bigger and already evaluated translation model. A downside to this is that the translation model used for comparisons may also make errors.

b)

The following example shows several 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?

According to “A history of machine translation from the Cold War to deep learning”, 2018 google has used "phrase-based statistical machine translation" since 2016 which has increased its accuracy though, the use of statistical ML translation opens up the system to massive amounts of human biases. The reason that the translation becomes "he" with professions traditionally associated with men and "she" with those often associated with women is probably directly due to the training data. This fact indicates that texts about doctors and computer programmers are more likely to refer to men and texts about babysitters to women.

In addition “A history of machine translation from the Cold War to deep learning”, 2018 writes that Google has used a crowd-sourcing system where users get several suggestions on what the translation should be. The user then gets to choose what translation is most accurate. People using Google Translate have probably thought that the translation from "ta" to "him" is most accurate in typically male-dominated industries and to "her" in women-dominated industries. This ability for human selection could have further increased the system's biases.

A bug would indicate that the system is predicting something inaccurate which seems to be the case with the prediction that "ta" is translated to him in the case of doctors. Estonia has a very high percentage of female doctors reported to be 72.5% in 2023 according to Staff, 2023. Thus "ta" should not be translated to "him" if the goal is to predict the most accurate pronoun of a doctor. This effect is thus a bug in some cases where the input data and society's stereotypes are out of line with reality. Another interesting thing to discuss would be how these biased systems affect society's biases. What effect does Google Translate have on people as it confirms their biased opinions? What would happen if their opinions were challenged or corrected instead?

c)

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 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? [NOTE: this example is not reproducible as of 2024 since Google Translate has been updated since the example was created.]

Context plays an important role in the world of statistical models consequently the first two sentences are probably translated correctly due to their homogenous context around the word "bat". The first sentence is connected with a sport. It is a common context for the 'slagträ' meaning of the word since the main purpose of a 'slagträ' is to hit balls. Therefore, the translation model probably has encountered the usage of the word bat in this context in the training data and therefore handles it in the right way. The same thing goes for the next sentence. Eating insects is clearly associated with the animal bat and could also have been encountered during training. However, the context of the third sentence can be harder to understand. The model probably connects 'lives' with the animal, whereof 'fladder', and then forest with 'trä', and hence creates the word 'fladderträet'. It could be because this specific context did not occur in the training data and therefore the model does not know how to handle it.

Another reason why this might be the case is because of how the Google translator may deal with rare words. "A history of machine translation from the Cold War to deep learning", 2018 explains that if a word is not included in the lexicon, it is broken down into smaller pieces and these pieces are translated instead. It is possible that the Google translator looked at translations in word pieces for the first two sentences as well, but in these contexts the most probable word following 'slag' and 'fladder' was correct.

References

Boesch, G. (2023, December). Image recognition: The basics and use cases(2024 guide). Retrieved February 18, 2024, from https://viso.ai/computer-vision/image-recognition/

Chiusano, F. (2022). Two minutes nlp — perplexity explained with simple probabilities. *NLPlanet*. https://medium.com/nlplanet/two-minutes-nlp-perplexity-explained-with-simple-probabilities-6cdc46884584

Choosing between a rule-based vs. machine learning system | TechTarget. (n.d.). Retrieved February 18, 2024, from https://www.techtarget.com/searchenterpriseai/feature/How-to-choose-between-a-rules-based-vs-machine-learning-system

Collins, M. (n.d.). Statistical machine translation: Ibm models 1 and 2. http://www.cs.columbia.edu/~mcollins/courses/nlp2011/notes/ibm12.pdf

A history of machine translation from the Cold War to deep learning. (2018, March). Retrieved February 18, 2024, from https://www.freecodecamp.org/news/a-history-of-machine-translation-from-the-cold-war-to-deep-learning-f1d335ce8b5/

Popplewell, M. (2023, September). 6 ingredients of a good translation. Retrieved February 20, 2024, from https://www.iti.org.uk/resource/6-ingredients-of-a-good-translation.html

Staff, G. (2023, May). Gender bias in medicine: Which countries have the most female doctors. Retrieved February 18, 2024, from https://geographical.co.uk/culture/countries-with-the-most-female-doctors