Chalmers Tekniska Högskola

CHALMERS
UNIVERSITY OF TECHNOLOGY

DESIGN OF AI SYSTEMS (DAT410)

Module 4: Natural language processing

Group 38

Himanshu Sahni
9812075498
sahni@chalmers.se
MPCAS

Nina Uljanić
9609134920
uljanic@chalmers.se
MPCAS

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.

Hours:
Himanshu Sahni: 25h
Nina Uljanić: 30h

1. Reading and reflection

Problem 1

Read the following popular-scientific overview (Links to an external site.) of the history of machine translation. 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.

Himanshu Sahni

1. Can you think of other AI problems where we can see a similarly wide range of approaches?

   In my opinion, AI has been contributing to a large number of domains in our daily life. There are countless approaches that we can find to solve problems using AI, and this has enhanced the user experience and improved efficiency as well. An excellent example would be google translate itself. Some of the areas where AI has contributed immensely would be control systems, robotics, healthcare, etc.

2. Can you think of similarities between some rule-based translation systems and state-of-the-art neural systems?
   According to me, the major similarity between the two systems is the idea on which these two systems are built, i.e., the "interlingua,". It is their principal point of resemblance. These systems are designed to decode the characteristics into the target language after they have been encoded using various techniques from the source language. Usually, all the translation systems work on the same features i.e., syntax, semantics, morphology, and phonology.

3. 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)?
   Although rule-based solutions may not be as precise or sophisticated as neural or statistical ones, they are easier to use and do not require as much advanced training as neural or statistical solutions do. Therefore, for tasks that are basic to intermediate such as classification or regression of low dimensionality, we can still use these old methods. These rules can be afforded by companies with limited resources or small-sized businesses as they are more affordable than neural implementation in terms of infrastructure.

Nina Uljanić

1. I do not have previous experience with applications of AI so I am not aware of how the corresponding fields have evolved over time, that is to say, that I do not know "of other AI problems where we can see a similarly wide range of approaches." If had to guess, I would assume that the fields of computer vision and robotics would be good candidates, however, I do not know the details of their evolution and can therefore not give a concrete answer.

2. Rule-based machine translation (RBMT) systems are based on linguistic rules that allow the words to be put in different places and to have different meanings depending on the context, while neural machine translation (NMT) systems utilize neural network techniques to predict the likelihood of a set of words in sequence. Both of these systems require large amounts of high-quality, albeit different, training data. They can also handle multiple languages, that is, they can be designed to translate between more than one language pair, unlike for example interlingua.

3. A rule-based machine translation system might be better than a modern neural or statistical system when translating between similar languages. RBMT systems do not require as great amounts of training data to perform, unlike NMT systems. The rules for translating between related languages can be similar, in which case it is easier to design a set of rules that can handle multiple language pairs. This creates a less complex system than in the case of NMTs. Such a system might be preferable in certain cases.

2. Implementation

Subtask a

The implementation of this task consists of reading the file (*read_file_to_pandas*), generating a dictionary (*generate_dictionary*) and calculating the probability of a word being chosen randomly (*get_probability*).

In *get_probability*, the variable *word_count* is assigned the value of the frequency of the input word in the dictionary of the language from which the word originates. The variable *all_words_count* is assigned the value of the total number of words in the dictionary *sv_en_dict*, which is calculated by summing up the values of all the entries; this is equal to the number of tokens in the corpus. The variable *probability* is calculated by dividing the frequency of the input word (*word_count*) by the total number of words in the corpus (*all_words_count*).

The probability of choosing the word *speaker* is 0.00003891, while for *zebra* it is 0.

The 10 most frequent words in English are:

	word	**count**
1	the	19327
2	of	9344
3	to	8814
4	and	6949
5	in	6124
6	is	4400
7	that	4357
8	a	4271
9	we	3223
10	this	3222

The 10 most frequent words in French are:

	word	**count**
1	apos	16729
2	de	14528
3	la	9746
4	et	6620
5	l	6536
6	le	6177
7	à	5588
8	les	5587
9	des	5232
10	que	4797

The 10 most frequent words in German are:

	word	**count**
1	die	10521
2	der	9374
3	und	7028
4	in	4175
5	zu	3169
6	den	2976
7	wir	2863
8	daß	2738
9	ich	2670
10	das	2669

The 10 most frequent words in Swedish are:

	word	**count**
1	att	9181
2	och	7038
3	i	5954
4	det	5687
5	som	5028
6	för	4959
7	av	4013
8	är	3840
9	en	3724
10	vi	3211

Subtask b

The implementation of this task consists of generating a bigram dictionary (*generate_bigram_dictionary*), calculating the probability of a bigram (*calculate_bigram_probability*) and calculating the probability of a sentence (*calculate_sentence_probability*).

(*generate_bigram_dictionary* iterates through all sentences in the corpus in order to create a bigram dictionary of the corpus. Each sentence is first split into tokens, after which the *bigram* Counter is updated with a list of bigrams created from each sentence by zipping together consecutive pairs of tokens using list comprehension. The function returns a list containing the *bigram* Counter and a variable called *total_bigrams*, which is the sum of all bigram counts in the Counter (i.e. the number of bigrams). This variable is used later for calculating the probability of a specific bigram.

calculate_bigram_probability first checks whether *w1* is in the *dictionary* and, if it is, it then calculates the probability by dividing the count of the bigram (*w1*, *w2*) by the count of *w1* in the dictionary. If *w1* is not in the dictionary, the probability is returned as 0 - else we'd get a division by 0 error. The function returns the probability of the bigram (*w1*, *w2*) occurring given *w1*.

calculate_sentence_probability first splits the sentence into tokens, then iterates over each bigram. For each bigram, it calls the *calculate_bigram_probability* function to get the probability of that bigram, given the preceding word. The probabilities of each bigram are multiplied together to get the probability of the sentence according to the bigram model, which is then returned. If any bigram has a probability of 0, then the sentence does as well.

With our implementation of the bigram model, we computed the probability of two short sentences. The first one is made up of randomly selected words from the English corpus ("My question fleet will raise targets annually .") while the other sentence is a piece extracted from a sentence found in the corpus ("the final part of my question ."). The probability for the first sentence is 0, while for the other one, the probability is $5.6360229e^{-10}$ or 0.00000000056360229. When computing the probability of a sentence that contains a word that did not appear in the training texts, we get probability 0. This is due to the implementation where we divide the bigram count by the word count in order to calculate probability. It is clear that a word not in the dictionary will pose an issue since we'd be dividing by 0. Therefore, the probability is set to 0. There do not seem to be any issues if testing on a very long sentence, however, the probability will be very low (close to 0) as many bigrams do not exist in our bigram dictionary.

Subtask c

The implementation of this task consists of implementing the IBM model 1 algorithm (*ibm_model_1*) and finding the most likely translation(s) for a word (*get_most_likely_translation*). The IBM model 1 is a statistical machine translation model which uses an Expectation-Maximization (EM) algorithm to estimate the parameters. The algorithm is used to estimate the probability of a target word given a source word, t(f|e). Our implementation is as follows.

In *ibm_model_1*, k denotes the number of sentences in the corpora (*target_corpus* and *source_corpus* have equal number of sentences so we count the number of sentences in only one of them). t represents the translation probability matrix, initialized with equal probabilities for each source word given each target word. The IBM model 1 algorithm is then executed over a specified number of iterations. In each iteration, the *count_e_given_f* matrix and *total_count_f* list are initialized to 0 (c(f,e)= 0, c(e)= 0); they are used to compute intermediate values for the translation probabilities.

For each sentence pair in the corpora, the sentences are extracted one at a time for further processing. The first loop extracts each word in the target sentence one by one and finds its index in the dictionary of the target corpus. Here we also initialize *total_count* to 0, which is used to store the sum of the probabilities of the target word given all the source words in a particular sentence and to then normalize the probabilities so that they sum up to 1. The second loop, just like the first one, extracts each word in the source sentence one by one and finds its index in the dictionary of the source corpus. The *total_count* is updated with the current t value for the target word and source word. The *alignment_probability* is then calculated by dividing the t value (corresponding to the target and source words) by the *total_count*, and this value is used to update both count_e_given_f and *total_count_f*. This process repeats until all sentences have been processed.

Finally, to finish off the iteration, the t matrix is updated by dividing *count_e_given_f* for each word in target and source dictionary by *total_count_f* for each target word and source word. This process is repeated for the specified number of iterations.

Once the method has been completed, it returns the translation probability matrix t with the estimated probabilities of each target word given each source word.

get_most_likely_translation first finds the index of the word whose translation we are looking for. It then extracts the probabilities of target words corresponding to the source word by using its index, into *candidates*. The number of words to be extracted is given as *nr_of_words*. This list is then sorted so that a given number of most likely translations can be extracted. Note that the output is not 0,1,...,9 but the actual indices. The words themselves are found using the indices and returned.

The following results were obtained with our first submission code, which was running fairly quickly but is no longer in use due to mistakes (which were noticed and promptly fixed). The new code's run time is too long to evaluate before the (new) deadline, however, we expect that the lists would change quite a bit.

The 10 words that the English word *european* is most likely to be translated into in French are :

1. copies
2. consacrent
3. pierre
4. extraordinaire
5. embaucher
6. économisé
7. arrêt
8. ferme
9. diminue
10. partout

The 10 words that the English word *european* is most likely to be translated into in German are :

1. einzeln
2. beschlossene
3. gewinnen
4. paket
5. inneren
6. dokumentiert
7. wofür
8. eindrucksvollen
9. bekundet
10. abkommen

The 10 words that the English word *european* is most likely to be translated into in Swedish are :

1. påfyllning
2. handicap
3. präglade

4. bristfälligt
5. förenkla
6. mikroföretag
7. hantverkare
8. subjekt
9. oförmågan
10. regelsystemet

These results were the same for 1, 5, and 10 iterations of the algorithm, possibly indicating that there were some errors in our (initial) implementation.

Subtask d

The implementation of this part consists of translating a sentence to English (*translate_sentence*). *translate_sentence* translates a sentence from a source language to a target language using the translation probability matrix generated above. To start off, an empty list is generated to store the translated words. For each word in the source sentence, it finds the word in the target language that has the highest probability. If a word does not exist in the source word, it is skipped. Else, it uses the function [18] described above to find a match. These "most likely translations" are stored in the *translated_sentence* list. Once all source words have been evaluated, the elements in the *translated_sentence* list are joined together to create a new string, which is returned as the output of the function.

For this approach to work, the corpus must be extensive. That is, the training data must contain a great amount of words combined in various ways (i.e. bigrams). The words being translated must be present in the dictionary generated from the corpus. In our implementation, words that do not exist in our dictionary are ignored, as we cannot know whether a fitting target word exists in our dictionary.

A drawback with this approach is that, while in reality not every word has a match in the target language, this method assigns a word to every source word. This could lead to translations that are nonsensical. Therefore, the translation generated by this method is, at best, an approximation, and at worst, a big mess.

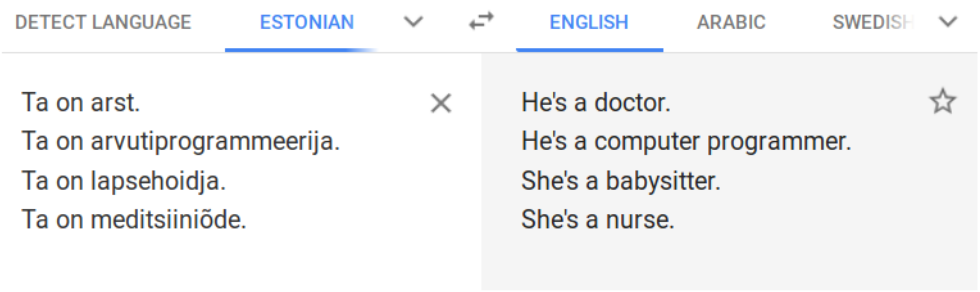
3. Discussion

1. 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.)

   One of the methods which were used earlier is human evaluation. It is the most reliable method to evaluate a translation. Human evaluation refers to the people who are language experts and they can measure the quality of translation from one language to another (source and target). It can be verified if the context is preserved or not. The only disadvantage is that this is a slow and tedious process and requires high manpower and it could be expensive as well.

   The other form of evaluation could be a dataset that has a lot of translations of words in different combinations and contexts and then they can be used as a reference to compare. The major thing here would be to have a large dataset that is approved by a language expert so that it can be used as a reference. This will be a faster and cheaper way to implement and evaluate and can be updated regularly. However, there are chances of low-quality of translations and the translations could not be accurate.

2. The following example 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:

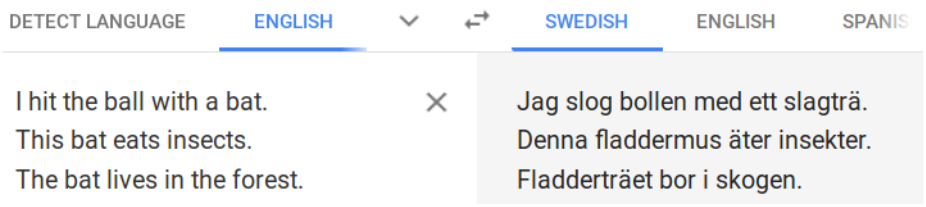


   - What do you think are the technical reasons we see this effect?
     It is clear that the word 'ta' is gender-neutral in Estonian. The translation machine assigns gender to the words in English based on the context and the dataset used for training. For example, here we can assume that the training set had the words 'doctor' and 'computer programmer' followed by or referenced as 'him' and the words 'babysitter' and 'nurse' as her. This can be a reason for this assignment to the words when translated to English.
   - Do you consider this to be a bug or a feature?

     The translation machine tries to translate a word by relating it to the context and finding the closest possible option i.e., giving the best possible translation in terms of quality. This should be considered a feature and one of the possible solutions to improve this feature could be to translate it to all the possible choices in the target language and then relate it to the context.

3. Below, we consider three sentences that include the English word bat and their automatic translation into Swedish by Google Translate. In the first example, the bat

refers 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 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?

  We know that the context and the relation between words are important while translating sentences from one language to another. To understand this, the first sentence has the word 'hit', 'ball', and 'bat', it is possible that while training the system it has seen more occurrences of these words and the relations between them relating to the sports. This is the reason google translate was able to translate the English word bat to its corresponding Swedish word.
  Similarly, for the second sentence, it has the words 'bat', 'eats', and 'insects', it is possible that while training the system it has seen more occurrences of these words and the relations between them relating to the living organisms.

- What might be the reason that it has invented a new nonsense word in the third case

  As per the given English sentence, the correct translation of the word 'bat' to Swedish should be related to the 'flying mammal" as in the second sentence. The possible reason for this could be the data on which the system is trained because there might not be enough occurrences relating to the context. It is possible that the system was unable to extract the pattern or the features from the data because the dataset has words and relations that are more focused on sports. We use the same words in different contexts as we see in this example, and this can create a misunderstanding for the system if we try to do the translation based on words rather than sentences.

Individual Summary and Reflection

Lecture 4: Natural language processing (2023-02-07)

Lecture summary by Himanshu Sahni

The lecture introduced the field of Natural language processing with a focus on machine translation. It highlighted that processing text is an important research area in the field of AI. The lecture covered different approaches to machine translation, and different problem-solving techniques with a focus on probabilistic models. Language processing involves understanding the language, some known properties of language are discrete, structured, diverse, and sparse. the typology of tasks in NLP are:

- categorization - it focuses on the nature of the entire text, for example, spam filter
- tagging - this aims at detecting relevant words within a text and assigning them a meaningful label
- parsing - this focuses on detecting relations between entities (words) to develop a structure tree
- generation - the goal is to produce a text given an input, for example, translation machines

The research in the field of NLP started in the late 80s with a focus on data-driven techniques i.e., probabilistic linear models from 1990 to 2000, discriminative linear models from 2000 to 2015, and then neural models from 2015 onwards, Although neural network models are expensive to build and offer end-to-end solutions, usually built-in blocks and transfer learning can be applied to them. However, they are some challenges as well, neural models are difficult to configure, have high training time, high energy consumption, special hardware requirements, etc.
Machine translation was a requirement majorly for the government and the military for intelligence tasks. Many approaches were based on an idealized intuition of the translation process known as interlingua. The idea was to map the source-language sentence into some “meaning representation” or "interlingua" and then convert the representation into the target language. The machine translation systems were categorized into two categories i.e., domain-specific MT systems and data-driven MT systems. The domain-specific were based on rules but did not work out in texts with a large number of words. On the other hand, the data-driven systems were trained on example texts. The statistical and neural MT systems had a supervised learning approach. The architecture of a neural MT consisted of an encoder used to summarize the information in the source sentence and a decoder used to generate the target-language output in a step-by-step fashion based on the encoding.
In the last part of the lecture, we looked into IBM models which are word-based statistical models. The goal was to translate sentence F (in language A) into E* (the most probable sentence in language B).
This can be written as
E* = $argmax_E$ P(F|E) = $argmax_E$ P(E)P(F|E)
where P(E) is called a language model and P(F|E) is called a translation model.
In a probabilistic model, we have a word sequence W, distributed as w_1, w_2,..., w_n
where P(w) = P(w_1, w_2,..., w_n) The idea is to write the next word in terms of the previous words and using the chain rule along with Markov’s assumption we estimate the probabilities using maximum likelihood estimation. We implement the same in this week’s assignment.

Module 3 reflection by Himanshu Sahni

In the previous module, we learned about the various AI tools that can be used to develop AI systems. I understood how data could be in different forms and understood some of the techniques to extract information from it. A critical learning from this module was to convert the data given into proper input for the model. It is necessary to understand the different distributions of data points in the dataset and bring them to the same scale using necessary operations and finally splitting them into the training, validation, and test sets. Although this was a small system with only one parameter to tune i.e., k (neighboring elements), it gave an insight into how these hyperparameters in a model can affect the accuracy of a system.

Lecture summary by Nina Uljanić

The lecture covered different approaches to NLP, with a focus on machine translation. The applications of NLP include spam filters, spell and grammar checkers, translation systems, and dialogue systems. Tasks in NLP include categorization, tagging, parsing, and generation. The lecture also discussed the advantages and challenges of neural models, but the latter stuck with me. These include training time, energy consumption, hardware requirements, explainability, and reproducibility. The goal of neural machine translation is to generate a target language text given a source language text. The fundamental idea in neural machine translation is the encoder-decoder architecture. The IBM models for word-based statistical machine translation were mentioned, and the IBM model 1 (least complex one) was discussed in detail. The IBM models use word alignments and the expectation-maximization algorithm to determine the most probable sentence.

Module 3 reflection by Nina Uljanić

In module 3 we looked at AI tools and even implemented one ourselves - for weather prediction. The main takeaway from the module (for me) was that there's many different types of data that can be used for training and that there ware many ways to process it. Different different system architectures require different types of data and many parameters need to be considered when designing an AI system. The choice, of course, depends on the use case and end goal.

Module 4 NLP

February 15, 2023

[ ]:
```
import pandas as pd
import numpy as np
from collections import Counter, defaultdict
import re
```

1 (a) warmup

1.1 Write code to collect statistics about word frequencies in the two languages. Print the 10 most frequent words in each language.

[ ]:
```
def read_file_to_pandas(filename):
    with open(filename, "r", encoding="utf-8") as file:
        data = file.readlines()
    return pd.DataFrame(data)
```

[ ]:
```
def generate_dictionary(file): # does not generate a dict, but a Counter␣
 ↪(slight differences, Counter is better fit here)
    words_list = []
    for sentence in file[0]:
        words = re.findall(r'\b\w+\b', sentence) # find all words in sentence
        words_list.extend(words)
    return Counter(words_list)
```

[ ]:
```
# divide the number of times the given word appears in the text by the total␣
 ↪number of words in the text
def get_probability(word, dictionary):
    word_count = dictionary[word]
    all_words_count = sum(dictionary.values())
    probability = word_count / all_words_count
    return probability
```

[ ]:
```
## SWEDISH-ENGLISH
# the Swedish part of the Swedish-English dataset
sv_sv = "C:
 ↪\\Users\\Nina\\Desktop\\STUDIES\\Uni\\MPCAS22\\DAT410_DAIS\\Assignment␣
 ↪4\\dat410_europarl\\europarl-v7.sv-en.lc.sv"
df_sv = read_file_to_pandas(sv_sv)
```

```
# the English part of the Swedish-English dataset
sv_en = "C:
 ↪\\Users\\Nina\\Desktop\\STUDIES\\Uni\\MPCAS22\\DAT410_DAIS\\Assignment␣
 ↪4\\dat410_europarl\\europarl-v7.sv-en.lc.en"
df_sv_en = read_file_to_pandas(sv_en)

## GERMAN-ENGLISH
# the German part of the German-English dataset
de_de = "C:
 ↪\\Users\\Nina\\Desktop\\STUDIES\\Uni\\MPCAS22\\DAT410_DAIS\\Assignment␣
 ↪4\\dat410_europarl\\europarl-v7.de-en.lc.de"
df_de = read_file_to_pandas(de_de)

# the English part of the German-English dataset
#de_en = "C:
 ↪\\Users\\Nina\\Desktop\\STUDIES\\Uni\\MPCAS22\\DAT410_DAIS\\Assignment␣
 ↪4\\dat410_europarl\\europarl-v7.de-en.lc.en"
#df_de_en = read_file_to_pandas(de_en)

## FRENCH-ENGLISH
# the French part of the French-English dataset
fr_fr = "C:
 ↪\\Users\\Nina\\Desktop\\STUDIES\\Uni\\MPCAS22\\DAT410_DAIS\\Assignment␣
 ↪4\\dat410_europarl\\europarl-v7.fr-en.lc.fr"
df_fr = read_file_to_pandas(fr_fr)

# the English part of the French-English dataset
#fr_en = "C:
 ↪\\Users\\Nina\\Desktop\\STUDIES\\Uni\\MPCAS22\\DAT410_DAIS\\Assignment␣
 ↪4\\dat410_europarl\\europarl-v7.fr-en.lc.en"
#df_fr_en = read_file_to_pandas(fr_en)

# the English datasets are equal for all 3 languages, so we are using only one␣
 ↪of them, europarl-v7.sv-en.lc.en
df_en = df_sv_en
```

[ ]:
```
# Generate dictionaries (unsorted)
en_dict = generate_dictionary(df_en) # using df_sv_en
sv_dict = generate_dictionary(df_sv)
de_dict = generate_dictionary(df_de)
fr_dict = generate_dictionary(df_fr)
```

[ ]:
```
# Get 10 most common words from the dictionaries
print("The top 10 English words are: ")
for key, value in en_dict.most_common(10):
    print("{} ({})".format(key, value))
```

```
print("\n")
print("The top 10 Swedish words are: ")
for key, value in sv_dict.most_common(10):
    print("{} ({})".format(key, value))

print("\n")
print("The top 10 German words are: ")
for key, value in de_dict.most_common(10):
    print("{} ({})".format(key, value))

print("\n")
print("The top 10 French words are: ")
for key, value in fr_dict.most_common(10):
    print("{} ({})".format(key, value))
```

[ ]:
```
# using english dictionary (en_dict)
print("{:.8f}".format(get_probability("speaker", en_dict))) # print 8 decimals␣
↪(in case of scientific notation)
print("{:.8f}".format(get_probability("zebra", en_dict)))
```

2 (b) language modeling

[ ]:
```
def generate_bigram_dictionary(corpus): # again, creates a Counter (of all␣
↪bigrams)
    bigrams = Counter()
    for sentence in corpus[0]:
        tokens = sentence.split()
        bigrams.update([(tokens[i], tokens[i+1]) for i in range(len(tokens) -␣
↪1)])
    total_bigrams = sum(bigrams.values()) # used for calculating the␣
↪probability of a certain bigram
    return [bigrams, total_bigrams]
```

[ ]:
```
def calculate_bigram_probability(dictionary, bigrams, w1, w2):
    probability = 0
    if w1 in dictionary:
        probability = bigrams[(w1, w2)] / dictionary[w1] # nr of bigrams of the␣
↪form (w1, w2) divided by nr of unigrams (w1)
    return probability
```

[ ]:
```
def calculate_sentence_probability(dictionary, bigrams, sentence):
    tokens = sentence.split()
    probability = 1
    for i in range(len(tokens)-1):
        w1, w2 = tokens[i], tokens[i+1]
```

```
        bigram_probability = calculate_bigram_probability(dictionary, bigrams,␣
 ↪w1, w2)
        probability *= bigram_probability
    return probability
```

[ ]:
```
# Bigram dictionaries (Counter)
[en_bigram, en_bigram_count] = generate_bigram_dictionary(df_en)
[fr_bigram, fr_bigram_count] = generate_bigram_dictionary(df_fr)
[de_bigram, de_bigram_count] = generate_bigram_dictionary(df_de)
[sv_bigram, sv_bigram_count] = generate_bigram_dictionary(df_sv)

print("English bigrams: ")
for key, value in en_bigram:
    print("{} {}".format(key, value))

print("\n")
print("French bigrams: ")
for key, value in fr_bigram:
    print("{} {}".format(key, value))

print("\n")
print("German bigrams: ")
for key, value in de_bigram:
    print("{} {}".format(key, value))

print("\n")
print("Swedish bigrams: ")
for key, value in sv_bigram:
    print("{} {}".format(key, value))
```

[ ]:
```
# Probability of a sentence
sentence = "My question fleet will raise targets annually .".lower()
print("{:.8}".format(calculate_sentence_probability(en_dict, en_bigram,␣
 ↪sentence))) # handle scientific notation as above
sentence = "the final part of my question ."
print("{:.8}".format(calculate_sentence_probability(en_dict, en_bigram,␣
 ↪sentence))) # handle scientific notation as above
```

3 (c) translation modeling

[ ]:
```
def ibm_model_1(source_corpus, source_words, target_corpus, target_words,␣
 ↪iterations): #  training corpora for the two languages (pandas dataframes)
    k = len(source_corpus) # = nr of sentences in target_corpus
    t = np.full((len(target_words), len(source_words)), 1/ len(source_words)) #␣
 ↪t(f|e); all words start with equal probability
```

```
    # The IBM model 1 algorithm
    for _ in range(iterations):
        count_e_given_f = np.zeros((len(target_words), len(source_words))) #c(f,e) = 0
        total_count_f = np.zeros(len(target_words)) # , c(e) = 0

        for sentence in range(k):
            source_sentence = source_corpus.iloc[sentence][0].split() # extract the source sentence
            target_sentence = target_corpus.iloc[sentence][0].split() # extract the corresponding target sentence

            for f in target_sentence: # for each target word
                if f in target_words:
                    j = [key for key in target_words.keys()].index(f) # get the index of the target word
                    total_count = 0

                    for e in source_sentence: # for each source word #18051
                        if e in source_words:
                            i = [key for key in source_words.keys()].index(e) # get the index of the source word
                            total_count += t[j][i]
                            alignment_probability = t[j][i] / total_count
                            count_e_given_f[j][i] += alignment_probability
                            total_count_f[j] += alignment_probability

        for f in target_words:
            j = [key for key in target_words.keys()].index(f) # get the index of the source word
            for e in source_words:
                i = [key for key in source_words.keys()].index(e) # get the index of the source word
                t[j][i] = count_e_given_f[j][i] / total_count_f[j]
    return t
```

[ ]:
```
# Translation probability matrix for English->Swedish
tpm_en_sv = ibm_model_1(df_en, en_dict, df_sv, sv_dict, 1)
```

[ ]:
```
# Translation probability matrix for English->French
tpm_en_fr = ibm_model_1(df_en, dict_en, df_fr, dict_fr, 1)
```

[ ]:
```
# Translation probability matrix for English->German
tpm_en_de = ibm_model_1(df_en, dict_en, df_de, dict_de, 1)
```

```
[ ]: # Store the resulting translation probability matrix to a file for easier␣
      ↪testing
     #np.savetxt('matrix_one_iter.csv', {translation_probability_matrix},␣
      ↪delimiter=',')
```

```
[ ]: # Load the translation probability matrix from the file
     #t = np.genfromtxt("C:
      ↪\\Users\\Nina\\Desktop\\STUDIES\\Uni\\MPCAS22\\DAT410_DAIS\\Assignment␣
      ↪4\\matrix_one_iter.csv",delimiter=',')
```

```
[ ]: def get_most_likely_translation(word, source_corpus, source_words,␣
      ↪target_corpus, target_words, nr_of_words, tpm):
         index = source_words[word] # find the index of the word
         candidates = tpm[:,index] # get the probabilities of target words in␣
      ↪relation to source word
         indices = np.argsort(candidates)[::-1][:nr_of_words] # extract indices of␣
      ↪10 words that have the highest probability
         print(indices)
         translations = [list(target_words.keys())[i] for i in indices] # fetch the␣
      ↪words given indices

         return translations
```

```
[ ]: # Get 10 most likely translations for the word "european" and run for 5␣
      ↪iterations

     word = "european"
     iterations = 5
     nr_of_words = 10

     top_10_fr = get_most_likely_translation(word, df_en, dict_en, df_fr, dict_fr,␣
      ↪nr_of_words, tpm_en_fr)
     for f in top_10_fr:
         print(f)

     top_10_de = get_most_likely_translation(word, df_en, dict_en, df_de, dict_de,␣
      ↪nr_of_words, tpm_en_de)
     for d in top_10_de:
         print(d)

     top_10_sv = get_most_likely_translation(word, df_en, dict_en, df_sv, dict_sv,␣
      ↪nr_of_words, tpm_en_sv)
     for s in top_10_sv:
         print(s)
```

4 d) Decoding

[ ]:
```
def translate_sentence(source_sentence, source_words, target_words, tpm):
    translated_sentence = []
    for word in source_sentence:
        if word not in source_words:
            continue
        else:
            translation = get_most_likely_translation(word, df_sv,␣
 ↪target_words_sv, df_en, source_words, 1, tpm)
            translated_sentence.append(translation)

    return " ".join(translated_sentence)
```

[ ]:
```
#sentence = "je vous invite à vous lever pour cette minute de silence"
sentence = "jag ber er resa er för en tyst minut ."
source_sentence = sentence.split()
tpm_sv_en = tpm_en_sv.T
translated_sentence = translate_sentence(source_sentence, dict_sv, dict_en,␣
 ↪tpm_sv_en)
print(translated_sentence)
```