IN4080 Exam Fall 2024

Solutions

January 6, 2025

---

1. Imagine you have a book written in English and the translation of the same book to another language (e.g. Russian). Does the **type-token-ratio** differ between the two languages? If so, which of the two do you expect to be higher and why? Justify your answer by explaining how the number of tokens and the number of types differs.

   (You can choose any 2 languages for this question, it doesn't have to be English and Russian.) (4 pts)

Solution: The main argument is morphological complexity. If the second language is morphologically richer than the first, then:

- The number of tokens goes down (because information expressed in separate words can be expressed in affixes)
- The number of types goes up (because there are more morphological variations of the same lemma)
- The type-token-ratio goes up.

Another explanation could refer to the lexical variety of a language. Some languages have "more words" to express the same meanings (e.g. a Romance-origin and a Germanic-origin word in English). The same result occurs if the translator isn't very skilled and has only a limited vocabulary. If the first language has high lexical richness and the second language has low lexical richness, then:

- The number of tokens stays roughly the same.
- The number of types goes down (because there is less lexical diversity).
- The type-token-ratio goes down.

A third explanation could refer to translationese, i.e. the observation that translated documents are often more verbose than the originals because the translators feel the need to "explain" some concepts. If the second language contains translationese, then:

- The number of tokens goes up (because the text becomes longer).
- The number of types stays roughly the same (because we assume that the added explanations mostly use words that are already present in the text).
- The type-token-ratio goes down.

2. You are given a training corpus containing the following words and their frequencies:

hopes	2
hoped	3
drop	4
dropped	3

Apply the BPE algorithm to this dataset.

1. List the **entire** subword vocabulary after **1 merging operation**.
2. How will the words *hopped* and *drops* be segmented with the model produced after **1 merging operation**?
3. List the **entire** subword vocabulary after **5 merging operations**.
4. How will the words *hopped* and *drops* be segmented with the model produced after **5 merging operations?** (6 pts)

Solution:

1. The initial vocabulary contains all single characters, and the first merging operation is $o\ p \rightarrow op$ (count of 12). Thus, the entire vocabulary after 1 merging operation is $h\ o\ p\ e\ s\ d\ r\ op$ (any order is fine of course).
2. $h\ op\ p\ e\ d$ and $d\ r\ op\ s$.
3. The five first merging operations are $o\ p \rightarrow op$ (12), $d\ r \rightarrow dr$ (7), $dr\ op \rightarrow drop$ (7), $e\ d \rightarrow ed$ (6) and $h\ op \rightarrow hop$ (5). The resulting vocabulary is $h\ o\ p\ e\ s\ d\ r\ op\ dr\ drop\ ed\ hop$.

   The operations with count 7 can be applied in any order, so $r\ op \rightarrow rop$ and $d\ rop \rightarrow drop$ are fine too. In this case, the vocabulary is $h\ o\ p\ e\ s\ d\ r\ op\ rop\ drop\ ed\ hop$.
4. $hop\ p\ ed$ and $drop\ s$.

3. The following table represents the confusion matrix of a three-class language identification model with labels Bokmål, Nynorsk and Danish:

		Gold		
		Bokmål	Nynorsk	Danish
Predicted	Bokmål	18	2	0
	Nynorsk	5	16	2
	Danish	3	3	17

Answer the following questions:

1. What is the overall accuracy of the classifier?
2. Compute the class-specific 2-by-2 matrices for each of the three classes.
3. What is the micro-average recall of the classifier?
4. What is the macro-average recall of the classifier?
5. Does one of the two averaging methods produce the same result as the accuracy? If so, which one and why? (10 pts)

Solution:

1. Accuracy: $\frac{18+16+17}{18+2+5+16+2+3+3+17} = \frac{51}{66} = \frac{17}{22} = 0.77$
2.

Bokmål	Gold True	Gold False
Pred True	18	2
Pred False	8	38

Nynorsk	Gold True	Gold False
Pred True	16	7
Pred False	5	38

Danish	Gold True	Gold False
Pred True	17	6
Pred False	2	41

3. Micro-average recall: $\frac{18+16+17}{(18+5+3)+(2+16+3)+(0+2+17)} = \frac{51}{66} = 0.77$
4. Macro-average recall: $\frac{1}{3} \cdot (\frac{18}{26} + \frac{16}{21} + \frac{17}{19}) = \frac{0.692+0.762+0.895}{3} = 0.78$
5. Micro-average recall is identical to accuracy. Both the numerator and the denominator contain the same values, just arranged differently. Since addition is commutative, the results are the same.

4. We want to build a Naive Bayes classifier for movie type classification. The following training corpus is given:

fun couple love love	comedy
furious shoot shoot fun	action
fast furious shoot	action
couple fly fast fun fun	comedy
fly fast shoot love	action

Estimate the priors and likelihoods. Apply add-one smoothing to the likelihoods.

1. Report all priors and likelihoods of the estimated model.
2. On the basis of these parameters, which class is predicted for the document *fast couple shoot*? Show the computations. (5 pts)

Solution:

1. Priors: $P(comedy) = \frac{2}{5}$, $P(action) = \frac{3}{5}$

   Likelihoods:

w	$C(w, comedy)$	$C(w, action)$	$P(w\|comedy)$	$P(w\|action)$
fun	3+1	1+1	$\frac{4}{16}$	$\frac{2}{18}$
couple	2+1	0+1	$\frac{3}{16}$	$\frac{1}{18}$
love	2+1	1+1	$\frac{3}{16}$	$\frac{2}{18}$
furious	0+1	2+1	$\frac{1}{16}$	$\frac{3}{18}$
shoot	0+1	4+1	$\frac{1}{16}$	$\frac{5}{18}$
fly	1+1	1+1	$\frac{2}{16}$	$\frac{2}{18}$
fast	1+1	2+1	$\frac{2}{16}$	$\frac{3}{18}$
TOTAL	9+7=16	11+7=18		

2. $P(comedy|fast, couple, shoot) = \frac{2}{5} \cdot \frac{2}{16} \cdot \frac{3}{16} \cdot \frac{1}{16} = \frac{12}{20480} = 0.0005$

   $P(action|fast, couple, shoot) = \frac{3}{5} \cdot \frac{3}{18} \cdot \frac{1}{18} \cdot \frac{5}{18} = \frac{45}{29160} = 0.0015$

   The predicted class is *action*.

5. The formula below represents the softmax function for a linear model:

$$P(y_j|x_i) = \frac{exp(\mathbf{w_j} \cdot \mathbf{f_i})}{\sum_{k=1}^{n} exp(\mathbf{w_k} \cdot \mathbf{f_i})}$$

Explain the formula by answering to the following questions:

1. What is y_j?
2. What is x_i?
3. What is $\mathbf{w_j}$?
4. What is $\mathbf{f_i}$?
5. What does $exp()$ stand for?
6. What does n represent?
7. Describe one NLP application in which the softmax function is used and explain its purpose. (7 pts)

Solution:

1. y_j represents the class/label to be predicted – the classes are ordered and y_j represents the jth entry in the ordered list.
2. x_i represents the example to be classified – the examples are presented in a list and x_i represents the ith entry in the list.
3. $\mathbf{w_j}$ is the weight vector corresponding to class y_j. There is one trained weight vector for each class.
4. $\mathbf{f_i}$ is the feature vector corresponding to example x_i. It encodes the example as a numeric vector of features.
5. $exp()$ represents the exponential function. In most cases, $exp(x) = e^x$.
6. n corresponds to the number of labels from which to choose.
7. Applications: any text classification task (sentiment analysis, language identification, ...), sequence labeling tasks (POS tagging, ...).

   Purpose: The dot product yields just a number, and it is hard to interpret that number. The softmax transforms the numbers into a conditional probability distribution.

6. Explain briefly how the BIO encoding scheme works and in which situations it is used.

   The following sentence contains three entities of type TODO, DATE and TIME respectively:

   Remind me to [call mom]$_{TODO}$ [today]$_{DATE}$ [at 2 p.m.]$_{TIME}$

   Annotate this sentence with the corresponding BIO tags. (3 pts)

Solution:

The BIO letters stand for B=begin, I=inside, O=outside. B and I are paired with a label (e.g. B-TODO, I-TODO), while O is not. The BIO-encoding is employed to annotate spans, i.e. sequences of more than 1 consecutive words, like named entities. Each word can be part of at most one span.

Annotated sentence:

```
Remind  O
me      O
to      O
call    B-TODO
mom     I-TODO
today   B-DATE
at      B-TIME
2       I-TIME
p.m.    I-TIME
```

7. Consider the two sentences

   - (1) He made me help .
   - (2) He gave me help .

   and the two tag sequences

   - (a) PRON VERB PRON NOUN PUNCT
   - (b) PRON VERB PRON VERB PUNCT

   1. Which tag sequence corresponds to which sentence? Justify your answer.
   2. Imagine you have the following sequence labeling models. They are all fully trained on a large corpus. Which of these models can assign the correct tag sequences to both sentences? Justify your answers.
      - A bigram HMM tagger with Viterbi decoding.
      - A trigram HMM tagger with Viterbi decoding.
      - A greedy logistic regression tagger with one-hot feature vectors and a window size of 3 (previous + current + next word).
      - A greedy logistic regression tagger with one-hot feature vectors and a window size of 5 (2 previous + current + 2 next words).
      - A greedy logistic regression tagger with word2vec embeddings of the current word.
      - A greedy logistic regression tagger with BERT embeddings of the current word. (8 pts)

Solution:

1. 1–b and 2–a. The easiest is to replace the word help by an unambiguous verb or noun, e.g. *He made me laugh* (laugh = VERB), *He gave me apples* (apples = NOUN).

2. Sequence labeling models:

   - Bigram HMM: no – when tagging *help*, the tagger only has access to the previous tag (PRON), which is the same in both cases.
   - Trigram HMM: no – when tagging *help*, the tagger only has access to the previous two tags (VERB, PRON), which are the same in both cases.
   - Greedy LR with window of 3: no – when tagging *help*, the tagger only has access to the previous tag (VERB) and the previous and next words (me, "."), which are the same in both cases.
   - Greedy LR with window of 5: yes – when tagging *help*, the tagger has access to two previous words (made me / gave me), which are different.

- Greedy LR with word2vec: no – the embedding of the current word is the same for both strings (help), and the tagger only has access to the previously predicted label (PRON), which is also the same in both cases.
- Greedy LR with BERT: yes – the embedding of the two occurrences of help is most likely different, so the tagger can predict different labels.

8. Assume the following simplified word vectors:

   - monkey: $[2, 2]$
   - elephant: $[3, 4]$
   - horse: $[4, 3]$

   Report the cosine similarities between *monkey* and *elephant*, and between *elephant* and *horse*. Which of the two words *monkey* or *horse* is more similar to *elephant*? (3 pts)

Solution:

$$cos(monkey, elephant) = \frac{6+8}{\sqrt{4+4} \cdot \sqrt{9+16}} = \frac{14}{2.83 \cdot 5} = 0.99$$

$$cos(horse, elephant) = \frac{12+12}{\sqrt{9+16} \cdot \sqrt{9+16}} = \frac{24}{5 \cdot 5} = 0.96$$

Monkey is more similar to elephant than horse.

9. There are several ways to create word vectors.

   - Describe one method that produces sparse vectors and one method that produces dense vectors.
   - What are the advantages and disadvantages of these two types of representations? (6 pts)

Solution: Sparse vectors: bag-of-words count-based vectors, TF-IDF (both can be used for term-document matrices and word-context matrices, but only the latter one is relevant here)

Dense vectors: Dimensionality-reduced count vectors (SVD, LSA, PCA), word2vec (SGNS, skipgram), FastText, BERT.

Advantages and disadvantages – aspects that can be mentioned:

- Sparse vectors are larger, therefore less efficient to include in models.
- The dimensionality for sparse vectors is determined by the data, while for dense vectors it can be freely chosen.
- In many cases, sparse vectors are derived deterministically, whereas creating dense vectors relies on a random component.
- Sparse vectors are interpretable, dense vectors are not.
- Dense vectors produce more "abstract" representations, e.g. synonyms are easier to detect with dense vectors.
- "Sparse" and "dense" are not counted as advantage/disadvantage without further explanation.

10. One of the main components of Transformer models is the **self-attention** mechanism.

    - Self-attention involves the three vectors $\mathbf{q}$ (query), $\mathbf{k}$ (key) and $\mathbf{v}$ (value). Describe the respective roles of these three vectors in a few sentences.
    - How does self-attention differ between **encoder** and **decoder** models? (5 pts)

Solution:

The target vector h_t for a given position t corresponds to a weighted average of the source vectors g_i of all positions i. Self-attention determines how much weight is given to each position.

The **value** vector is a linear projection of the source vector that determines the content that is being transferred from layer g to layer h. It is used in the same way as in a feed-forward NN.

The **key** and **query** vectors are also linear projections of the source vectors, but they are only used to compute the attention value. The attention value between two positions t and i is defined as the dot product of q_t and k_i. The query refers to the target position t, whereas the key refers to any other position in the sentence, called the context position.

Encoder self-attention covers the entire sequence, i.e. both the words to the left of the current position and the words to the right of the current position. Decoder self-attention only covers the words to the left of the current position (because the words to the right are not yet known at test time).

11. The Transformer uses so-called **multi-head attention**.

    - What does a head represent in this context?
    - What are, in your opinion, the advantages of using multiple heads compared to a single head? (3 pts)

Solution: A head refers to one "way" of attending the tokens of the sentence, i.e. a particular pair of key and query vectors.

Different heads can focus on different (linguistic) aspects. For example, one head can represent syntactic relations, another one can track co-reference, a third one can focus on more semantic relations. With a single head, all aspects would need to be dealt with by the same attention mechanism.

12. You are a researcher in bioinformatics and work on genome sequencing. Based on some genome data at your disposal, you have estimated a simple, letter-based bigram model that describes the probability of a DNA letter (A, T, C and G) following another DNA letter.

    The bigram model $P(x_t|x_{t-1})$ you have estimated is as follows:

$P(x_t \mid x_{t-1})$	Values for x_t:				
		A	**T**	**C**	**G**
Values for x_{t-1}:	**A**	0.1	0.4	0.3	0.2
	T	0.6	0.0	0.3	0.1
	C	0.9	0.0	0.0	0.1
	G	0.0	0.2	0.3	0.5

    You wish to use your bigram model to generate short genome sequences. More precisely, given a start letter **A**, you wish to generate the **three next letters**.

    1. Use **greedy decoding** to generate the three next letters following **A**, using the probabilities specified in the bigram model above. Indicate the probability of the sequence that was generated.

       Solution: For greedy decoding, we simply select the letter with highest transition probability according to the bigram model. Given **A**, the most likely next letter is therefore **T**, then **A**, then **T** again.
       The probability for the sequence **ATAT** is $0.4 \times 0.6 \times 0.4 = 0.096$.

    2. Use **beam search** (with beam size = 2) to generate the three next letters following **A** based on the bigram model. Show your calculations, and compute the probability of the best sequence you obtain.

       Solution: For beam search we keep a beam of K partial hypotheses of highest probability, and update this beam after each step. K is here set to 2.
       Given A, the two most likely continuations are **T** and **C**. The beam at that point is then $\{\textbf{AT} : 0.4, \textbf{AC} : 0.3\}$.
       We then evaluate $2 * 4$ potential continuations given this beam. The two most likely ones are **ATA** and **ACA**. The beam becomes $\{\textbf{ATA} : 0.24, \textbf{ACA} : 0.27\}$.
       Finally, the most likely continuations for the last letter are **ATAT** and **ACAT**, with corresponding probabilities 0.096 and 0.108. Beam search will therefore **ACAT** as output sequence, as it yields the highest probability.

    3. Use **top-K sampling** (with K=2) to generate the three next letters following **A**. You can "fake" the sampling by simply picking up manually one of the possible

choices. Describe your steps and compute the probability of the sequence you have sampled.

Solution: With top-K sampling, we restrict the sampling to the K values with highest probabilities. Given **A**, we can therefore sample either **T** or **C**, with probabilities of respectively $0.4/0.7$ and $0.3/0.7$.

Assuming the sampler selects **C**, the top-2 values at this state are then **A** and **G**, respectively with probabilities 0.9 and 0.1.

Assuming the sampler now selects **A**, the top-2 values at this state are again **T** or **C**, with probabilities of respectively $0.4/0.7$ and $0.3/0.7$.

Our sampled sequence is then **ACAT**, with probability 0.108.

Comment

As the sampling is faked, there will be multiple valid answers here.

4. Use **top-p sampling** (with p=50%) to generate the three next letters following **A**. You can again simulate the actual sampling by selecting one of the possible choices. Describe your steps and compute the probability of the sequence you have sampled. (14 pts)

Solution:

In top-p sampling, we consider the smallest set of most likely outcomes whose total probability mass meets or exceeds the threshold (50 % in this case). Given **A**, the two most likely outcomes **T** and **C** have a probability mass of 0.7, which means they will constitute the two values to sample from.

Assuming the sampler selects **T**, there is only one single value to sample from, **A**, whose probability exceeds by itself the threshold. The sampler will therefore have no other choice than selecting **A**.

For the final letter, we again have the choice between **T** and **C**. Assuming the sampler selects **T**, the final sequence becomes **ATAT**, with probability 0.096.

Comment

As the sampling is faked, there will be multiple valid answers here.

13. In the final obligatory assignment, you were asked to implement a talking elevator that could interact with human users and execute their requests to move between floors. More precisely, our goal was to implement a typical NLU-based pipeline, with an intent recognizer, a slot-filling module, and a rule-based response engine.

    The code skeleton for the talking elevator was as follows:

```
class TalkingElevator:
    def __init__(self, start_floor:int =1):
        self.cur_floor = start_floor
        ...
    def move_to_floor(self, floor_number : int):
        ...
    def stop(self):
        ...
    def process_input(self, user_input: str):
        ...
    def _say_to_user(self, system_response: str):
        ...
```

Now, let us assume that we wish to adopt an *end-to-end, generative approach* instead of this NLU pipeline, and re-implement the `process_input` method based on an instruction-tuned LLM such as Llama (used *as is*, without any domain-specific fine-tuning).

To keep things simple, we will simplify the functionalities that need to be covered. Your LLM-powered talking elevator will need to handle the following:

- If the user express a wish to go to floor X, the elevator should go to that floor.
- The user requests can also be relative, for instance *"Go one floor up"*.
- The elevator should provide *grounding feedback* to the user along with the physical movements.
- The elevator should also be able to inform the user about the current floor
- Finally, if the user asks the elevator to stop, the elevator should stop, and ask for clarification regarding the actual user intent.

Tasks:

1. Write the **prompt** that you would use for this talking elevator. The prompt may contain placeholders that can be filled in at runtime. You can also include some few-shot examples in your prompt if you find it useful.

   Solution: Here is a prompt that should work reasonably well:

You are a voice-controlled elevator for a building with 10 floors, and are currently on floor {CURRENT_FLOOR}.

You should output a JSON output with two keys:

(a) "actions" is a comma-separated list of actions to complete. The possible actions are either `go_to_floor(x)`, where `x` is a number between 1 and 10, or `stop()`.

(b) "response" is a string corresponding to what the elevator should answer back to the user.

The elevator should implement the following functionalities:

(a) If the user express a wish to go to floor X, the elevator should go to that floor, and say "OK, going to floor X".

(b) The user requests can also be relative, for instance "Go one floor up".

(c) The elevator should also be able to inform the user about the current floor

(d) Finally, if the user asks the elevator to stop, the elevator should stop, and ask for clarification regarding the actual user intent.

Examples:
User: "please go to the third floor"
Output: {"actions": "`go_to_floor(3)`", "response": "OK, going to the third floor"}

User: "now go one floor down"
Output: {"actions": "`go_to_floor(2)`", "response": "OK, going one floor down, to the second floor"}

User:"Where are we now?"
Output: {"actions": "", "response": "We are on the second floor"}

User: "Please stop!"
Output: {"actions": "`stop()`", "response": "Ok, stopping now. Do you wish to go to another floor?"}

Here is the previous dialogue history: "{DIALOGUE_HISTORY}".
And here is the current user input: "{USER_INPUT}"

Comment

There are many possibilities to write this prompt, but the following elements should be there one way or another:

(a) The prompt should indicate that the system should act like a talking elevator.

(b) The prompt should explain what the system should do (like the functionalities 1-4 above), although they can be formulated in many different ways

(c) The prompt should specify what kind of outputs are expected. In the prompt above, we use a JSON format as an example, but many alternatives exist (one can also have the system output a simple string, as long as one has a method for finding out what corresponds to a system action, and what corresponds to a verbal response). The format should allow the output to consist of both a system action and a verbal response at the same time

(d) The prompt should provide some in-context examples (few shots) of input-output pairs

(e) The prompt should specify what the current floor is, as it is necessary to answer questions such as ”where are we now” ?

(f) And obviously, the prompt should include the user input

We have added the dialogue history here in the prompt, but it is not strictly necessary.

2. Describe how this prompt would be employed as part the `process_input` method, and in particular what would need to happen *before* and *after* running the LLM with the prompt you have written. You don't need to write any code, just describe at a high level the main steps in your approach.

Solution: The prompt above is a template, which needs to be filled in with current values, notably the current floor, user input, and possibly the dialogue history.

After running the LLM and converting the output into a string, one needs to process the output format (JSON in our example) and extract what should be the response (if any) and what should be the system actions (if any). The system actions should result in a call to their corresponding methods, here `move_to_floor(self, floor_number)` and `stop(self)`), while the verbal response should be forwarded to the `_say_to_user` method.

14. Dialogue systems may be evaluated in several ways, such as:

    1. Through user surveys
    2. Through automated metrics
    3. Through an LLM-as-a-Judge method

    For *each* of those three approaches:

    - explain briefly (in 2-3 sentences) how that type of evaluation is conducted
    - name at least *one* benefit and *one* limitation of that type of evaluation. (9 pts)

Solution:

User surveys A simple method for evaluating the quality of the system's dialogue behaviour is to ask users to interact with the system, and ask them to rate their experience with the system through a questionnaire. The ratings will often relate to different aspects of the system performance, such as its ability to understand the user, respond something appropriate, whether it is verbose or not, etc.

Benefits: provides information about how real users perceive the system and their actual experience interacting with it.
Limitations: Data collection is expensive and time-consuming (need to recruit users and have them fill surveys). When developing a system, it cannot be used for every iteration of the dialogue system. Somewhat subjective.

Automated metrics An alternative is to apply automated metrics to measure the system performance. If we have a corpus of "reference" responses, we can compare the system output in a given context to those references, using metrics such as BLEU, ROUGE, BERTScore, etc. There are also reference-free metrics (ADEM, USR, etc.), which typically output scores that are optimised to correlate with human ratings, by identifying features that characterize "good responses". There also exists a wide range of metrics to assess the performance of specific modules within a dialogue system (such as precision/recall for intent recognition or WER for speech recognition).

Benefits: Can be applied automatically without the need to ask users to rate interactions (although one still needs dialogue data).
Limitations: Reference-based metrics may not correlate with actual human judgments. Reference-free metrics do not have this issue, but are often limited to specific domains/tasks. Module-specific metrics only measure the performance of a specific component, not the end-to-end performance of the system.

LLM-as-a-Judge Asks a LLM to judge the quality of a given response by the system. The LLM should preferably be different from the one employed to generate the response (if the dialogue system relies on an LLM). This assessment should be done on the basis of well-defined evaluation criteria.

Benefits: Can be applied automatically, and provide useful insights about the system performance. Can be applied to many types of dialogues.
Limitations: Large variations in the quality of the assessments, depending on the prompt, LLM used, and types of dialogue. No "calibration" of the quality scores (making it difficult to compare systems).

15. Compute the Word Error Rate between this ASR transcription:

    hello bot do you see and hiccup on the table uh

    And the actual, gold standard reference:

    hi robot do you see a big cup on the table (4 pts)

Solution:

We first need to compute the number of edits (insertions, deletions or substitutions). The easiest method is to simply find the minimal list of edits, like this:
hello $\rightarrow$ hi (substitution)
bot $\rightarrow$ robot (substitution)
a $\rightarrow$ and (substitution)
$\emptyset \rightarrow$ big (insertion)
hiccup $\rightarrow$ cup (substitution)
uh $\rightarrow \emptyset$ (deletion)

Which gives us a total of 6 edits.

To compute the Word Error Rate, we then simply divide the number of edits by the number of words in the reference, and multiply by 100 to get a percentage:

$$100 \times \frac{6}{11} = 54.55 \quad (1)$$

The students may also determine the number of edits by writing down an edit-distance matrix (this is entirely optional of course) :

	hello	bot	do	you	see	and	hiccup	on	the	table	uh	
	0	1	2	3	4	5	6	7	8	9	10	11
hi	1	1	2	3	4	5	6	7	8	9	10	11
robot	2	2	2	3	4	5	6	7	8	9	10	11
do	3	3	3	2	3	4	5	6	7	8	9	10
you	4	4	4	3	2	3	4	5	6	7	8	9
see	5	5	5	4	3	2	3	4	5	6	7	8
a	6	6	6	5	4	3	3	4	5	6	7	8
big	7	7	7	6	5	4	4	4	5	6	7	8
cup	8	8	8	7	6	5	5	5	5	6	7	8
on	9	9	9	8	7	6	6	6	5	6	7	8
the	10	10	10	9	8	7	7	7	6	5	6	7
table	11	11	11	10	9	8	8	8	7	6	5	6

16. Explain the concept of "predictive parity" when talking about algorithmic fairness. Illustrate your explanation with a short example. (5 pts)

Solution: Let us assume a population of individuals distinguished by a sensitive attribute A, such as race or gender. We wish to build a system to make predictions $\hat{Y}$ about those individuals given some input vector $\mathbf{X}$, and we have access to a database of actual outcomes for a set of individuals, allowing us to compare the predictions of a given model to those actual outcomes.

Predictive parity is achieved if the *precision* of our model predictions are the same across both groups:

$$P_{A=0}(Y = y|\hat{Y} = y) \approx P_{A=1}(Y = y|\hat{Y} = y) \tag{2}$$

Example:
Let's consider a predictive model used to predict whether individuals are likely to default on a loan. We examine two groups of applicants (A=0 or A=1).

Predictions:
For Group A=0 (say, males): The model predicts that 100 individuals will default on their loan.
For Group A=1 (say, females): The model predicts that 80 individuals will default on their loan.

Actual Outcomes:
In reality, out of the 100 individuals predicted to default in Group A, 90 actually default.
In reality, out of the 80 individuals predicted to default in Group B, 72 actually default.

For A=0: The probability of actual default given a predicted default = 90/100 = 90%
For A=1: The probability of actual default given a predicted default = 72/80 = 90%
Here, both groups have a 90% chance of defaulting when the model predicts they will default. Therefore, the model achieves predictive parity.