Stockholm
University

Classification in NLP

Thomas Vakili

February 5, 2026 — Stockholm University

Agenda

1. Introducing classification
2. Implementing a Naïve Bayes classifier
3. Evaluating classifiers
4. LLMs for classification

Stockholm University

Text classification

What is text classification?

- Text classification is used to assign a *class* to a *document*
- Typically implemented using *supervised machine learning*

To create a classifier, we want to find a *function*

$$f(x) : X \rightarrow Y$$

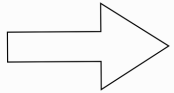
where X is the domain of *inputs* and Y the classes that can be assigned to them.

Example: ICD-10 coding

Patient from Åre was treated with antacids by

Dr. Lundvall at Södersjukhuset for an ulcer

after presenting with stomach pain on Jan 12th



ICD-10 Classification

K20 - K31

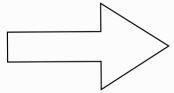
Diseases of esophagus, stomach and duodenum

Example: ICD-10 coding

Patient from Åre was treated with antacids by

Dr. Lundvall at Södersjukhuset for an ulcer

after presenting with stomach pain on Jan 12th



ICD-10 Classification

K20 - K31

Diseases of esophagus, stomach and duodenum

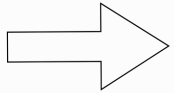
- $x \in X$ is any **electronic health record**

Example: ICD-10 coding

Patient from Åre was treated with antacids by

Dr. Lundvall at Södersjukhuset for an ulcer

after presenting with stomach pain on Jan 12th



ICD-10 Classification

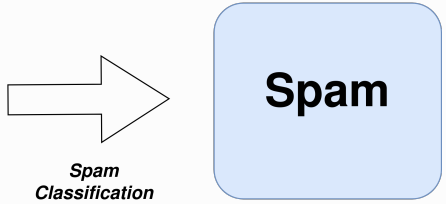
K20 - K31

Diseases of esophagus, stomach and duodenum

- $x \in X$ is any **electronic health record**
- Y is the set of ICD-10 code blocks (multiclass classification)

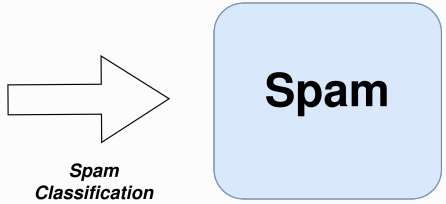
Example: spam classification

Esteemed Dr Prof Vakili,
Greetings!! From the Journal of Dentistry.
We are impressed by your recent publication
and kindly request that you submit a paper.



Example: spam classification

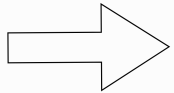
Esteemed Dr Prof Vakili,
Greetings!! From the Journal of Dentistry.
We are impressed by your recent publication
and kindly request that you submit a paper.



- $x \in X$ is any **email**

Example: spam classification

Esteemed Dr Prof Vakili,
Greetings!! From the Journal of Dentistry.
We are impressed by your recent publication
and kindly request that you submit a paper.



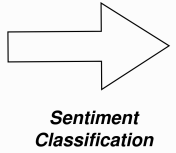
Spam Classification

Spam

- $x \in X$ is any **email**
- Y is either **spam** or **not spam** (binary classification)

Example: (binary) sentiment classification

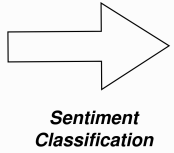
The room was noisy, the food was bland, and the service was abysmal. They did not even serve Gränges?! Will not come back.



Negative

Example: (binary) sentiment classification

The room was noisy, the food was bland, and the service was abysmal. They did not even serve Gränges?! Will not come back.

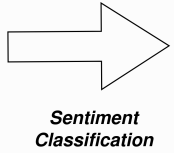


Negative

- $x \in X$ is any **review**

Example: (binary) sentiment classification

The room was noisy, the food was bland, and the service was abysmal. They did not even serve Gränges?! Will not come back.

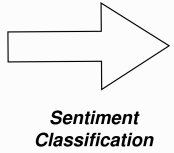


Negative

- $x \in X$ is any **review**
- Y is either **positive** or **negative** (binary classification)

Example: (binary) sentiment classification

The room was noisy, the food was bland, and the service was abysmal. They did not even serve Gränges?! Will not come back.



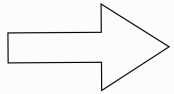
Negative

In NLP, X is the *infinite* set of all possible sentences. Y is *finite* and will depend on the task and domain.

$$|Y| = |\{\text{positive, negative}\}| = 2$$

Example: (binary) sentiment classification

The room was noisy, the food was bland, and the service was abysmal. They did not even serve Gränges?! Will not come back.



Sentiment Classification

Negative

In NLP, X is the *infinite* set of all possible sentences. Y is *finite* and will depend on the task and domain.

$$|Y| = |\{\text{positive, negative}\}| = 2$$

$$|X| = |\{\text{"This bar is great", "I'm missing Gränges", ...}\}| = \infty$$

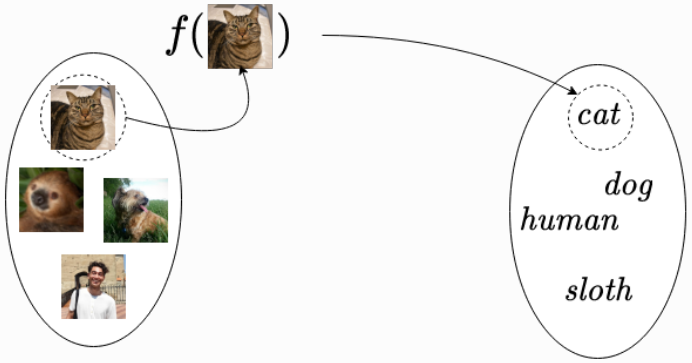
Stockholm University

Building classifiers

Classification revisited

Remember

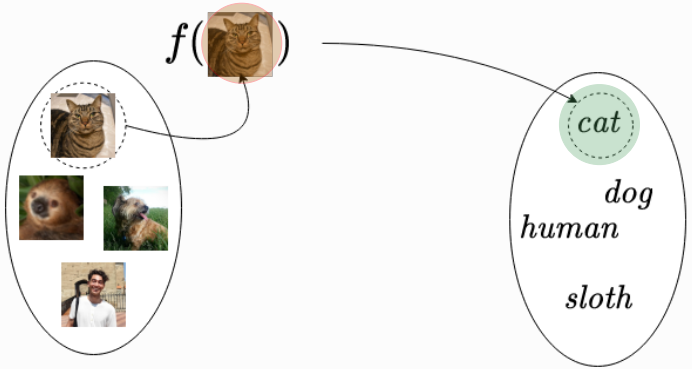
Sentiment analysis is a document classification task! We want $f : X \rightarrow Y$ that maps from **inputs** $x \in X$ to **labels** $y \in Y$



Classification revisited

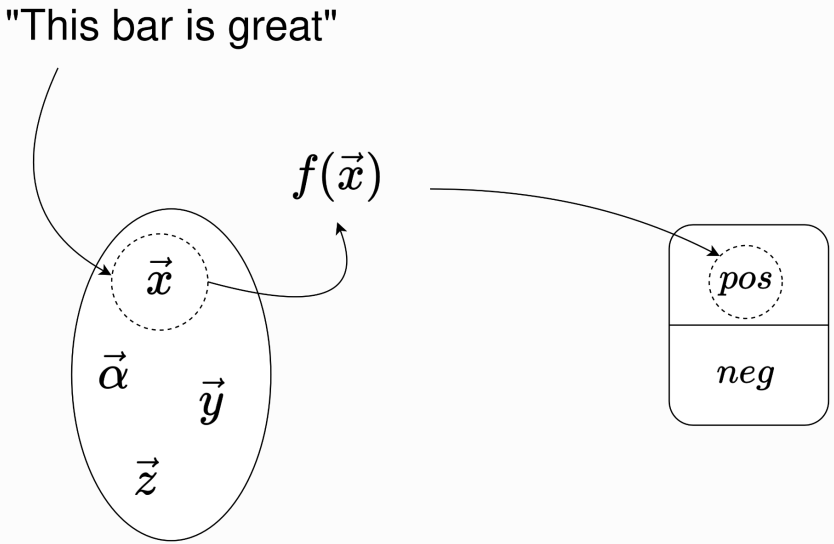
Remember

Sentiment analysis is a document classification task! We want $f : X \to Y$ that maps from **inputs** $x \in X$ to **labels** $y \in Y$

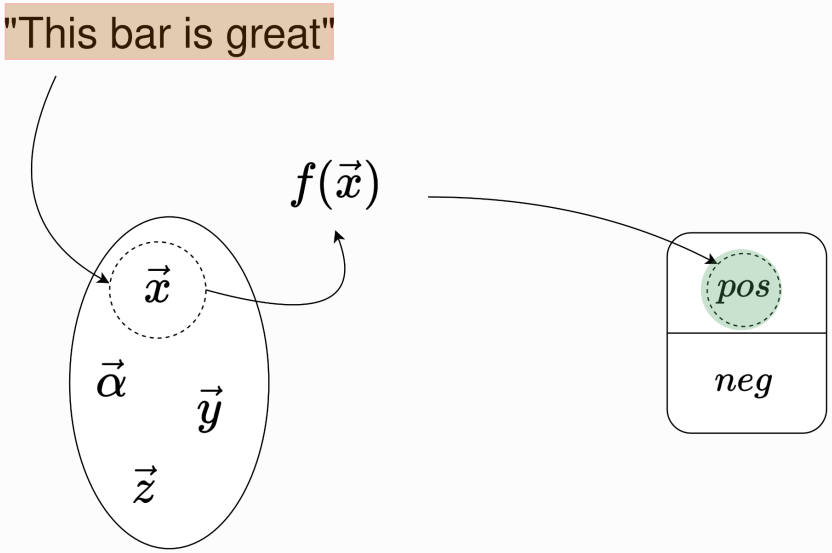


f : images $\to$ animal types

Text classification

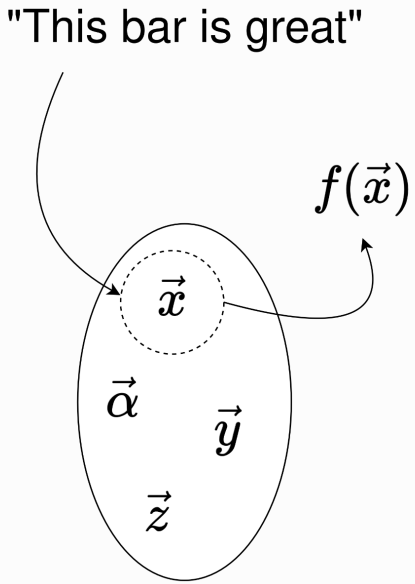


Text classification



$$f : reviews \rightarrow sentiments$$

Vectorization



We need a good way to transform our input text to a **vector**

Vectorizing *sequences*

Word2Vec

We can compute the *average* of a sentence's word vectors (lecture 4)

Transformer models

Modern LLMs can build contextual representations (also lecture 4)

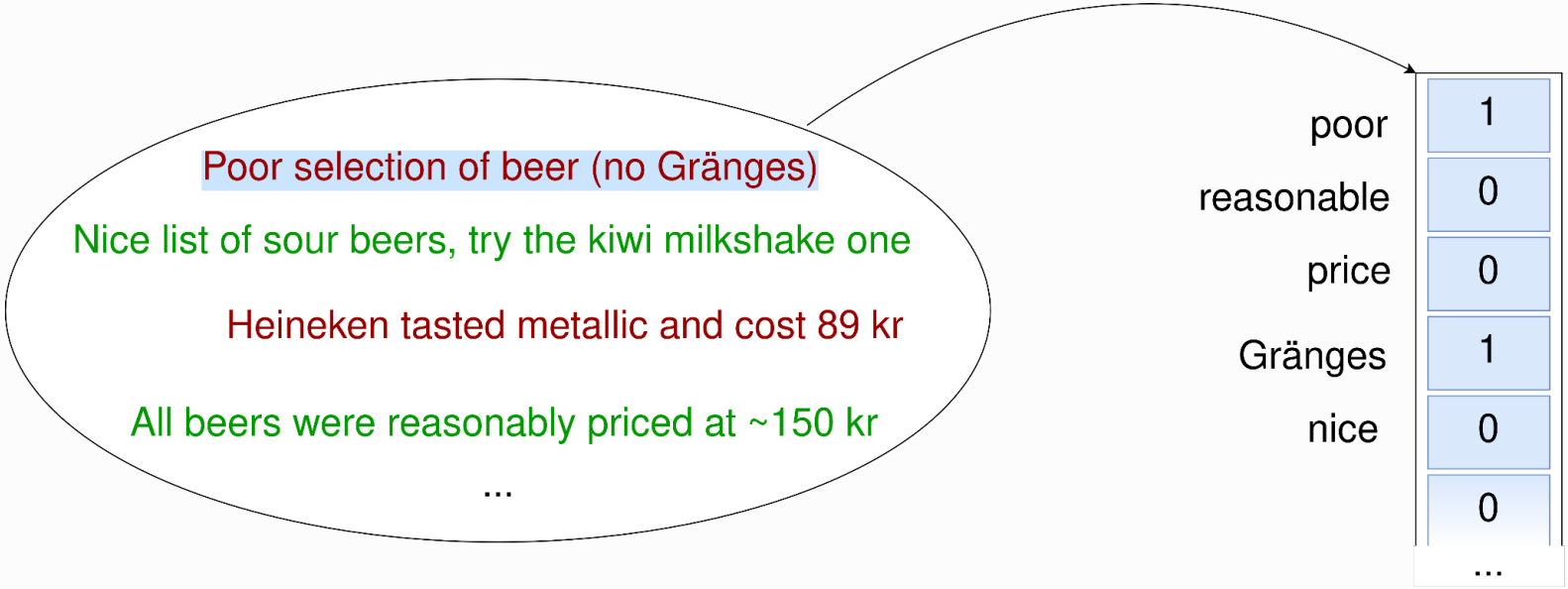
Count vectorization

Sometimes, we can keep it simple!

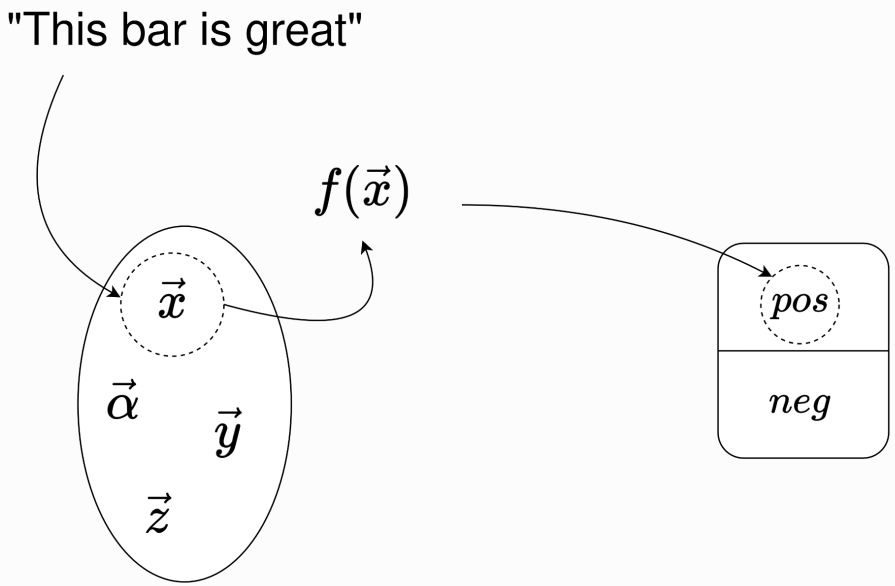
Bag-of-words (BoW) models

- Bag-of-words (BoW) models are simple but often effective classifiers
- Consider all words in a document, ignoring order (BoW assumption)
- Pre-processing can be applied (stemming, stop word removal, etc)
- Words can be represented by count, weights, as binary features, etc

Assigning indices



So how do we classify?



Text classification with Naïve Bayes

Naïve Bayes is a probabilistic classifier: $\hat{y} = \arg\max_{y \in Y} P(y \mid x)$

Select the *class* $y \in Y$ with the highest *posterior probability* given a document x

Text classification with Naïve Bayes

Naïve Bayes is a probabilistic classifier: $\hat{y} = \arg\max_{y \in Y} P(y \mid x)$

Select the *class* $y \in Y$ with the highest *posterior probability* given a document x
We can find $P(y \mid x)$ using Bayes' rule:

$$P(y \mid x) = \frac{P(x \mid y) \cdot P(y)}{P(d)}$$

Text classification with Naïve Bayes

Naïve Bayes is a probabilistic classifier: $\hat{y} = \arg\max_{y \in Y} P(y \mid x)$

Select the *class* $y \in Y$ with the highest *posterior probability* given a document x
We can find $P(y \mid x)$ using Bayes' rule:

$$P(y \mid x) = \frac{P(x \mid y) \cdot P(y)}{P(d)}$$

$P(x \mid y)$ is the likelihood of a document given a class

$P(y)$ is the prior class probability

$P(x)$ is the probability of a document

The easy part: $P(y)$

We don't even need to process the text to get $P(y)$:

N_{doc} is the number of training documents
N_y is the number of documents belonging to class y in the training data

That's all we need:

$$P_y = \frac{N_y}{N_{doc}}$$

Estimating $P(x \mid y)$

How do we estimate $P(x \mid y)$? We need to process the data

- Conditional probability of a single word: $P(w_i \mid y) = \frac{count(w_i,c)}{\sum_{w \in V} count(w,c)}$
- We can merge all documents by class and count word frequencies

What about full documents?

BoW simplification

We ignore word order:

$$P(\text{this book was really amazing}) = P(\text{really this book amazing was}) = \ldots$$

Naïve simplification

Assume all words are independent, i.e.

$$P(\text{this amazing book} \mid POS) = P(\text{this} \mid POS) \cdot P(\text{amazing} \mid POS) \cdot P(\text{book} \mid POS)$$

Last piece of the puzzle: $P(x)$

What is the probability of the document?

$$P(x) = P(x \mid y) + P(x \mid \neg y)$$

- $P(x)$ does not depend on the class y

Last piece of the puzzle: $P(x)$

What is the probability of the document?

$$P(x) = P(x \mid y) + P(x \mid \neg y)$$

- $P(x)$ does not depend on the class y
- We are not interested in actually finding $P(y \mid x)$ but finding the *argmax*

$$\hat{y} = \arg\max_{y \in Y} P(y \mid x)$$

Last piece of the puzzle: $P(x)$

What is the probability of the document?

$$P(x) = P(x \mid y) + P(x \mid \neg y)$$

- $P(x)$ does not depend on the class y
- We are not interested in actually finding $P(y \mid x)$ but finding the *argmax*

$$\hat{y} = \arg\max_{y \in Y} P(y \mid x) = \arg\max_{y \in Y} \frac{P(x \mid y) \cdot P(y)}{P(x)}$$

Last piece of the puzzle: $P(x)$

What is the probability of the document?

$$P(x) = P(x \mid y) + P(x \mid \neg y)$$

- $P(x)$ does not depend on the class y
- We are not interested in actually finding $P(y \mid x)$ but finding the *argmax*

$$\hat{y} = \arg\max_{y \in Y} P(y \mid x) = \arg\max_{y \in Y} \frac{P(x \mid y) \cdot P(y)}{P(x)} = \arg\max_{y \in Y} P(x \mid y) \cdot P(y)$$

Unseen words

Remember how we calculate $P(x \mid y)$

$$P(\text{this amazing book} \mid POS) = P(\text{this} \mid POS) \cdot P(\text{amazing} \mid POS) \cdot P(\text{book} \mid POS)$$

But what about $P(\text{this unseen book} \mid POS)$?

$$count(\text{unseen} \mid POS) = 0 \implies P(\text{this unseen book} \mid POS) = 0$$

Laplace smoothing

Simple solution: add 1 to all word counts

Laplace smoothing

With Laplace smoothing, we get:

$$P(w_i \mid y) = \frac{count(w_i, c) + 1}{\sum_{w \in V} (count(w, c) + 1)} = \frac{count(w_i, c) + 1}{\sum_{w \in V} count(w, c) + |V|}$$

This way, no word will have zero probability!

Final optimization

Perform calculations in log-space to make sure we don't run into underflow due to tiny numbers. Instead of:

$$P(w_1, w_2, ..., w_m \mid y) = \prod_{i=1}^{m} P(w_i \mid y)$$

we can look at:

$$\log P(w_1, w_2, ..., w_m \mid y) = \sum_{i=1}^{m} \log P(w_i \mid y)$$

(because we are looking for the *argmax*)

Stockholm University

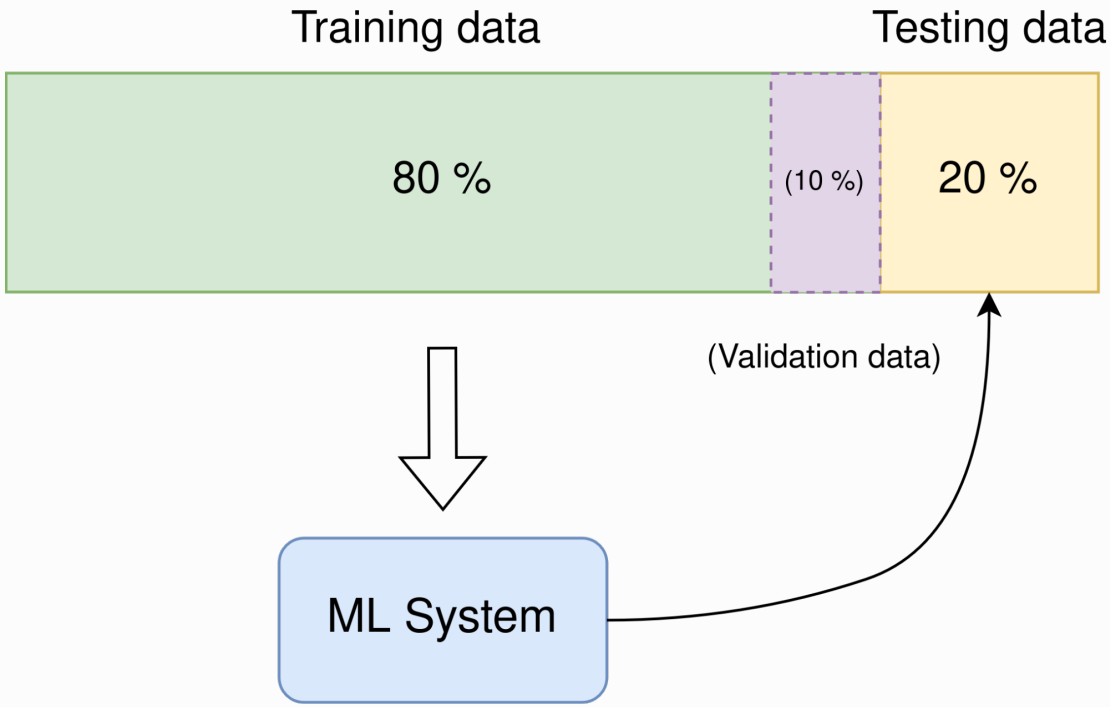
Evaluating classifiers 101

Warning: repetition ahead

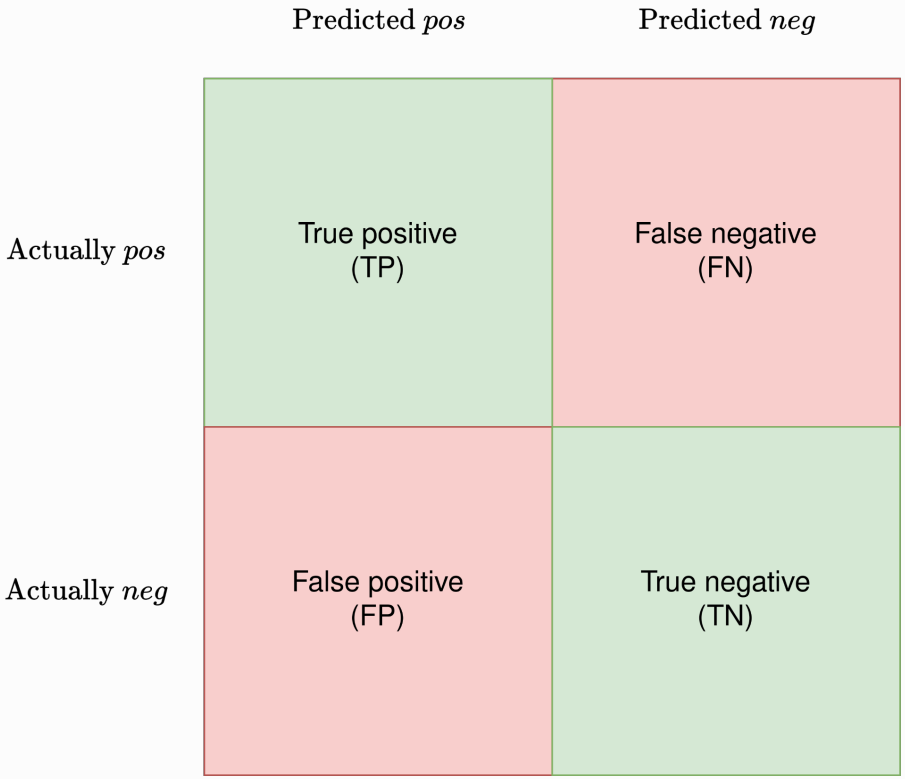
You've probably (?) taken an introductory machine-learning course.

If so, this will be a bit of (healthy?) repetition.

Train-test(-val) split



Different types of errors



The four metrics

The four most important metrics in NLP:

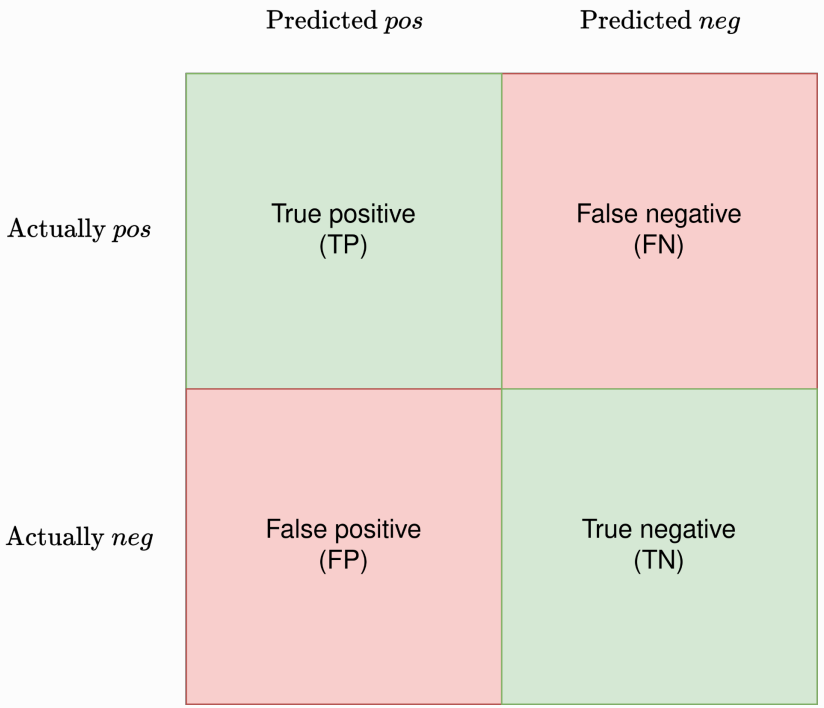
- $\text{Accuracy} = \dfrac{\text{TP} + \text{FP}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}}$
- $\text{Precision} = \dfrac{\text{TP}}{\text{TP} + \text{FP}}$
- $\text{Recall} = \dfrac{\text{TP}}{\text{TP} + \text{FN}}$
- $\text{F}_\beta = (1+\beta^2) \cdot \dfrac{\text{Precision} \cdot \text{Recall}}{\beta^2 \cdot \text{Precision} + \text{Recall}}$

	Predicted *pos*	Predicted *neg*
Actually *pos*	True positive (TP)	False negative (FN)
Actually *neg*	False positive (FP)	True negative (TN)

The four metrics

The four most important metrics in NLP:

- $\text{Accuracy} = \dfrac{\text{TP} + \text{FP}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}}$
- $\text{Precision} = \dfrac{\text{TP}}{\text{TP} + \text{FP}}$
- $\text{Recall} = \dfrac{\text{TP}}{\text{TP} + \text{FN}}$
- $\text{F}_1 = 2 \cdot \dfrac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$



Tricked by accuracy?

	Predicted +	**Predicted -**
Actual +	TP = 2	FN = 8
Actual -	FP = 18	TN = 972

$$\text{Accuracy} = \frac{2 + 972}{1000} = 97.4\%$$

Tricked by accuracy?

	Predicted +	**Predicted -**
Actual +	TP = 2	FN = 8
Actual -	FP = 18	TN = 972

$$\text{Accuracy} = \frac{2 + 972}{1000} = 97.4\%$$

$$\text{Precision} = \frac{2}{2 + 18} = 10\% \quad \text{Recall} = \frac{2}{2 + 8} = 20\%$$

Tricked by accuracy?

	Predicted +	**Predicted -**
Actual +	TP = 2	FN = 8
Actual -	FP = 18	TN = 972

$$\text{Accuracy} = \frac{2 + 972}{1000} = 97.4\%$$

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = 2 \cdot \frac{0.1 \cdot 0.2}{0.1 + 0.2} = 0.1\overline{333}$$

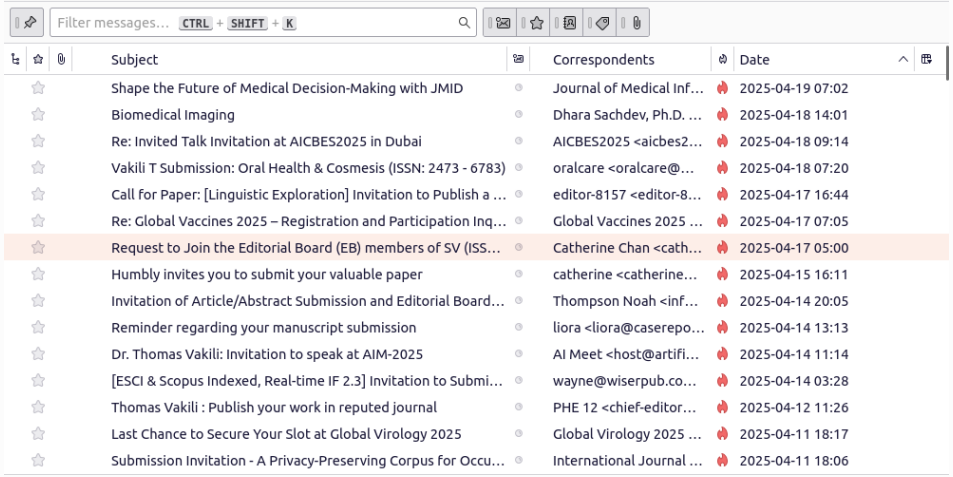
DEMO!

Stockholm University

LLMs for classification

What about LLMs?

- In the last two lectures, we introduced LLMs such as BERT
- BERT is more powerful, but Naïve Bayes is much faster (and does quite well)



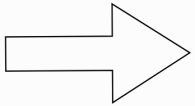
What about LLMs?

- In the last two lectures, we introduced LLMs such as BERT
- BERT is more powerful, but Naïve Bayes is much faster ~~(and does quite well)~~

Patient from Åre was treated with antacids by

Dr. Lundvall at Södersjukhuset for an ulcer

after presenting with stomach pain on Jan 12th



ICD-10 Classification

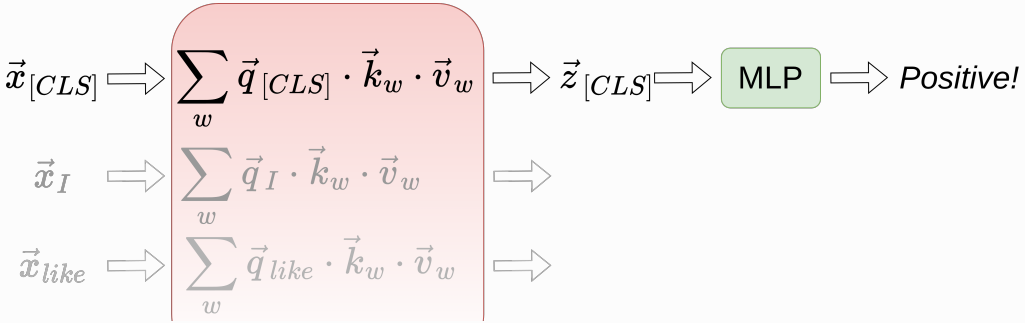
K20 - K31

Diseases of esophagus, stomach and duodenum

LLM classification

For some tasks, simpler models just won't do. We need more powerful models that can capture context and semantics: LLMs.

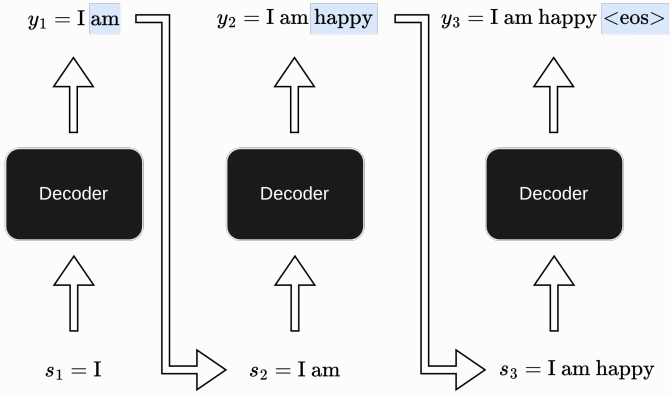
- **Encoder** models create contextual embedding vectors that can be classified



LLM classification

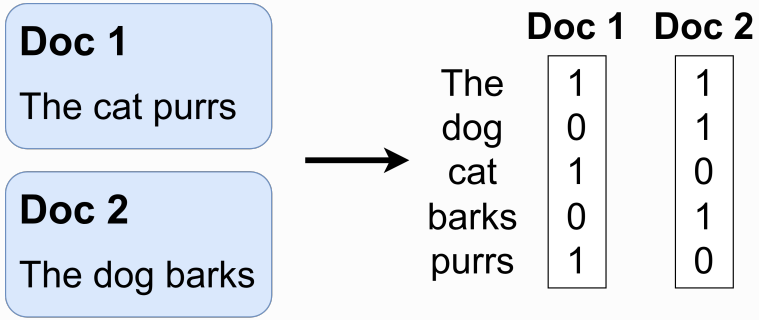
For some tasks, simpler models just won't do. We need more powerful models that can capture context and semantics: LLMs.

- **Encoder** models create contextual embedding vectors that can be classified
- **Decoder** models classify by continuing a token sequence (i.e., autocomplete)



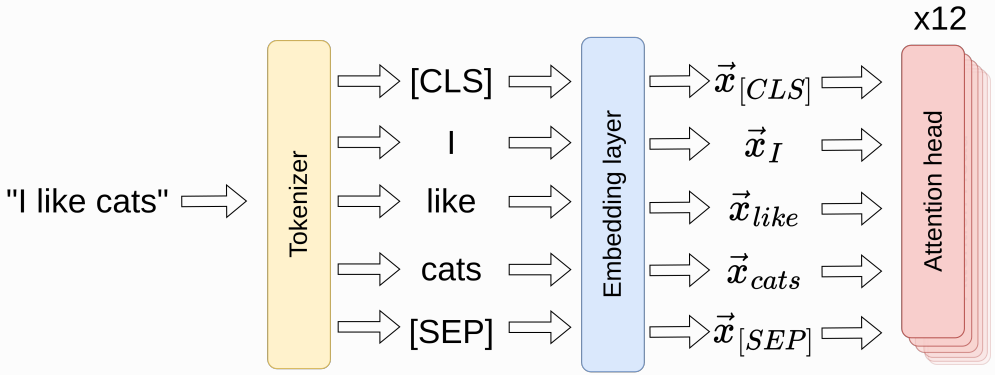
Encoder models

- Encoders classify *vector representations* of documents
- These vectors are called *embeddings*
- Classic methods like Naïve Bayes also rely on vector representations



Encoder models

- Encoders classify *vector representations* of documents
- These vectors are called *embeddings*
- Classic methods like Naïve Bayes also rely on vector representations but encoder models produce vector representations using *neural networks*



Training an LLM – in practice

- LLMs are implemented as deep neural networks
- Today, we'll introduce the how to implement an already-existing classifier

DEMO!

Classifying with generative LLMs

- Generative LLMs have become really popular
- Natural language generation (NLG) used to be very niche
- Increasingly, classification is treated as an NLG task
- But next-token prediction *is itself* a classification task

Next-token prediction

Generative LLMs functionally the same as autocorrect systems. We want to find a token $t_{i+1} \in V$ such that:

$$t_{i+1} = \arg\max_{v \in V} P\left(v \mid t_i, t_{i-1}, \dots\right)$$

This is a classification task, predicting the next token based on the previous tokens.

$$f : \text{sequence of tokens} \rightarrow \text{token}$$

Next-token prediction

Generative LLMs functionally the same as autocorrect systems. We want to find a token $t_{i+1} \in V$ such that:

$$t_{i+1} = \arg\max_{v \in V} P\left(v \mid t_i, t_{i-1}, \dots\right) *$$

This is a classification task, predicting the next token based on the previous tokens.

$$f : \text{sequence of tokens} \rightarrow \text{token}$$

$*$ In practice, we *sample* from the distribution rather than taking the arg max

Generative models for classification

Using next-token prediction to perform classification is a bit circular, but it has some benefits:

- The largest LLMs are (currently) generative
- You (usually) don't need to spend computational power for fine-tuning
- Consequently, you also don't need any training data
  - Although, you might want some for few-shot prompting
  - And also for evaluation (!)

Problem

Perform sentiment analysis. Based on the input text, answer whether the input is NEGATIVE or POSITIVE. Examples: “This movie is great” => POSITIVE. “I hate this t-shirt” => NEGATIVE.

Input: "I wish they had added more salt to the stew."

The input "I wish they had added more salt to the stew." conveys a slightly negative sentiment. Therefore, the sentiment analysis result is NEGATIVE.

Solution?

We don't need to sample from *all* $v \in V$. We can restrict the sampling to *only* consider tokens that relate to our task.

Instead of $v \in V$ we can sample from $v \in \{\text{negative}, \text{positive}\}$. This idea is called *structured prediction*.

Example: vLLM

```python
from vllm import LLM, SamplingParams
from vllm.sampling_params import StructuredOutputsParams

llm = LLM(model="HuggingFaceTB/SmolLM2-1.7B-Instruct")

structured_outputs_params = StructuredOutputsParams(choice=["Positive", "Negative"])
sampling_params = SamplingParams(structured_outputs=structured_outputs_params)
outputs = llm.generate(
    prompts="Classify this sentiment: vLLM is wonderful!",
    sampling_params=sampling_params,
)
print(outputs[0].outputs[0].text)
```

Stockholm University