Human-AI Interaction

1.1

The level of agreement between annotators can be calculated using the Cohen Kappa coefficient. This coefficient measures the level of agreement between annotators and it is given by:

$$k = \frac{p_o - p_e}{1 - p_e}$$

- p_o: Observed agreement (percentage of times they agreed $\frac{|\text{agreements}|}{N}$).
- p_e: Chance agreement (probability they would agree randomly based on their selection frequencies). This is given by $p_e = \frac{1}{N^2} \sum_k n_{k_1} n_{k_2}$.
  - N: Number of labeled items.
  - n_{k_i}: Number of times annotator i labelled category k.
  - There are $|k|$ categories.

In this case:

- $N = 16$
- $p_o = \frac{13}{16} \approx 0.81$
- $p_e = \frac{1}{16^2}(4 \times 3 + 12 \times 13) \approx 0.66$
- $k = \frac{0.81 - 0.66}{1 - 0.66} \approx 0.44$

1.2

Human-centered AI is a framework that aims to use AI as an enhancer of human performance rather than replacing humans.

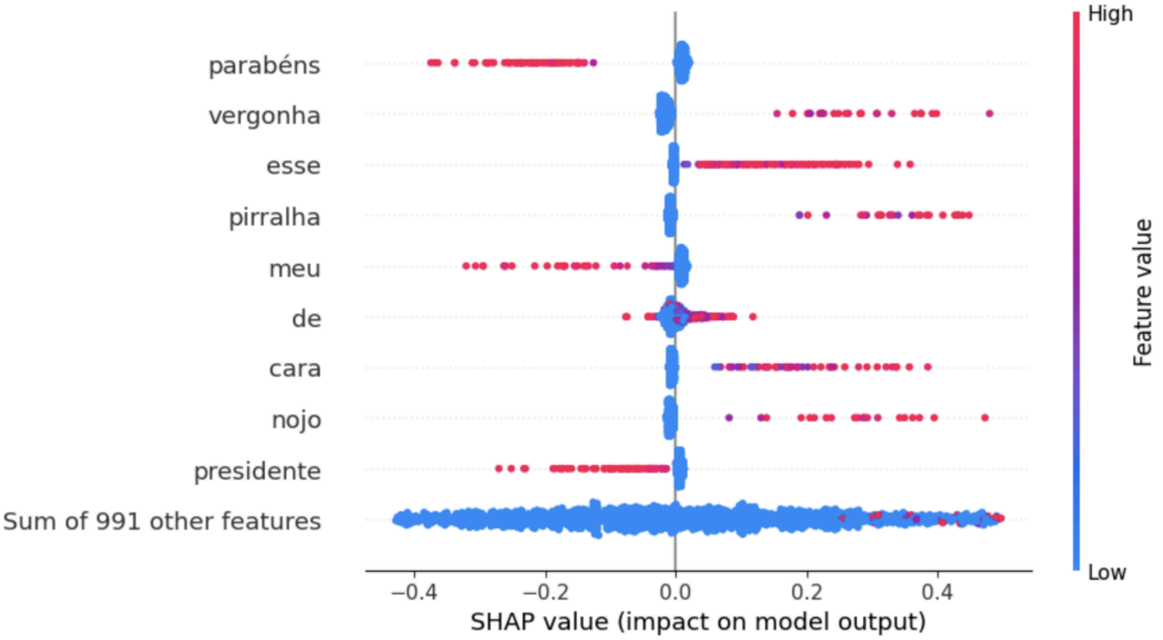
- **Supervisor / Annotator:** Label training samples for supervised learning.
- **Validator / Judge:** Evaluate the performance of the AI model. They can give feedback to the model through Reinforcement Learning with Human Feedback (RLHF).
- **Collaborator / Co-creator:** Humans work alongside the models to make decisions (e.g., medical diagnosis, pilot document review, risk assessment). Humans can also use AI as a creative partner.
- **Decision-maker / Operator:** Humans act as the final operators who oversee the system and make the ultimate decisions based on AI suggestions.
- **Developers:** Designing an AI system, defining its rules and how it learns.
- **Producers of Data:** Humans produce massive amounts of data.
  - **Explicit:** Comments, posts, likes, shares, tagging, classification, feedback, surveys, money transactions, ...

- **Implicit:** Activity patterns (web, smartphone, wearables), other IoT sensors, ...

1.3

These are models that aid in interpreting the black-box responses of the AI models without directly analyzing the relationship between the model's inputs and outputs or modifying internal parameters or architecture.

- **LIME (Local Interpretable Model-agnostic Explanations):**
  1. Interpret a decision by generating a dataset with perturbed samples (variation on the instance of interest).
     - For images, it might grey out big chunks of the image. For text, it may remove words. For tabular data, it might add noise to numerical features, ...
  2. Record the black-box predictions for those perturbed samples to see how the outputs change.
     - For example, if removing the word "bad" changes the sentiment from negative to positive, LIME notes this high impact.
  3. LIME calculates the distance of the perturbed samples to the original inputs. Samples that look more like the original input are given higher weight.
  4. Trains a simple interpretable model (e.g., Decision Tree or Linear Regression) on this new weighted dataset. The new simple model learns to approximate the complex model's behavior on that specific local area.
- **SHAP (SHapley Additive exPlanations):** SHAP treats the model's features as "players" in a game and the model's predictions as the "payout". The goal is to fairly distribute the payout (difference between the actual prediction and the average prediction of the dataset) among the players (features).
  1. It creates various combinarions ("coalitions") of features.
     - For example, if we have features A, B, and C, it tests the model with {A, B}, {B, c}, {A}, ...
  2. It calculates how much the prediction changes when a specific feature is added to a coalition ("marginal contribution").
  3. The final SHAP value for a feature is the weighted average of its marginal contributions across all possible combinations.
  4. We can then plot the SHAP values to see how much a feature contributes to the model's output



- **Pros and Cons:**
    - SHAP is more consistent, if a model is changed so that it relies more on a specific feature, the SHAP value for that feature will not decrease, unlike LIME, which does not guarantee that.
    - Calculating exact SHAP values is NP-hard, but there are approximations. Generally, LIME is faster.

1.4

The quality of the AI system is influenced by the quality of the data annotation (in supervised machine learning).
We can get large amounts of data through crowdsourcing. This outsources the labeling of data to a distributed group of people instead of just a few experts.
The "Wisdom of Crowds" concept states that the knowledge of that distributed group can often be as accurate (or even better) than the few experts.

- **Pros:**
    - Scalability, many people are available to work at once.
    - Low costs.
- **Cons:**
    - Lower quality / credibility (also due to low payment)
    - Risky for complex tasks.
    - More effort required to manage collaborators and guarantee quality.
    - Work ethics (this is not regulated work).
- **How to mitigate problems:**
    - Use aggregation method such as majority voting or weighted average to filter noise.

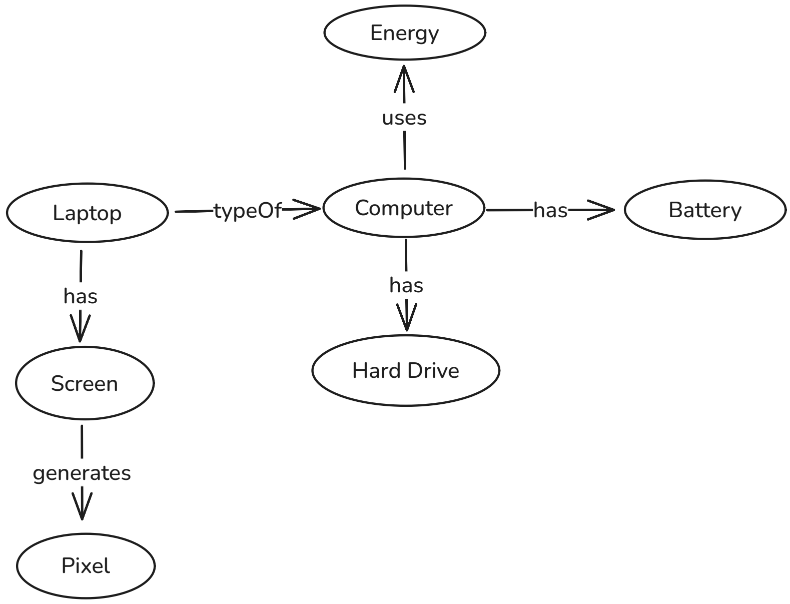
The Kappa coefficient measures inter-annotator agreement to ensure that the data is reliable.

Knowledge Representation

2.1

- Some students took MAS in 2019.
  - \exists_{x}&nbsp;(\text{Student}(x) \land \text{Take}(x,\text{MAS},2019))
- Every student who takes KL passes it.
  - $\forall_{x,y}(\text{Student}(x) \land \text{Take}(x, \text{KL}, y) \implies \text{Passes}(x, \text{KL}, y))$
- In 2024, there was a course taken by a single student.
  - $\exists_{x,y}(\text{Course}(x) \land \text{Student}(y) \land \text{Take}(y, x, 2024) \land \forall_z(z \neq y \Rightarrow \neg\text{Take}(z, x, 2024)))$
- The best score in KL is always higher than the best score in MAS.
  - $\forall_x \exists_y \text{Score}(x, \text{MAS}) < \text{Score}(y, \text{KL})$

2.2



2.3

- **Ontological Commitments:** What a logic assumes that exists to make its sentences true.
- **Propositional Logic:**
  - Assumes that the world consists of facts that are true, false, or unknown.
  - Treats the world as a set of fixed statements (propositions).
  - It's ontologically blind to individual objects (e.g., "Socrates is mortal" is just a single variable P, the existance of "Socrates" and

the property "mortal" do not exist as separate entities).

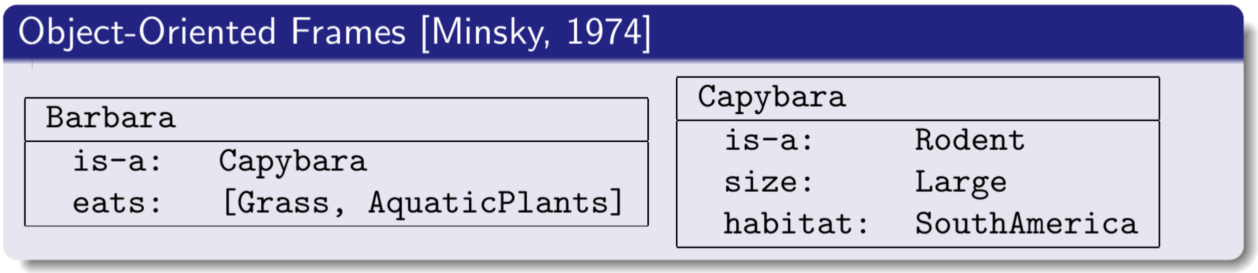
- As a consequence, we have to create a unique rule for every single student
  - $\text{Student}_1 \rightarrow \text{TakesCourse}$
  - $\text{Student}_2 \rightarrow \text{TakesCourse}$

- **First-Order Logic:**
  - Assumes a richer world.
  - Has **objects** (e.g., people, numbers, houses).
  - Has **relations** between the objects
    - Properties (e.g., red, round, ...)
    - n-ary (e.g., brother of, bigger than, owns, ...)
  - As a consequence, we can write a generic rule that applies to any object in the domain
    - $\forall_x(\text{Student}(x) \rightarrow \exists_y(\text{Course}(y)) \land \text{Takes}(x, y))$

2.4

Frame-based representation allow for a hierarchical representation of knowledge. It can use inheritance and default assumptions to fill information gaps.
Frames are structured by defining classes (concepts) and individuals (instnace) linked through "is-a" relationships.

- **Object-Oriented Frames:** They represent entities or attributes.



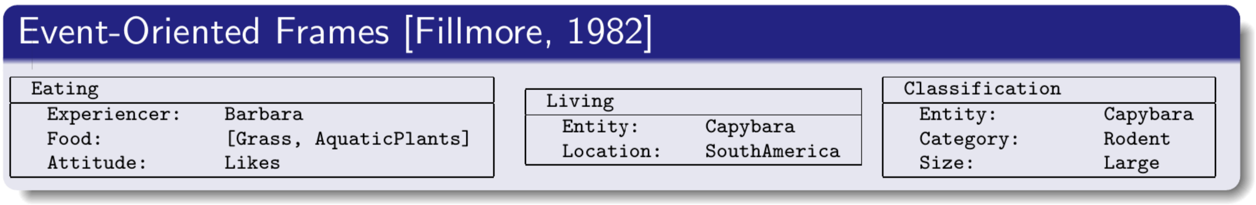
Object-Oriented Frames [Minsky, 1974]

Barbara	
is-a:	Capybara
eats:	[Grass, AquaticPlants]

Capybara	
is-a:	Rodent
size:	Large
habitat:	SouthAmerica

  - Here, the "Capybara" frame defines stereotypical attributes for that class. Even though "Barbara" does not have an explicit "size" attribute, it inherits that attribute from the "Capybara" frame.

- **Event Frames:** These organize knowledge by defining stereotypical situations or actions.



Event-Oriented Frames [Fillmore, 1982]

Eating	
Experiencer:	Barbara
Food:	[Grass, AquaticPlants]
Attitude:	Likes

Living	
Entity:	Capybara
Location:	SouthAmerica

Classification	
Entity:	Capybara
Category:	Rodent
Size:	Large

  - Here, for example, if an "Eating" event is detected, we know that there must be an "Experiencer" and "Food". So we can organize unstructured observations into a coherent narrative.

2.5

The vocabulary just lists terms. The ontology takes the vocabulary terms and forms a set of rules by constraining the possible interpretations of terms, defining classes, individuals and properties.

To enable automatic inference by an engine, OWL defines logical restrictions on how properties relate to classes. The mentioned examples of these constraints are:

- `allValuesFrom` : All values of a property must come from a specific class.
    - Example: If a `VegetarianRestaurant` class is defined with a restriction on the property `servesFood` using `allValuesFrom` `VegetarianFood` , we can deduce that any food served by an instance of that restaurant is vegetarian.
- `someValuesFrom` : At least one of the values for a property must come from a specific class.
    - Example: A `JapaneseRestaurant` class can be defined as a restaurant that serves at least some `Sushi` . If an instance is classified as a Japanese Restaurant, we know that it serves sushi as a menu item.

Vocabularies

3.1

- `FlashMemory` is a type of `Consumable` .
- `Printer` is a type of `Hardware` .
- `LCD` is a type of `Monitor` .
- An object with the property `Capacity` must be a `FlashMemory` .
- An object with the property `PagesPerMinute` must be a `Printer` .
- An object with the property `Model` must be a `Hardware` , a `Consumable` , and a `Monitor` .
    - RDFS interprets multiple domains as an intersection! If we wanted to write this as a disjunction we need OWL.

3.2

- "An animal eats living things"

```
<owl:Class rdf:about="#Animal">
    <rdfs:subClassOf> <!-- Being an Animal implies what's inside of this block -->
        <owl:Restriction> <!-- Places a restriction on a property -->
```

```
            <owl:onProperty rdf:resource="#eats"/> <!-- Selects the property eats -->
            <owl:someValuesFrom rdf:resource="#LivingThing"/> <!-- At least one of the things that it eats is a LivingThing -->
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

- "A parent of a living thing is a living thing"

```
<owl:Class rdf:about="#LivingThing">
    <rdfs:subClassOf>
        <owl:Restriction> <!-- If the property hasParent exists, it's value must come from the class LivingThing -->
            <owl:onProperty rdf:resource="#hasParent">
            <owl:allValuesFrom rdf:resource="#LivingThing">
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

- "A parent is an animal and parent of animals"

```
<owl:ObjectProperty rdf:about="hasParent"> <!-- If an object has the property hasParent -->
    <rdfs:range rdf:resource="#Animal" /> <!-- That object is an animal -->
    <rdfs:domain rdf:resource="#Animal" /> <!-- And the parent value is also an Animal -->
</owl:ObjectProperty>
```

- "A pet is an animal with a single human owner"

```
<owl:Class rdf:about="#Pet" />
    <rdfs:subClassOf rdf:resource="#Animal" /> <!-- Being a pet implies being an Animal -->
    <rdfs:subclassOf> <!-- It also implies what's inside of this block -->
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasOwner" />
            <owl:allValuesFrom rdf:resource="#Human" /> <!-- The owner that exists must be human -->
            <owl:cardinaliy rdf:datatype="&xsd;nonNegativeInteger">
                1 <!-- A pet must have exactly one owner -->
            </owl:cardinality>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

3.3

"allValuesFrom" makes it optional to have the restricted propery.
"someValuesFrom" makes it mandatory that we have at least one value of the restricted property.

- If an object is a UC_Engineer, it is also must also be an engineer an Engineer.
- If a UC_Engineer has at least one friend that is also an Engineer.
- If the UC_Engineer has the hasQuality property, then it's value must be GoodQuality
- A UC_Enginner studied at UC

Semantic Web & Linked Data

4.1

- The query is asking for things (?container) that taubz:me owns (ex:own).
    - For those things (containers), find what (?what) is inside them (ex:contains).

1. The things that match `taubz:me ex:own ?container` are:
    - `taubz:my_apartment`
    - `taubz:my_computer`
    - `taubz:my_desk`
2. Inside each of these containers, find things that match `?container ex:contains ?what`.
    - inside `taubz:my_apartment`:
        - `taubz:my_computer`, `taubz:friends_junk`
    - inside `taubz:my_computer`:
        - Nothing
    - inside `taubz:my_desk`:
        - `taubz:my_pens_and_pencils`
3. Final result: `taubz:my_computer`, `taubz:friends_junk`, `taubz:my_pens_and_pencils`

4.2

- Use URIs as names for things
    - Documents and digital contents available on the Web.
    - Also real objects and abstract concepts.
- Use HTTP URIs so that a client can look up those names.

    - When URIs identify real objects, they should be distinguished from documents that describe them.
- When someone looks up a URI, useful information should be provided.
    - Use standards such as RDF for this purpose.
- Include links to other URIs, so that a client can discover more things.
    - External RDF links are the "glue" that connects data islands into a global, interconnected data space.

      Using HTTP URIs provides a standard look up mechanism to retrieve a description of the object identified by the URI from the web. This ensures that when a URI is accessed via the standard HTTP protocol, machines receive machine-readable RDF data and humans receive a human-readable representation, such as HTML.

4.3

The RDF model enables the creation of a global, interconnected KG by providing a standardized framework for representing, exchanging, and linking data across different systems on the Web.

RDF organizes information into triples, this naturally creates a directed graph where the subject and object are nodes, while the predicate is the connecting edge.

These components (subject, predicates, and objects) are identified using URIs. By using HTTP URIs as names for things, RDF ensures that concepts and entitites are unambiguous and can be looked up over the Web.

URIs are global, so a dataset hosted on one server can refer to a resource defined in a dataset on a completely different server. These external RDF links connect isolated "data islands" into a single, global data space.

The mentioned triple structure is the abstract data model, to write and store this graph data (and process it with computers), we need serialization formats:

- **RDF/XML:** Original XML-based syntax for RDF, compatible with XML tools.
- **Turtle (Terse RDF Triple Language):** More compact and human-readable. Simplifies writing and reading triples.
- **JSON-LD (JSON for Linked Data):** Easily integrated into web applications using JSON, makes RDF accessible to web developers.

4.4

These principles aim to make the data machine-readable and interconnected.

- **Findable:** Data must be published in a way that makes it easy for both humans and computers to locate it.
- **Accessible:** Data must be retrievable using standard access protocols (HTTP) and identified by resolvable URIs, ensuring that machines can look up and retrieve information.
- **Interoperable:** Data should be structured so that it can be combined across different knowledge databases with minimal effort, allowing systems to integrate disparate data sources.
- **Reusable:** Data should adopt shared vocabularies used by different datasets and be released under open licences, ensuring it can be legally and technically used by various applications.

4.5

- Works authored by someone named "Bob Dylan"

```
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT ?work
WHERE {
    ?author rdfs:label "Bob Dylan"@en .
    ?work dbo:author ?author
}
```

- Works authored by winners of the Nobel Prize in Literature

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbr: <http://dbpedia.org/resource/>
SELECT ?work
WHERE {
    ?person dbo:award dbr:Nobel_Prize_in_literature .
    ?work dbo:author ?person
}
```

- English names of winners of the Nobel Prize in Literature and of works authored by them, with the language and genre of the work, if available

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?person, ?work, (str(?language) AS ?lang) (str(?genre) AS ?gen)
WHERE {
    ?pr dbo:award dbr:Nobel_Prize_in_Literature .
    ?pr rdfs:label ?person .
    ?wr dbo:author ?pr
    ?wr rdfs:label ?work .

    OPTIONAL {?wr dbp:language ?language} .
    OPTIONAL {?wr dbp:genre ?genre} .

    FILTER (lang(?person) = 'en').
    FILTER (lang(?work) = 'en')
}
```

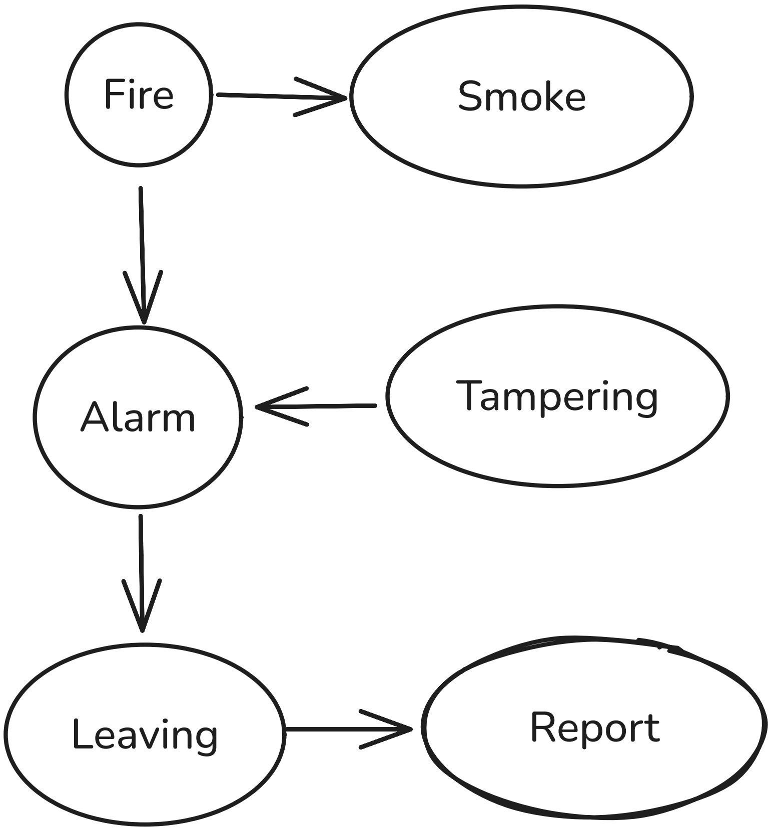
Uncertainty and Bayesian Networks

5.1

1. **Calculate Prior Probabilities:**
    1. $P(= \text{es}) = \frac{9}{14}$
    2. $P(= \text{no}) = \frac{}{14}$
2. **Calculate Conditional Probabilities:**
    1. $P(\text{Age } 30| = \text{es}) = \frac{2}{9}$ (Out of 9 yes rows, 2 of them have Age 30)
    2. $P(\text{ncome} = \text{medum}| = \text{es}) = \frac{4}{9}$
    3. $P(\text{Student} = \text{es}| = \text{es}) = \frac{6}{9}$
    4. $P(\text{Credt} = \text{ar}| = \text{es}) = \frac{6}{9}$
    5. Do the same for negative class...
3. **Calculate Posterior Probabilities:**
    1. $P(| = \text{es}) \times P(= \text{es}) \approx P(= \text{es}) \times P(\text{eature}| = \text{es}) = \frac{2}{9} \times \frac{4}{9} \times \frac{6}{9} \times \frac{6}{9} \times \frac{9}{14} \approx 0.028$
    2. $P(| = \text{no}) \approx 0.00$
4. **Normalize scores:**
    1. $\text{Tota} = 0.028 + 0.00 = 0.03$
    2. $P(= \text{es}|) = \frac{0.028}{0.03} = 80$
    3. $P(= \text{no}|) = \frac{0.0.00}{0.03} = 20$
    4. So the Naive Bayes classifier classifies this as the positive class.

5.2

- **Bayesian Network:**



- **Parameter Count**
  - Without a Bayesian network, the full joint probability distribution is calculated by using every combination of every variable. The parameters are the numbers we need to know to define the world (the probability of each combination). We have 6 binary variables so we need to know $2^6 = 64$ probabilities, however, we also know that the sum of the probabilities is 1, so the last parameter is redundant, hence, we need 63 parameters.
  - With a Bayesian network, this number is reduced because we do not assume that everything causes everything else. We break the giant JPD table into smaller, local tambles called Conditional Probability Tables (CPTs). We only need parameters for nodes given its direct parents.
    - For nodes with 0 parents, we just need 1 parameter, the probability of it being true ($P()$) because $P()$ is just given by $1 - P()$.
    - For nodes with 1 parent, we need a probability for every state of the parent. So, we need 2 parameters, one for $P(|\text{Parent}) = \text{ase}$ and another one for $P(|\text{Parent}) = \text{True}$.
    - For nodes with 2 parents, we need a probability for every combinations of state of the two parents (4 combinations).
    - In this case, we have 2 nodes with 0 parents, 3 nodes with 2 parents and 1 node with 2 parents. We need $1 + 1 + 2 + 2 + 2 + 4 = 12$ parameters.
- **Probability Calculation:**

  - $P(,,,\neg,) = P()\ P()\ P(|,)\ P(\neg|)\ P(|)$
  - This is obtained by multiplying the probability of each node given its parents (chain rule for Bayesian Networks).

5.3

The DAG structure enforces conditional independence, when two nodes are not connected in the graph, that means that they are independent.

Once we know the state of a node's direct parents, knowing the state of its grandparents, siblings, etc, adds no new informations about that node. This means we can ignore the majority of the network when calculating the probability of a single node.

We can simplify the chain rule from

$$P(_1, \ ,_n) = P(_1)\ P(_2|_1)\ P(_3|_1,_2)$$

where we need to assume everything depends on everything ($2^N - 1$ parameters) to

$$P(_1, \ ,_n) = {}_{i=1}^{n} P(_i|\text{Parents}(_i))$$

where we can drop the non-parents from the condition.

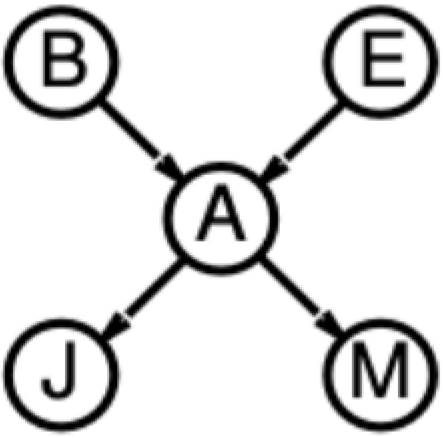
The global table size grows with the number of variables N as 2^N and the local tables sizes grow with $N\ 2^k$ where k is the maximum number of parents.

5.4

- Global semantics define the full joint probability distribution the the product of local conditional distributions:

  $$P(x_1, \ ,x_n) = {}_{i=1}^{n} P(x_i|\text{Parents}(_i)))$$

  for example:



$P(,,,\neg,\neg e) = P(|)\ P(|)\ P(|\neg,\neg e)\ P(\neg)\ P(\neg e)$

- Local semantics says that each node is conditionally independent of its nondescendants given its parents, which means that once we know a node's direct parents, no other node can provide additional information.

If we define a Bayesian network using these local conditional probabilities (local semantics), we can reconstruct the full joint probability distribution (global semantics) and vice-versa.

5.5

The Markov blanket is the minimal set of nodes required to make a node conditionally independent of all other nodes in the network.

The Markov blanked of that node consistes of:

- Parents of
- Children of
- Spouses of (other parents of 's children)

It's role in independence is the boundary of information relevance (once the values of the variables in the Markov blanket are known, probability of is unaffected by any other variable in the network).

$$P(|\mathrm{Markoanket}(),\mathrm{Aterodes}) = P(|\mathrm{Markoanket}())$$

5.6

Without a network structure, we assume every variable is correlated with every other variable. We need a probability for every possible combination of n variables.

- Total states: 2^n
- Parameters required: $2^n - 1$
  In a Bayesian network, we assume that a node depends only on its k parents.

So we break the full JPD table into n smaller independent tables (Conditional Probability Tables)

- Per node, we have 2^k parent combinations (we only need 1 probability per parent configuration, because $P(= \text{ase}|\text{Parents}()) = 1 - P(= \text{True}|\text{Parents}()))$
- In total, we have $n\ 2^k$ parameters because we have n nodes

5.7

The initial ordering of variables is the specific sequence $(_1, _2, , _n)$ in which you choose to analyze and add variables to the network.

When constructing the network, we apply the chain rule of probability based on this order. For every new node $_i$ we add, we must determine:

- Which of the previously added nodes are required to make $_i$ independent from the rest?
  These required nodes become the parents of $_i$.

The ordering can cause the graph to be sparse (efficient) or dense (inefficient)

- **Causal Ordering:**
  - We add "root causes" first, then intermediate variables, then "effects".
  - A node's parents become simply its direct causes. The parent set is small.
  - This makes the network sparse (few edges) and the number of parameters remains low ($(n\ 2^k)$).
- **Non-Causal Ordering:**
  - We add the "effects" first or mix the order randomly.
  - To preserve the correct JPD, the network is forced to add artificial dependencies. If we know a symptom (added first) but not the disease (added later), adding the disease node later requires drawing edges from the symptom (and potentially all other preceding nodes) to properly account for correlations.
  - The network becomes fully connected (dense). In the worst case, the last node $_n$ requires all previous $n-1$ nodes as parents. The number of parents scales with (2^n).

Example:

- We have 3 binary variables:
  - Burglary ()
  - Earthquake ()
  - Alarm ()

- In reality, and are independent causes and both cause .
- **Good, causal ordering:**
    - , ,
    - Add : No parents
    - Add : Is dependent on Burglary? No
        - Edges: None
    - Add : Does the Alarm depend on Burglary and Earthquake? Yes
        - Edges:
            - →
            - →
    - Result: →
- **Bad, mixed ordering:**
    - ,,
    - Add : No parents
    - Add : Is Burglary dependent on the Alarm? Yes (If the alarm is on, burglary probability goes up).
        - Edges:
            - →
    - Add : Is Earthquake dependent on Alarm and Burglary? We know it depends on Alarm (→).
        - Given that we know the Alarm is ringing (), does knowing there was a Burglary () tell us anything about the Earthquake ()? Yes, this is the "Explaining Away" effect. If the alarm is ringing and you see a burglar, the probability that an earthquake also happened drops. If you don't see a burglar, the probability of an earthquake goes up.
        - Because provides information about (conditional on), we must add an edge from to .
        - Edges:
            - →
            - →
    - Result: → , → , → .
    - We have created a fake dependency between Burglary and Earthquake that requires extra parameters.

5.8

In a diagnostic system (e.g., medical diagnosis), a single symptom (effect)Y often has many potential causes (parents). If a symptom has k diseases as parents, a standard Conditional Probability Table requires 2^k parameters. We can reduce that parameter count to linear ((k)) with functional relationships and hidden variables:

Functional Relationships such as Noisy-OR:

- Instead of listing every combination of parents, you assume that each cause acts independently to produce the effect. The effect happens if at least one active cause succeeds in triggering it.
  - **Noisy-OR:** Requires only k probabilities instead of 2^k (one "link probability" p_i for each parent $_i$, representing the chance that $_i$ alone triggers the effect).
  - **Example:**
    - If a patient has fever (). The potential causes (parents) are flu ($_1$), cold ($_2$), and malaria ($_3$).
    - **Using the Standard Method (Full CPT):** We must define the probability of fever for every combination of the 3 diseases. We need $2^3 = 8$ probabilities.
    - **Noisy-OR Method:** Each disease causes fever independently. If we have both flu and cold, the fever is triggered if either the flu mechanism succeeds OR the cold mechanism succeeds.
      - **Parameters Needed:** We only need 3 numbers (the "link probabilities"):
        1. $_1 = P(|\text{uon}) = 0.8$
        2. $_2 = P(|\text{Codon}) = 0.4$
        3. $_3 = P(|\text{Maaraon}) = 0.9$
      - **Calculation:** If a patient has flu and cold (but not malaria), the probability of fever is:

$$P(|_1, _2, \neg_3) = 1 - (1 - _1)\ (1 - _2)P() = 1 - 0.2\ 0.6 = 1 - 0.12 = 0.88$$

        so we can model the whole interactions using just 3 parameters instead of 8.

Hidden Variables:

- This modifies the topology. If a node has many parents, you introduce intermediate hidden nodes to break the parents into smaller groups.
  - Instead of 10 parents pointing to , we group parents into 1-5 hidden node $_1$ and parents 6-10 into hidden node $_2$. Then $_1$ and $_2$ point to .
  - **Example:**
    - We want to model a "car won't start" () node. It has 4 parents: Battery Dead (), Starter Broken (), No Gas (), and Fuel Pump Broken (p).

- **Standard Model (Full CPT):** Node has 4 parents. We need $2^4 = 16$ rows in the table.
- **Hidden Variable Method:** We introduce intermediate "Summary" nodes to group related causes.
  - **Group 1 (Electrical):** Create hidden node $_1$ ("Electrical Failure"). Parents are and .
  - **Group 2 (Fuel):** Create hidden node $_2$ ("Fuel System Failure"). Parents are and p.
  - **Final Step:** Node now only has 2 parents ($_1$ and $_2$).
  - **New Parameter Count:**
    - Table for $_1$ (Parents ,): $2^2 = 4$ rows.
    - Table for $_2$ (Parents , p): $2^2 = 4$ rows.
    - Table for (Parents $_1$, $_2$): $2^2 = 4$ rows.
    - Total: 12 parameters (instead of 16).

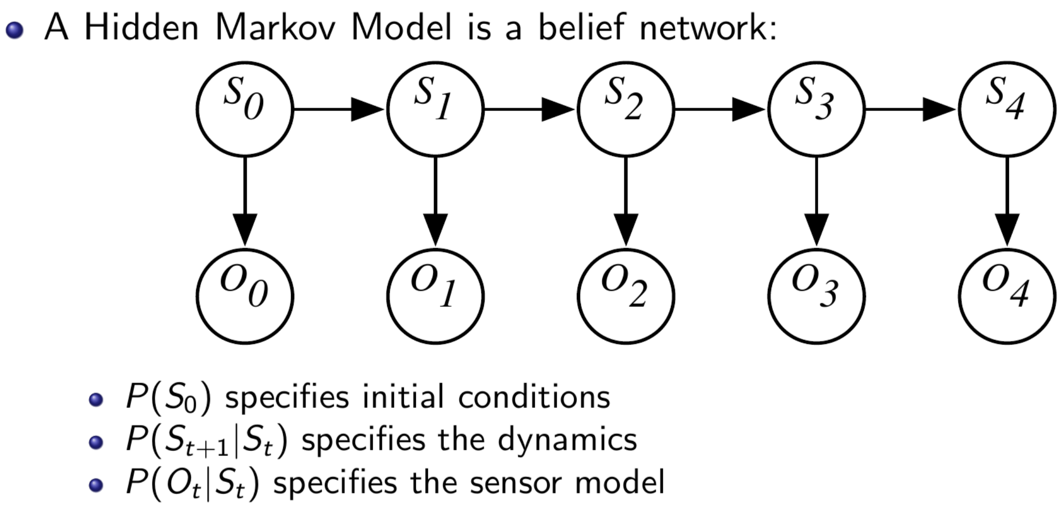
Probabilistic Reasoning Over Time

6.1

Hidden Markov Model (HMM) problem. An HMM is a statistical model used to predict the behavior of systems where we can only observe the symptoms (observations), not the cause (hidden states).
Components of an HMM:

- **Hidden States ():** The reality we want to know but can't see.
  - In this case: "Is the machine Working or Faulty?"
- **Observations ():** The evidence we can see.
  - In this case: "Is the light Red or Green?"
- **Transition Model ($P(_{+1}|)$):** The physics of how the world changes.
  - In this case, for example: If the machine is working today, there is an 80% chance it will still be working tomorrow ($P(_{+1}|) = 0.8$).
- **Sensor/Emission Model ($P(|)$):** The reliability of our evidence.
  - In this case, for example: If the machine is faulty, there is an 80% chance the light will be red and 20% green ($P(|) = 0.8$ and $P(|) = 0.2$). It operates under the Markov Assumption, which simplifies the math. It assumes that the future depends only on the present, not the past. In this case, to predict if the machine will be broken tomorrow, we only need to know its state today. We don't need to know it's entire

history of breakdowns.



- A Hidden Markov Model is a belief network:

  - $P(S_0)$ specifies initial conditions
  - $P(S_{t+1}|S_t)$ specifies the dynamics
  - $P(O_t|S_t)$ specifies the sensor model

- Task 1 is a filtering (state estimation) task where we need to estimate the CURRENT belief state () based on all evidence observed up to this point (e_1). We combine the previous knowledge (prior) with the new observation (evidence) using Bayes rule to get an updated posterior belief.
- Task 2 is a prediction task, where we need to estimate a future state ($_{+k}$) based on the current belief state. We take the posterior belief we calculated from task 1 and projects it forward one step using the transition probabilities (how the system changes over time).

TASK 1: Compute the Posterior Probability Today

- Update belief about machine's state (or) given observation of .
- We can use the formula $P(|e) = P(e|)P()$ (Bayes' Rule with normalization constant).
  - **For :** $P(|)P() = 0.1 \times 0. = 0.0$
  - **For :** $P(|)P() = 0.8 \times 0. = 0.40$
  - Normalize ():
    - $\mathrm{Totannormaed} = 0.40 + 0.0 + 0.4.$
    - $P(|) = \frac{0.0}{0.4} \approx 0.111$
    - $P(|) = \frac{0.40}{0.4} \approx 0.889$
  - So there is a 11.1% chance the machine is working and 88.9% chance it's faulty.

TASK 2: Predict the Probability Tomorrow
We predict the state at $+1$ using the posterior probabilities from today and the transition model.

- **Formula:** $P(_{+1}) = \sum P(_{+1}|)P(|e)$
- For $_{+1}$: $P(_{+1}) = P(_{+1}|)P(|) + P(_{+1}|)P(|) \approx 0.3$
- For $_{+1}$ The probabilities sum to 1, so $P(_{+1}) = 1 - 0.3 = 0.64$

6.2

Same methods as the previous question

TASK 1: Posterior Probability Today

- $P(|) = P(|)P() = 0.4$
- $P(|) = P(|)P() = 0.1$
- Normalization:
  - $P(|) = 0.$
  - $P(|) = 0.2$

TASK 2: Probability for Tomorrow
$P(_{+1}|) = P(_{+1}|)P(|) + P(_{+1}|)P(|) = 0.62$
$P(_{+1}|) = 1 - 0.62 = 0.3$

6.3

The difference between first-order and second-order Markov Processes is how much history is needed to predict the current state.

- **First-Order:**
  - Assumes that the current state () depends only on the immediate previous state ($_{-1}$). The past history is irrelevant once $_{-1}$ is known.
  - $P(|_{0-1}) = P(|_{-1})$.
  - Simplest model but may not be exactly true in the real world.
- **Second-Order:**
  - Assumes that the current state () depends on the two previous states ($_{-1}$ and $_{-2}$).
  - $P(|_{0-1}) = P(|_{-2}, _{-1})$.
  - This model is more accurate for systems where velocity and acceleration matter (trends) but it increases the complexity of the Bayesian Network because each state has more parents.

Sensor Markov Assumption:

- This states that the evidence observed at time () depends only on the current state ().
- It assumes evidence is independent of all past states and all past evidence.
- $P(|_{0}, _{0-1}) = P(|)$
- Without this assumption, interpreting a sensor would require reading the entire history of the system. This assumption allows us to make calculations such as filtering recursive and computationally feasible.

6.4

Basically they differ in which time slice they target relative to the available evidence (e_1).

Filtering:

- **Target:** Current state ()
- **Evidence Used:** All observations up to the current moment (e_1)
- **Goal:** Estimate what's happening right now. Recursive algorithm typically used for real-time tracking, such as GPS estimating a car's location.
- **Formula:** $P(|e_1)$.

Prediction:

- **Target:** A future state ($_{+k}$ where k 0).
- **Evidence Used:** All observations available up to the current moment (e_1), without any future evidence.
- **Goal:** Project the current state forward to guess what will happen. This is useful for evaluating the potential consequences of a sequence of actions.
- **Formula:** $P(_{+k}|e_1)$.

Smoothing:

- **Target:** A past state ($_k$ where 0 $k <$).
- **Evidence Used:** All observations up to the current moment (e_1), effectively using hindsight (future evidence relative to k).
- **Goal:** Refine the estimate of what happened previously by including later observations, the estimate of past state becomes more accurate than it was at the time. Essential for learning.
- **Formuka:** $P(_k|e_1)$.

Most Likely Explanation:

- **Target:** The entire sequence of states ($_1$).
- **Evidence Used:** The entire sequence of observations (e_1).
- **Goal:** To find the single specific path or sequence of states that best explains the evidence. This is distinct from smoothing because the most likely sequence is not necessarily the same as the sequence of the most likely individual states.
- **Formula:** $\arg\mathrm{ma}_1 P(_1|e_1)$.

6.5

Filtering calculates the belief distribution for the current state. It considers all possible paths that could have led to the current state and sums their probabilities. By summing, it ignores the exact past states and blends all history into the current belief. This results in a probability distribution, e.g., "70% rain, 30% sun".

- Uses summation to combine probabilities from all possible previous states $_{-1}$.
-

Most Likely Explanation uses the Viterbi algorithm to find the single most probable sequence of states ($_{1}$) that explains the evidence. Its goal is to identify a specific trajectory or path the system took over time. Instead of summing probabilities from previous steps, it looks for the maximum probability path reaching the current state. It tracks the "best path to here" instead of "total probability of being here". This results in a sequence, e.g., "rain-rain-sun-rain".

- Uses maximization (ma) to pick the single best previous state $_{-1}$ that leads to ,

---

Text Classification and Knowledge Extraction

7.1

1. Extract all of the unique words in the training set:
   - `{experiência, muito, fraca, grande, desilusão, tempo, à, espera, boa, relação, qualidade, preço, excelente, serviço}`
   - Vocabulary size: $|| = 14$
   - Total documents: $N =$
2. Calculate prior probabilities $P()$:
   - This is the probability of a class occurring based solely on the number of documents.
   - $P(-) = \frac{3}{}$
   - $P(+) = \frac{2}{}$
3. Calculate likelihoods $P(|)$:
   - We calculate the probability of each word appearing in a class. We use Laplace Smoothing (adding 1 to the numerator and $||$ to the denominator so that we don't have 0 probabilities anywhere).
   - Formula:

$$P(_i|) = \frac{\text{count}(_i,) + 1}{\text{count}(\text{aordsn}) + ||}$$

- Counts:
  - Total words in (-): $N_{-} = 9$ (`{experiência, muito, fraca, grande, desilusão, muito, tempo, à, espera}`). NOTE: In the class material, our professor ignores `à` because it is a stop word.
  - Total words in (+): $N_{+} =$ ({ `muito, boa, relação, qualidade, preço, excelente, serviço` })
  - $|| = 14$
- We want to classify "tanta, espera, mas fraca qualidade" but "tanta" and "mas" are out of vocabulary, so we just ignore it.

4. Calculate Final Scores:
   - We multiply the **Prior** by the **Likelihoods** of the known words.
     - **Class Negative (-):**
       - $\text{Score}(-) = P(-) \times P(\text{esera}|-) \times P(\text{raca}|-) \times P(\text{uadade}|-)$
       - $\text{Score}(-) = \frac{3}{} \times \frac{2}{23} \times \frac{2}{23} \times \frac{1}{23}$
     - **Class Positive (+):**
       - $\text{Score}(+) = P(+) \times P(\text{esera}|+) \times P(\text{raca}|+) \times P(\text{uadade}|+)$
       - $\text{Score}(-) = \frac{2}{} \times \frac{1}{21} \times \frac{1}{21} \times \frac{2}{21}$
   - Then choose the one with the highest score.

7.2

- **Naive Bayes:**
  - $\text{Accurac} = \frac{110+30}{200} = 0.0$
  - $\text{Precson} = \frac{110}{110+40} = 0.3$
  - $\text{eca} = \frac{110}{110+20} \approx 0.8$
  - $1 = 2 \times \frac{0.33 \times 0.846}{0.33+0.846} \approx 0.9$
- **Rule-Based:**
  - $\text{Accurac} = \frac{80+0}{200} = 0.6$
  - $\text{Precson} = \frac{80}{80+20} = 0.80$
  - $\text{eca} = \frac{80}{80+0} \approx 0.62$
  - $1 = 2 \times \frac{0.80 \times 0.61}{0.80+0.61} \approx 0.0$
    Rule-Based has higher precision so it's "safer" when it predicts positive. Naive Bayes is better at finding the positive comments (recall) and has higher overall accuracy. The F1 score also favors NB.

7.3

"the water"

- Calculate the probability of starting with each tag using $P(\text{Tag}|) \times P(\text{te}|\text{Tag})$.
  - D: $0.4 \times 1.0 = 0.4$

    - N: $0.4 \times 0 = 0$
    - V: $0.2 \times 0 = 0$
    - Current best path: We MUST start with D.
- Calculate the score of transitioning from the previous valid () to the next possible tags.
    - Path D → D: $0.4 \times P(|) \times P(\text{ater}|) = 0.4 \times 0 \times 0 = 0$
    - Path D → N: $0.4 \times P(N|) \times P(\text{ater}|N) = 0.4 \times 1.0 \times 0. = 0.28$
    - Path D → V: $0.4 \times P(|) \times P(\text{ater}|) = 0.4 \times 0 \times 0.3 = 0$
- The only valid non-zero path is: D, N

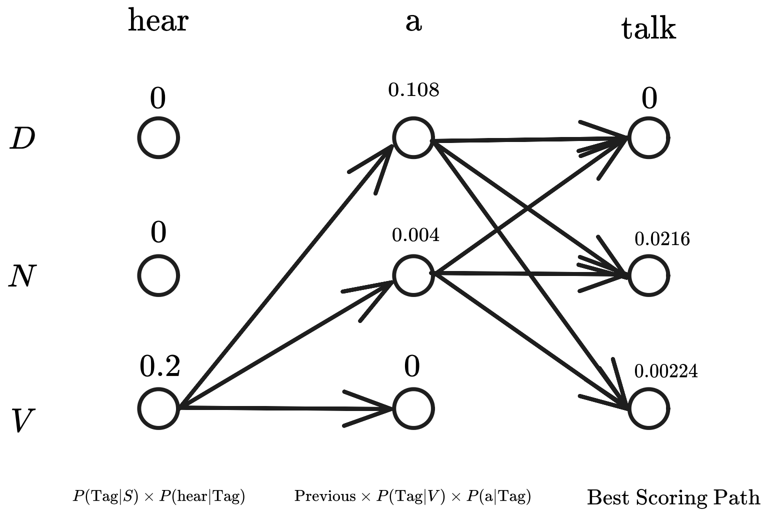
"a talk"

- Starting with each tag $(P(\text{Tag}|) \times P(\text{a}|\text{Tag}))$
    - D: $0.4 \times 0.9 = 0.36$
    - N: $0.4 \times 0.1 = 0.04$
    - V: $0.2 \times 0 = 0$
- Get the best path to reach each possible tag at $= 2$:
    - To reach N:
        - From D: $0.36 \times P(N|) \times P(\text{tak}|N) = 0.36 \times 1.0 \times 0.2 = 0.02$
        - From N: $0.04 \times P(N|N) \times P(\text{tak}|N) = 0.04 \times 0.1 \times 0.2 = 0.0008$
        - Best path to N: D, N
    - To each V:
        - From D: $0.36 \times P(|) \times P(\text{tak}|) = 0.36 \times 0 \times 0.8 = 0$
        - From N: $0.04 \times P(|N) \times P(\text{tak}|) = 0.04 \times 0. \times 0.8 = 0.0224$
        - Best path to V: N, V
- Ending at N has a higher score. Backtracking from N, the previous state was D, so the best path is D, N

"hear a talk"

- Starting with each tag $(P(\text{Tag}|) \times P(\text{ear}|))$
    - D: $0.4 \times 0 = 0$
    - N: $0.4 \times 0 = 0$
    - V: $0.2 \times 1.0 = 0.2$
    - Current best: must start with V
- Score of transitioning from V to the next possible tags.
    - From $\to$: $0.2 \times P(|) \times P(\text{a}|) = 0.2 \times 0.6 \times 0.9 = 0.108$
    - From $\to N$: 0.004
    - From $\to$: 0
- Score of transitioning from the non-zero paths to each possible tag.
    - To reach N:

    - From D (0.108): $0.108 \times P(N|) \times P(\text{tak}|N) = 0.108 \times 1.0 \times 0.2 = 0.0216$
    - From N (0.004): $0.004 \times P(N|N) \times P(\text{tak}|N) = 0.00008$
    - So the best path to N is $\to \to N$
  - To reach V:
    - From D: D can't transition to V, so it's 0
    - From N (0.004): $0.004 \times P(|N) \times P(\text{tak}|) = 0.004 \times 0. \times 0.8 = 0.00224$
    - Best path to V: $\to N \to$
  - The highest score is the path ending in N (VDN)


hear
a
talk
D
N
V
0
0
0.2
0.108
0.004
0
0
0.0216
0.00224
$P(\text{Tag}|S) \times P(\text{hear}|\text{Tag})$
Previous $\times P(\text{Tag}|V) \times P(\text{a}|\text{Tag})$
Best Scoring Path

7.4

Word Sense Disambiguation (WSD) is the task of identifying which distinct meaning (sense) of a word is activated by its use in a particular context. Many words are polysemous (have multiple meanings).

The sense embeddings provided are embeddings of each specific meaning of bank. The context vector is an embedding of the word bank shaped by its surroundings. We want to check which sense embedding is the closest to .

To solve this, we just calculate the dot product or cosine similarity between and every sense embedding and choose the best scoring sense.

7.5

Sequence labeling (aka token classification) is the task of classifying every token in a sequence. Part-of-Speech (PoS) tagging is the process of assigning a gramatical category (such as noun, verb, adjective) to words within a text, it helps identifying the content words that carry the meaning of the text. Named Entity Recognition (NER) is the task of identifying and delimiting mentions of real-world entities (e.g., people, organizations, locations) in text and classifying them into predefined categories.

Sequence labeling provides the computational framework to solve both NER and PoS tagging. Its contributions to these tasks are:

- Unified problem formulation (both tasks are a classification problem where the goal is to assign a label to every token).
- Contextual prediction (by framing the task as a sequence, we can use models such as HMM or BERT to predict the most likely sequence of tags based on the context of neighboring words, rather than treating words in isolation),
- Handling Multi-Word Entities for NER (sequence labeling uses schemes like BIO (begin, inside, outside) to precisely delimit where an entity mention starts and ends, which is crucial for extracting entities that span multiple tokens, e.g., "University of Coimbra").

7.6

Distant supervision is a technique used to automatically "label" large amounts of text for training.
It retrieves a massive amount of known entity pairs (seeds) from the large external database such as DBPedia, scans the text volume to find sentences that mention both entities from a retrieved pair, assumes that any sentence containing both entities expresses the known relation from the database and automatically tags the sentence with that relation.
This process creates a large, labeled corpus that is then used to train a supervised classifier, bypassing the need for manual human annotation.

7.7

Open Information Extraction (Open IE) is a paradigm that extracts structured relational triples (subject, relation, object) from text without being limited to a fixed, pre-defined set of categories (like "born_in" or "employee_of"). Instead of looking for specific relations, it attempts to extract any relation expressed in the text, enabling the discovery of massive amounts of information without relation-specific training data.

It allows for the retrieval of relational triples without predefined types by:

- **Relying on Linguistic Structures:** Instead of looking for specific words associated with a fixed schema, it uses general linguistic patterns such as PoS tags, text chunks and syntactic dependencies to identify phrases that function as relations.
- **Extracting Surface Forms:** It identifies the relation predicate directly from the text itself (e.g., extracting the phrase "was founded in" as the relation) rather than classifying text into a closed set of ontology labels.

- **Using General Patterns:** It employs patterns (often based on verbs) to locate potential relations dynamically. For example, in the sentence "Chicago was founded in 1833" it detects the verb phrase structure to extract the triple `(Chicago, was Founded In, 1833)`.

Sentiment Analysis

8.1

🤣👉 @TiagoSilva 👈🤣

8.2

TF-IDF (Term Frequency-Inverse Document Frequency) is a numerical statistic used to reflect how important a word is to a document in a corpus. It solves the issue where frequent words (e.g., "the", "and") appear often but don't carry a lot of information while rare words are more significant for distinguishing documents.

- Term Frequency ($\mathrm{t}_{i,}$): measures how frequently a term i occurs in a document .
  - $\mathrm{t}_{i,} = \frac{_{i,}}{\sum_{||=0} ,}$ (frequency of term / total terms in doc).
- Inverse Document Frequency (d_i): measures how rare a term is across the entire corpus.
  - $\mathrm{d}_i = \mathrm{og}\,\frac{N}{n_i}$ (log of total docs / docs containing term).
- The final TF-IDF weight is the product $\mathrm{T}(i,) = \mathrm{t}_{i,} \times \mathrm{d}_i$.

The Naive Bayes model makes two naive independence assumptions:

1. **Bag-of-Words Assumption:** Assumes the position of words in the document does not matter.
2. **Conditional Independence:** Assumes that the probability of a feature (word) occurring ($P(x_i|)$) is independent of all other features given the class .

TF-IDF weighting improves text classification in Naive Bayes by modifying how the word probabilities ($P(_i|)$) are calculated during the learning phase. Instead of simply counting word occurrences (raw frequency), the model uses the TF-IDF weights to account for word relevance:

- Words are weighted by their importance to a specific document relative to the corpus, so the probability of a word given a class is estimated by summing the weights of that word across all documents in the class, rather than just raw counts:

$$P({}_i|) = \frac{\sum \mathrm{egt}({}_i,)}{\sum \sum \mathrm{egt}(,)}$$

This weights down frequent, less informative words like stopwords and weights up distinctive words.

8.3

Sentiment Lexicons are lexical resources (lists or databases) that associate words with specific sentiment values or scores. Instead of learning from data, these lexicons act as a pre-defined knowledge base where:

- Words are assigned polarity (positive/negative scores) or classified into emotions.
- Some lexicons, like NRC VAD, assign values for Valence (positive/negative), Arousal (calm/excited) and Dominance.
- e.g., AFINN, SentiWordNet, SenticNet
- This allows for a rule-based approach to sentiment analysis (simply looking up the words in a text and aggregating their scores to determine overall sentiment).

These approaches fail (sometimes) because they treat words in isolation with simple rules, making them vulnerable to complex linguistic structures.

- **Negation:** Lexicons assign a fixed score to a word (e.g., "good" is positive). However, negation flippers can reverse this meaning (e.g., "never good" or "do not dislike"). A simple lexicon lookup might just assign a positive score after reading "good".
- **Anaphora:** Resolving references like pronouns to their real entities. For example, in "my girlfriend was happy with her phone and its sound quality", the word "its" refers to the phone. Lexicons lack the context to understand what "its" refers to, it's difficult to attribute the sentiment (sound quality) to the correct target (phone).
- **Figurative Language (Irony/Sarcasm):** We often use words with a positive literal meaning to express negative sentiment. For example, saying "bonito serviço" when something goes wrong. A lexicon would just detect "bonito" as positive words.

NLP, Distributional Semantics, Word Embeddings

9.1

Formal programming languages are designed to be deterministic and unambiguous. NLP is unstructured and ambiguous, it requires probabilistic interpretation rather than strict logic.

Ambiguity:

- **Phonological:** Identical sounds can form different valid sequences (e.g., "mal a" vs. "mala").
- **Lexical:** Words can be different parts of speech (noun/verb), have multiple meanings (polysemy), or refer to different entity types (person/location).
- **Syntactic:** Boundaries are unclear (e.g., tokenizing compounds), and references (anaphora) are difficult to resolve.
- **Semantic:** Literal words often differ from intended meaning due to negation, irony or sarcasm.

Vagueness & Subjectivity: Language involves continuous concepts (e.g., degree of "goodness") and subjective feelings, unlike the binary truth values of code.

Solutions: NLP addresses these by using probabilistic models to calculate likelihoods and distributional semantics (vectors to capture context-based meaning).

9.2

Traditional Pipeline:

- **Analysis:**
  - Decomposes language processing into a strict sequence of independent steps, performing morphological (e.g., "tokenization", "lemmatization"), syntactic (e.g., PoS Tagging, Parsing), and semantic analysis (e.g., NER, Relation Extraction) explicitly. Errors in early steps propagate to later ones.
- **Training:**
  - Relies on Supervised Learning, requires large annotated datasets (corpora) to train specific models (e.g., HMMs, CRFs or fine-tuned BERT) for each specific task in the pipeline.
- **Flexibility and Input:**
  - Rigid. Changing the task requires retraining the model or rearranging the pipeline components.

Zero-shot LLM:

- **Analysis:**

    - Treats the task as a single end-to-end text generation problem. Does not perform explicit intermediate analysis steps.
    - Morphological, syntactic and semantic knowledge is implicitly encoded in the model's pre-trained weights and activated via NLP prompt.
- **Training:**
    - Relies on Pre-training and Prompting. The model predicts the answer given a task description (prompt) without gradient updates or fine-tuning.
- **Flexibility and Input:**
    - Very flexible, the task is defined by the prompt (e.g., "Identify all relations in the text", "Translate to French"). The same model can switch between tasks (sentiment analysus, QA, Information Extraction).

9.3

Distinction Between Conversational Agents:

- **Virtual Assistants (Task-Oriented):**
    - Designed to be utilitarian tools focuses on completing specific tasks efficiently.
    - They want to minimize the number of turns in a conversation to achieve a user's intent quickly (e.g., setting a timer, finding a restaurant, ...).
    - Priorize brevity and functionality.
- **Companions (Chat-Oriented):**
    - Focus on engagement and maintaining a conversation.
    - They want to maximize the number of turns and the duration of the interaction.
    - Simulate social interaction, aim to be entertaining, empathetic, ...

ELIZA is a chatbot that created an illusion of understanding, but it had limitations:

- **Not :** ELIZA relied on simple pattern matching (e.g., "I feel X" → "Why do you fell X?") and substitution rules. It did not "understand" the meaning of words.
- **Lack of Context:** No memory of previous turns in the conversation or knowledge of the world.
- **Superficiality:** The responses are formulaic rephrasings or the user's input. If it doesn't recognize a keyword, it generates a generic fallback phrase (e.g., "Please go on").

9.4

The distributional hypothesis states that "a word is characterized by the company it keeps" and that frequent words in similar contexts convey similar meanings.

The co-occurence matrix implements this by quantifying "company" as the frequency with which a target word appears in a specific context (e.g., 5-word window) of other words.

In the matrix, rows represent words and columns represent their contexts. Words that frequently share the same "company" (e.g., car and van both appear near drive) end up with similar vector representations.

9.5

Advantages of Dense Embeddings:

- **Better Memory Efficiency:** Co-occurence matrices scale with the vocabulary size $(|| \times ||)$ and contain many zeros (very sparse), dense embeddings are fixed, lower-dimensional vectors.
- **Better for Machine Learning:** Dense vectors are easier to use as features for machine learning models.
- **Generalization:** Less overfitting than sparse representations.

Maintenance of Linguistic Regularities:

- **Learning Mechanism:** Instead of simple counting, models like Word2Vec train a classifier to predict a word given its context (or vice-versa) using a large corpus. The learned weights from this supervision become the word embeddings. This captures deep semantic relationships.
- **Vector Arithmetic:** The vectors preserve linguistic regularities that can be used with algebra. $\text{Kng} - \text{Man} + \text{oman} \approx \text{ueen}$.

9.6

Models like Word2Vec and GloVe assign a single, fixed vector to the word "bank", transformer-based models generate dynamic embeddings where the vector for a token is a function of the entire input sequence. The representation of "bank" changes based on context.

Large Language Models & Knowledge

10.1

LLMs can generate coherent text because they are trained on massive datasets to predict the probability of the next token in a sequence. They memorize patterns and superficial linguistic forms effectively.
However, they do not "understand" the meaning of what they say, just like parrots that just mimic language.

The Paradox: There are situations where the model speaks coherently but produces hallucinations. The generation process is statistical, not semantic.

10.2

These techniques ground the model's generation in external verifiable data rather than just the pre-trained internal weights.

- **Retrieval Augmented Generation (RAG):**
  - Retrieves relevant textual documents (or "chunks") from an external index during the inference phase and combines them with the user's prompt.
  - This allows the model to answer based on specific information, it also makes the answer verifiable (we can just check the sources) and enables updated responses without retraining.
- **Knowledge Injection:**
  - Incorporates structured knowledge (e.g., from knowledge graphs) into the model.
    - It can do it dynamically (retrieve facts during inference) or statically (embed knowledge into parameters via fine-tuning or adapters).
    - By explicitly providing factual triples (`(subject, relation, object)`) in the prompt or training, it guides the model to the correct entity or fact, correcting potential errors.

10.3

Plausible Explanation: Coherent, convincing, and aligns with human reasoning.
Faithful Explanation: Accurately represents internal computational process the model used to reach a decision.

LLMs generate explanations as new text sequences. These are often hallucinations of reasoning, they might be stories invented to justify an output instead of a trace of the actual decision process. For example, a model might claim it "carried the 1" to solve a math problem (plausible), while internally it simply recognized a pattern of numbers (unfaithful).

A plausible explanation can make users trust the model's logic while it's incorrect.

LLMs are uninterpretable, we rely on explanations. If we cannot distinguish between a plausible story and what actually happened, we cannot safely use these models in high stakes domains like healthcare