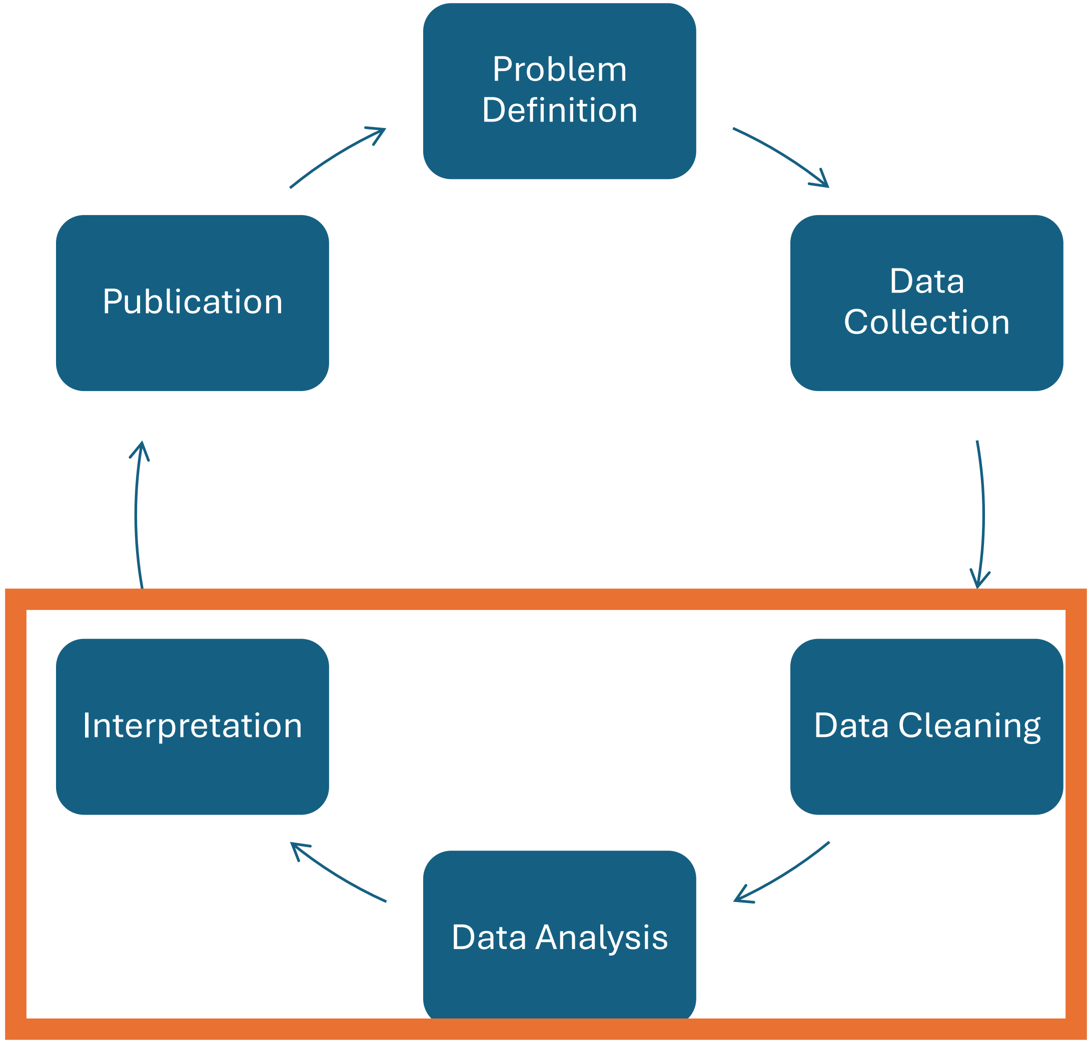
JAN BISCHOFF
DATA SCIENCE & VISUALIZATION

Mastering Natural Language Processing with Python

Organisation of the course

- Alternation between theory sessions and hands-on programming sessions in Google Colab
- Focus on practical application rather than theory
- Broad overview about NLP concepts
- Q&A after each programming session
- Lunch break at 12:00
- Coffee breaks in the morning and afternoon

The Research Lifecycle



Problem Definition

Data Collection

Data Cleaning

Data Analysis

Interpretation

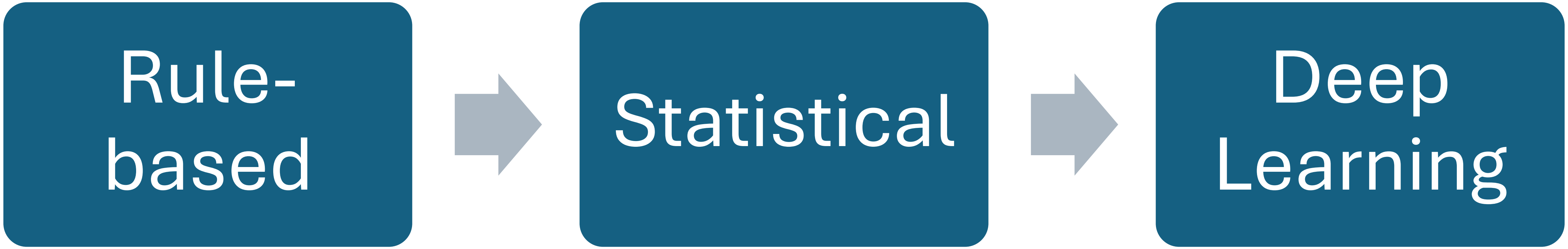
Publication

Focus of this course

NLP (with Python) offers new possibilities for researchers

- Semantic search and understanding of unstructured data
- Automated text analysis at scale
- Contextual capabilities
- Discovery of hidden patterns
- Python is the most used language for data science
- Python offers many libraries for working with text

The Evolution of NLP



Rule-based

Statistical

Deep Learning

Course content

1. Traditional methods for working with text in Python
2. Pipeline approach for processing text
3. Vectorization
4. Text classification and topic modelling
5. Deep learning methods
6. NLP best practices for researchers

Practice time

Let’s get comfortable with coding in Google Colab notebooks.

1. Traditional methods for working with text in Python

String methods (1)

- String is the Python data type for storing text data
- Python offers a wide range of built-in functions for working with texts
- The available methods can be accessed via the dir() function
- The documentation of the methods can be found with the help() function applied on the datatype, e.g. (help(str))

String methods (2)

- Commonly used methods are:
  - lower()
  - upper()
  - capitalize()
  - strip()
  - replace()
  - find()
  - split()
- To apply str methods to pandas datatypes you must include the str accessor

Practice time

Let’s get hands-on and practice String methods!

Open your Google Colab exercise notebook.

Solve exercises 1 – 3.

Regular Expressions - Overview

What they are:

- A specialized "search language" used to identify, extract, and manipulate specific patterns within text
- A sequence of characters forms a search pattern

Can be used for:

- Extracting specific information out of a string
- Validating a specific format, e.g. mail
- Cleaning your data, e.g. remove html tags
- Finding and replacing complex patterns

Regular Expressions - Examples

- Email and Password Validation: When you sign up for a website, regex is used in the background to verify that an email contains an @ and a domain (e.g., ^[\w\.-]+@[\w\.-]+\.\w+$)
- URL Parsing and Routing: Web frameworks use regex to look at the address in your browser. It might identify that in website.com/user/123/, the 123 is a specific user ID that needs to be pulled from a database

Regular Expressions – Character Classes

Character classes are the building blocks of regex. They allow you to match types of characters rather than specific literals.

- \d: Matches any digit (0-9)
- \D: Matches any non-digit
- \w: Matches any word character (alphanumeric + underscore)
- \s: Matches any whitespace (space, tab, newline)
- [a-z] / [A-Z]: Custom ranges for lowercase or uppercase letters
- [^...]: A negated set (matches anything not inside the brackets)

Regular Expressions – Anchors & Wildcards

Anchors do not match characters; they match positions within the text. Wildcards allow for flexibility.

- **. (Dot)**: The **Wildcard**. Matches any single character except a newline
- **^ (Caret)**: Matches the **start** of a string
- **$ (Dollar)**: Matches the **end** of a string
- **\b**: Matches a **word boundary** (the edge between a word character and a space/punctuation)

Regular Expressions – Greedy Operators

- Quantifiers tell regex how many times to match a pattern
- By default, they are **greedy** (they take as much as possible)
  - *** (Greedy)**: Matches 0 or more repetitions
  - **+ (Greedy)**: Matches 1 or more repetitions
  - **? (Lazy/Non-greedy marker)**: When added after a quantifier (*? or +?), it forces the engine to match the **smallest** possible segment

Regular Expressions – Lookahead / Lookbehind

They check if a pattern exists before or after your target without including that pattern in the final result

- **X(?=Y) Positive Lookahead**: Match X only if followed by Y
- **(?<=Y)X Positive Lookbehind**: Match X only if preceded by Y
- **?! / ?<!**: Negative versions (match if *not* followed/preceded by)

Regular Expressions – Special Flags and Functions

Python's re module provides "Flags" to change how the entire engine behaves

- **re.DOTALL**: Makes the . (dot) match newlines as well
- **re.IGNORECASE**: Makes the search case-insensitive
- **re.MULTILINE:** Makes ^ and $ match the start/end of every line, not just the whole string
- re.VERBOSE: Linebreaks inside a string are ignored and can be commented
- **re.compile()**: Pre-compiles a pattern for better performance if used inside a loop

Practice time

Let’s get hands-on and practice regular expressions!

Open your Google Colab exercise notebook.

Solve exercises 4 – 6.

2. Pipeline approach for processing text

The challenges of raw text analysis

- Dimensionality Explosion: "Apple," "apple," and "APPLE" are treated as three unique entities
- Stopword Saturation: Frequent words like "the," "is," and "and" usually carry the least information
- Semantic Fragmentation: a computer doesn't know that "running," "runs," and "ran" all refer to the same action
- Skewed Feature Importance: Machine Learning models may assign weight to irrelevant noise

Text preprocessing as solution

- Tokenization: segmenting the text into a list of tokens
- Lowercasing: converting all words to lowercase
- Stop word removal: Stop words are commonly occuring words in a language
- Stemming or lemmatization: reducing a word to its base form
- Removing punctuation
- Part-of-Speech tagging: POS assigns a tag to each word (e.g. NOUN)
- Named Entity Recognition (NER): Identifying and classifying named entities in text

Text processing frameworks

NLTK

- Built as a toolkit for researchers and students
- Modular and allows you to choose between different algorithms for the same task
- Generally slower because it is written in pure Python and requires manual step-by-step processing
- Requires you to manually tokenize, then tag, then chunk

spaCy

- Built for performance and production
- Opinionated, providing one high-quality, pre-trained model for each task
- Highly optimized (written in Cython) and significantly faster for large-scale text analysis
- Automatically runs a full pipeline (Tokenization → POS → NER) when you load a string into the nlp object, returning a clean Doc object with all attributes ready

Practice time

Let’s get hands-on and practice the NLP pipeline!

Open your Google Colab exercise notebook.

Solve exercises 7 – 9.

3. Vectorization

The need for numerical representations

- Machines can only perform calculations on numbers
- Transforming text into a vector space (also known as Word Embeddings) is the bridge between human language and mathematical analysis
- Vectorization allows us to calculate the similarity between texts
- Vector spaces organize text data into a structured format where every word becomes a "feature"

Bag of Words (BoW)

- Counting the frequency of each word in a document while completely ignoring grammar and word order.
- Vocabulary of all unique words and represents each document as a vector where each dimension corresponds to a word's count
- **Simplicity**: It is easy to understand and compute, making it a great baseline for text classification
- **High Dimensionality**: The resulting vectors are very "sparse" (mostly zeros) and can become massive as your vocabulary grows
- **Loss of Context**: Because it ignores order, "The dog bit the man" and "The man bit the dog" result in identical vectors

TF-IDF (Term Frequency-Inverse Document Frequency)

TF-IDF improves upon Bag of Words by penalizing common words (like "the" or "is") and rewarding rare, meaningful words that characterize a specific document.

- **TF (Term Frequency)**: Measures how often a word appears in a specific document.
- **IDF (Inverse Document Frequency)**: Reduces the weight of words that appear frequently across *all* documents in your collection.
- **Insight**: It highlights "keywords" that make a document unique, which is essential for search engines and information retrieval.

(Dense) Word Embeddings

Unlike BoW or TF-IDF, which create sparse vectors, embeddings create "Dense" vectors where words are represented by a fixed set of numbers (e.g., 300 dimensions) that capture their meaning.

- **Semantic Relationships**: Words with similar meanings are mathematically "close" in the vector space (e.g., "King" is to "Queen" as "Man" is to "Woman").
- **Contextual Learning**: Algorithms like Word2Vec learn by looking at the "neighbors" of a word in a sentence.
- **Efficiency**: These models provide a rich representation of language in a compact numerical format that deep learning models can easily consume

Similarity measures

Once text is converted into vectors (whether via TF-IDF or Embeddings), we use mathematical formulas to measure how "similar" two documents are.

- **Cosine Similarity**: The most common measure in NLP; it calculates the cosine of the angle between two vectors.
- **Magnitude Independent**: It focuses on the *direction* of the vectors rather than their length, meaning a short document and a long document can still be considered highly similar if they use the same vocabulary.
- **Scale**: The result is a value between 0 (completely different) and 1 (identical).

Practice time

Let’s get hands-on and practice vectorization!

Open your Google Colab exercise notebook.

Solve exercises 10 – 12.

4. Text classification and topic modelling

Text Classification Overview

Text classification involves assigning predefined categories to text documents. The approach changes depending on whether you have labeled data or not.

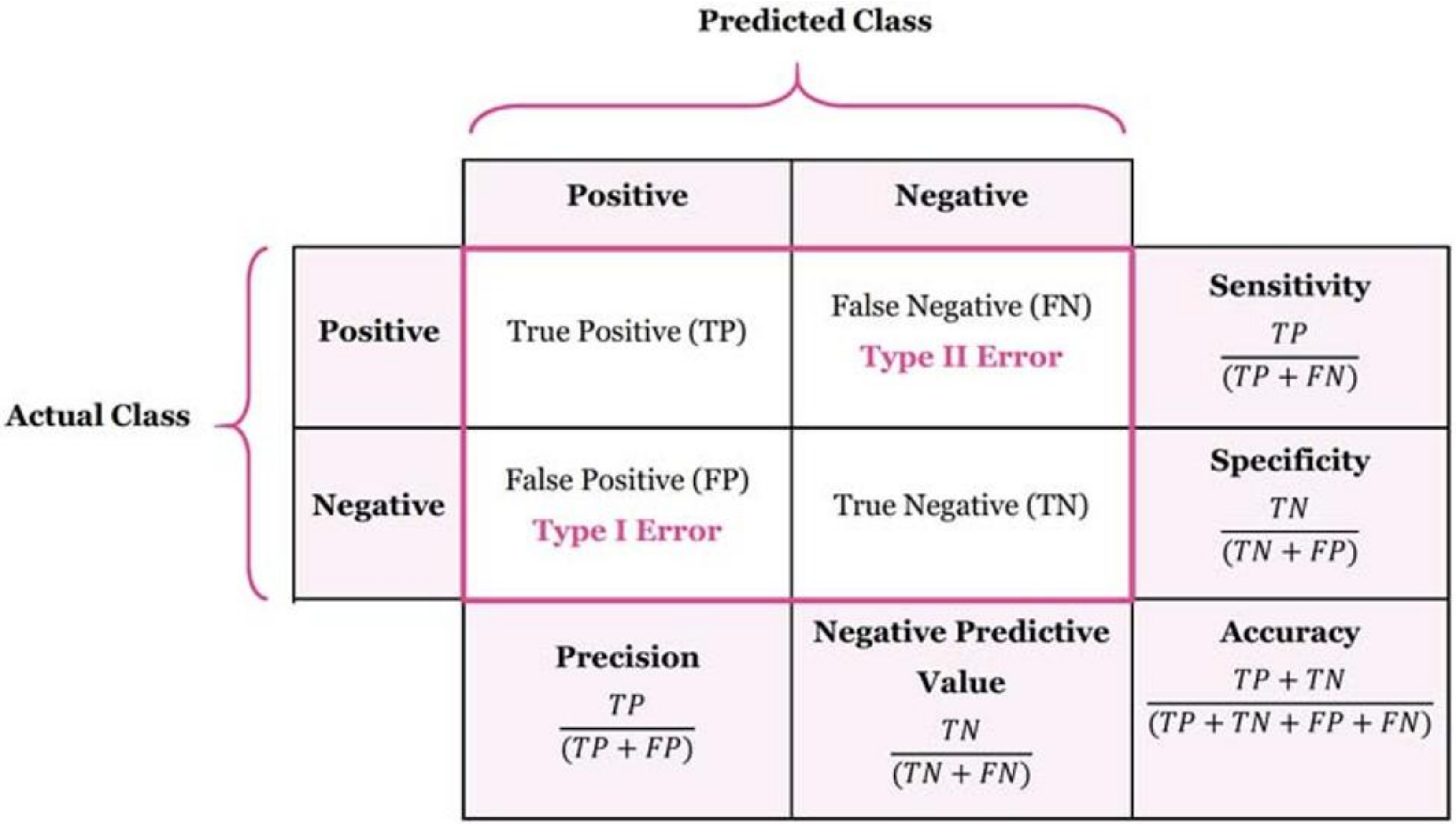
- **Supervised Learning:** The most common approach; the model is trained on a "labeled" dataset where each document has a known category (e.g., "Spam" vs. "Not Spam").
- **Unsupervised Learning:** The model finds patterns or clusters in text without any prior labels (e.g., grouping research papers by similar vocabulary).

Classification Metrics (1)

To know if your classification model is working, you must measure its performance using a confusion matrix and specific statistical scores.

- **Accuracy:** The percentage of total guesses that were correct; can be misleading if your classes are imbalanced.
- **Precision:** "Of all items predicted as Category A, how many were actually Category A?" (Focuses on avoiding False Positives).
- **Recall (Sensitivity):** "Of all actual Category A items, how many did the model find?" (Focuses on avoiding False Negatives).
- **F1-Score:** The harmonic mean of Precision and Recall; provide a single balanced score for model quality.

Classification Metrics (2)



		Predicted Class		
		Positive	Negative	
Actual Class	Positive	True Positive (TP)	False Negative (FN) Type II Error	Sensitivity $\frac{TP}{(TP+FN)}$
	Negative	False Positive (FP) Type I Error	True Negative (TN)	Specificity $\frac{TN}{(TN+FP)}$
		Precision $\frac{TP}{(TP+FP)}$	Negative Predictive Value $\frac{TN}{(TN+FN)}$	Accuracy $\frac{TP+TN}{(TP+TN+FP+FN)}$

Source: https://medium.com/@m.virk1/classification-metrics-65b79bfdd776

Sentiment Analysis

- Sentiment analysis is a sub-field of classification that detects the "emotional tone" of text.
- VADER (Valence Aware Dictionary and sEntiment Reasoner) is a rule-based tool specifically tuned for social media and short texts.
  - **Lexicon-Based:** VADER uses a dictionary where words are pre-ranked as positive or negative.
  - **Intensity Aware:** It understands that "EXCELLENT!!!" is more positive than "excellent" due to capitalization and punctuation.
  - **Context Handling:** It accounts for "but" (which shifts sentiment) and "not" (which negates it)

Topic Modelling

- Topic modeling is an unsupervised technique used to discover "hidden" themes across a large collection of documents
- LDA (Latent Dirichlet Allocation) is the most popular algorithm for this
  - **Probabilistic Model:** LDA assumes every document is a "mixture" of topics and every topic is a "mixture" of words.
  - **Word Clusters:** It identifies words that frequently appear together (e.g., "patient," "doctor," and "hospital" might form a "Healthcare" topic).
  - **Discovery:** You do not provide labels; the algorithm suggests groups, and the researcher interprets what those groups represent.

Practice time

Let’s get hands-on and practice text classification and topic modelling!

Open your Google Colab exercise notebook.

Solve exercises 13 – 14.

5. Deep Learning methods

5. Deep Learning overview

Traditional NLP (like BoW or LDA) relies on manual feature engineering. Deep Learning uses multi-layered neural networks to automatically learn hierarchical representations of text.

- **Neural Networks:** Models like Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTMs) were designed to process sequences of text by maintaining a "memory" of previous words.
- **Word Embeddings as Input:** Instead of sparse counts, these models take dense vectors (like Word2Vec) as input to understand semantic context.
- **End-to-End Learning:** Deep learning models can map raw text directly to a target output (like a translation or a category) without intermediate steps like lemmatization

Transformers

Introduced in the "Attention is All You Need" paper, Transformers replaced RNNs by using a mechanism called **Self-Attention**, allowing the model to look at all words in a sentence simultaneously.

- **Parallelization:** Unlike RNNs that process word-by-word, Transformers process entire sequences at once, making training much faster on GPUs.
- **Self-Attention:** The model learns which words in a sentence are most relevant to each other (e.g., in "The animal didn't cross the street because **it** was too tired," the model learns "it" refers to "animal").
- **Foundation for LLMs:** This architecture is the core of modern Large Language Models like BERT, GPT, and Claude

Hugging Face Ecosystem

Hugging Face is the "GitHub of AI." It provides an open-source library (transformers) and a "Hub" where thousands of pre-trained models are shared.

- **Model Hub:** Access to state-of-the-art models for text, image, and audio classification, translation, and summarization.
- **Tokenizers:** Specialized tools that convert text into the specific numerical format (input IDs) required by a specific model.
- **Transfer Learning:** Instead of training a model from scratch, you download a "base" model that already understands language and "fine-tune" it on your specific research data

Hugging Face Pipelines

Pipelines are the easiest way to use Deep Learning for NLP. They wrap the complex steps of preprocessing, model inference, and post-processing into a single line of code.

- **Task-Specific:** You simply name the task (e.g., "sentiment-analysis", "summarization", "ner") and the library handles the rest.
- **Multi-Lingual:** Many pipelines support hundreds of languages out of the box.
- **Zero-Configuration:** You don't need to define the neural network architecture; Hugging Face selects a high-performing default model for you

Practice time

Let’s get hands-on and practice deep learning methods!

Open your Google Colab exercise notebook.

Solve exercises 15 – 16.

6. NLP best practices for researchers

Reproducability

Reproducibility ensures that other researchers can achieve the same results using the same methodology. In Deep Learning, this is challenging due to the stochastic (random) nature of training algorithms.

- **Code & Data Versioning:** Use tools like Git for code and DVC for large datasets to track every change.
- **Environment Management:** Share exact library versions using Docker containers or environment files (e.g., requirements.txt) to avoid dependency conflicts.
- **Documentation:** Log all hyperparameters (learning rate, batch size) and hardware specs used for training.
- **Controlling Randomness:** Set fixed seeds for all random number generators to ensure deterministic outcomes

Explainability

Explainable AI (XAI) seeks to make "black-box" models transparent. It helps build trust and identify why a model made a specific prediction.

- **LIME (Local Interpretable Model-agnostic Explanations):** Explains a single prediction by creating a simpler, "surrogate" model around that specific instance.
- **SHAP (SHapley Additive exPlanations):** Uses game theory to assign each feature (word) a value representing its contribution to the final prediction.
- **Local vs. Global:** LIME is best for rapid, instance-specific explanations, while SHAP provides both local and global insights into overall model behavior

Bias and Hallucinations

- **Bias**: Models learn and amplify social or statistical skewed patterns present in their training data. This can lead to unfair treatment of underrepresented groups.
- **Hallucinations**: The model generates fluent, plausible-sounding text that is factually incorrect or unsupported by its context.
- **Mitigation**: Use high-quality, diverse data, implement human oversight, and use structured prompting strategies like "Chain-of-Thought"

Data Privacy in Public Interfaces

When using public AI tools or APIs, sensitive information can be unintentionally exposed or stored by the provider.

- **Data Leakage:** Public models may "remember" sensitive prompt data and inadvertently show it to other users in later conversations.
- **PII Masking:** Personally Identifiable Information (names, IDs) should be redacted or masked before being sent to an external API.
- **Enterprise-Grade APIs:** Use commercial versions of AI tools that offer "Zero Data Retention" policies, ensuring your prompts are not used for retraining.
- **Encryption:** Store sensitive embeddings in encrypted databases to prevent "inversion attacks" that try to reconstruct the original text from numerical vectors.

Thank you very much for participating!

Further Questions?