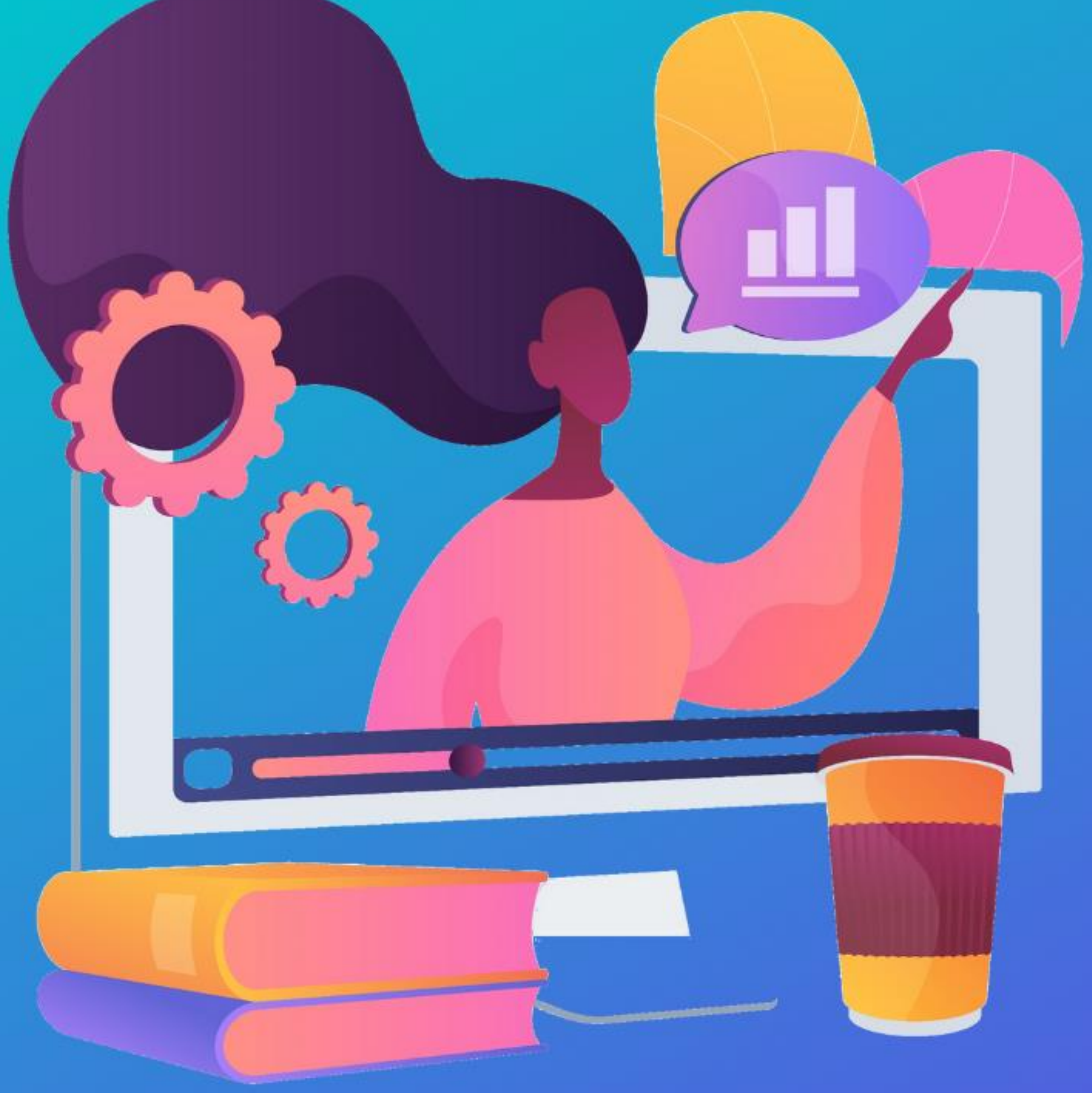


# MBAZA NLP COMMUNITY

## VIRTUAL TRAINING

## 22-24.06 | 3 - 5 PM

BASIC DATA WRANGLING WITH PANDAS
TEXT PRE-PROCESSING BASICS
NLP MODELING & ALGORITHMS

german cooperation
DEUTSCHE ZUSAMMENARBEIT

Implemented by
giz Deutsche Gesellschaft für Internationale Zusammenarbeit (GIZ) GmbH

Digital Transformation Center Rwanda

FAIR FORWARD
Artificial Intelligence for all.

DIGITAL UMUGANDA

# Why are we here?

**Module 1**
Basic Data Wrangling with Pandas

Any Data Science work you do in the future

**Module 2**
Text Pre-Processing Basics

Any NLP work you do in the future

Dataset cleaning challenge in July

**Module 3**
NLP Modelling & Algorithms

Your introduction to Machine Learning

Community activities on Chatbots and Voice

# Review of Yesterday

**Module 2**
Text Pre-Processing
Basics

- Natural Language Processing (NLP) is an important field of AI and has many applications
- Basic tasks include **Natural Language Understanding (NLU)** and **Generation (NLG)**, as well as **speech processing**
- **Text Pre-Processing** is a crucial step in the NLP pipeline; it is done after data collection and before transforming text and using Machine Learning algorithms
- Basic text pre-processing steps include **cHaNgInG cAsInG**, **~~removing~~ text parts**, and **handling** empty **values**
- **Regular expressions** are a flexible tool to selecting specific text part and dealing with it
- **Parallel datasets** include the same text data, but in two languages

# Learning Outcomes of Today

**Module 3**
NLP Modelling & Algorithms

**You understand:**

- what the scikit-learn library does
- how the TF-IDF approach converts text to numeric information
- which Machine Learning tasks exist
- basics of how Neural Networks function
- how to evaluate your model’s performance
- approaches to improve your model’s performance

**You can:**

- use Machine Learning for classifying news articles
- use scikit-learn for TF-IDF text pre-processing
- split your data into training and test sets
- train a neural network with scikit-learn
- generate an evaluation report with scikit-learn

# NLP Pipeline

## *Module 1*

### Data Collection

- Documents
- Web Pages
- User comments
- …

## *Module 2*

### Text Pre-processing

- Word extraction (tokenization)
- fIx cAsInG
- ~~remove~~ irrelevant characters & words
- handle empty values
- …

## *Module 3*

### Transformation

- TF-IDF
- Advanced methods (e. g. Embeddings)

### Machine Learning Model

- Classification
- Machine Translation
- Natural Language Understanding
- Natural Language Generation
- …

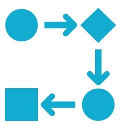
# ML accomplishes a range of tasks

**classification**

Use **labelled data** to train a model to output a **probability** of belonging to a certain class

| y | X |
|---|---|
| *A* | |
| *B* | |
| *C* | |

$$\hat{y} = f(X)$$



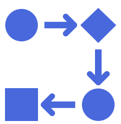
$$\hat{y} = \mathbf{0.828}$$

**regression**

Use **labelled data** to train a model to output a **continuous value**

| y | X |
|---|---|
| *797* | |
| *853* | |
| *816* | |

$$\hat{y} = f(X)$$



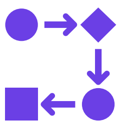
$$\hat{y} = \mathbf{827}$$

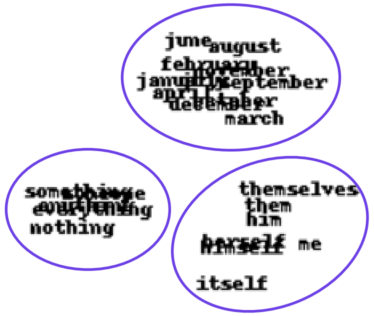
**clustering**

Find structures (clusters) of related objects in **unlabelled data**

| X |
|---|
| |
| |
| |

$$f(X)$$

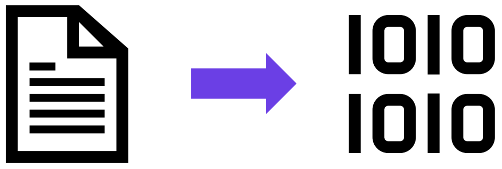




# **scikit-learn** is a Python Machine Learning library

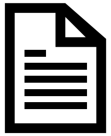
- **Open-source** Python library for **Machine Learning**
- Includes machine learning **algorithms** for various tasks such as classification, regression, and clustering
- Provides functions for important tasks such as model **evaluation**, data **pre-processing**, etc.

# TF-IDF or “how can computers read text?”




- “muraho” has no meaning to a computer
- Challenge: How to *represent* text as numbers?
- Multiple approaches exist
- Here, we cover **Term Frequency – Inverse Document Frequency (TF-IDF)**

**Idea**: Words appearing frequently in a single document, but rarely in a range of documents must be a good indicator of the document’s meaning

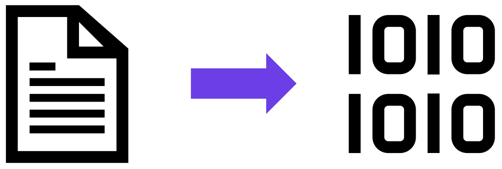


*Document 1: “local soccer team wins a soccer match”*
*Document 2: “a local actor wins oscar”*

Which words are most important?

# TF-IDF or "how can computers read text?"

*Document 1: "local soccer team wins a soccer match"*
*Document 2: "a local actor wins oscar"*




**TF** = Term Frequency (number of times the word occurs in a document)

**IDF** = Inverse Document Frequency $\frac{Number\ of\ documents}{Number\ of\ documents\ where\ word\ appears}$

| | a | actor | local | match | oscar | soccer | team | wins |
|---|---|---|---|---|---|---|---|---|
| TF(Document 1) | 1 | 0 | 1 | 1 | 0 | 2 | 1 | 1 |
| TF(Document 2) | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| IDF | 1 | 2 | 1 | 2 | 2 | 2 | 2 | 1 |

**TF-IDF** = $TF * IDF$

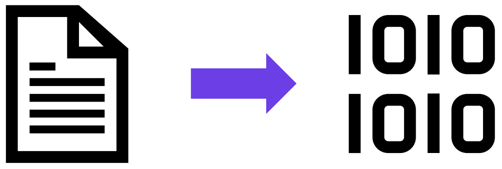
**TF-IDF Matrix**

| | a | actor | local | match | oscar | soccer | team | wins |
|---|---|---|---|---|---|---|---|---|
| Document 1 | 1 | 0 | 1 | **2** | 0 | **4** | **2** | 1 |
| Document 2 | 1 | **2** | 1 | 0 | **2** | 0 | 0 | 1 |

*Please note that this describes TF-IDF in its simplest form. Many variations exist.

# TF-IDF or "how can computers read text?"

**TF-IDF** = $TF * IDF$




**TF-IDF Matrix**

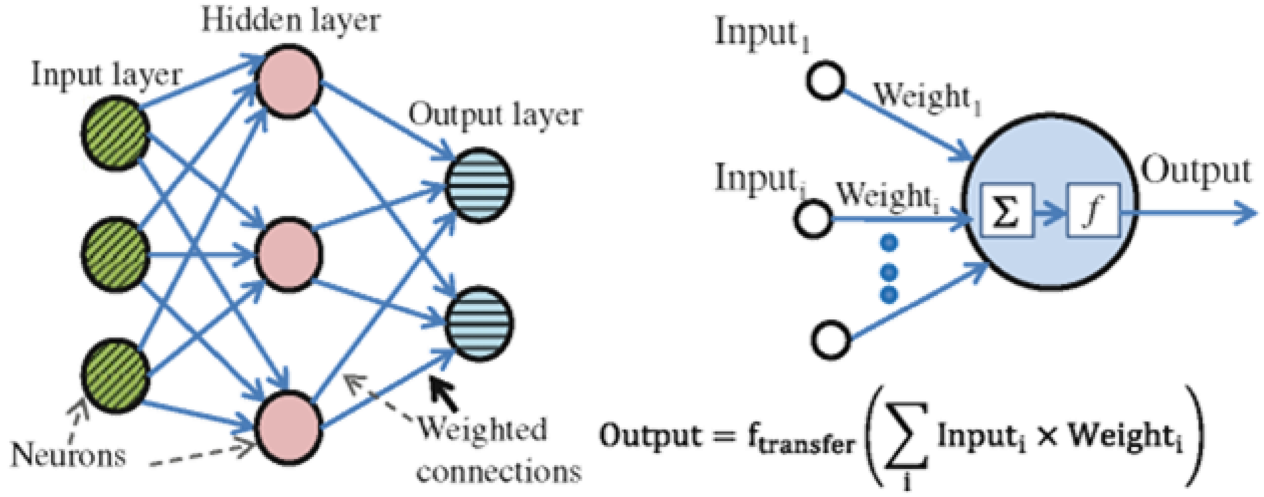
| | a | actor | local | match | oscar | soccer | team | wins |
|---|---|---|---|---|---|---|---|---|
| Document 1 | 1 | 0 | 1 | **2** | 0 | **4** | **2** | 1 |
| Document 2 | 1 | **2** | 1 | 0 | **2** | 0 | 0 | 1 |

## Using TF-IDF in scikit learn

| Task | Code | Output |
|---|---|---|
| Import TF-IDF | `from sklearn.feature_extraction.text import TfidfVectorizer` | |
| Declare vectorizer | `vectorizer = TfidfVectorizer()` | |
| Transform input documents | `vectorizer.fit_transform(X).toarray()` | `array` |
| Get list of words | `vectorizer.get_feature_names()` | `list` |

# Brief Introduction to **Neural Networks**

- Neural networks consist of a range of “neurons” connected by “weights”
- Goal: Set the **weights** to specific values so that a desirable output is reached
- Correct weights are “learned” in a step-by-step procedure called **backpropagation:**
  1. Start with **random weights**
     1. Note: The randomness can be a problem when trying to reproduce results (solution: fix the random state)
  2. Take a **training example**, give as input and measure the output
  3. Check if the output is correct and **update the weights accordingly**
     1. How strongly the weights are updated depends on the learning rate/ step size
  4. **Repeat** from step 2 until a maximum number of iterations of reached




ebrary.net

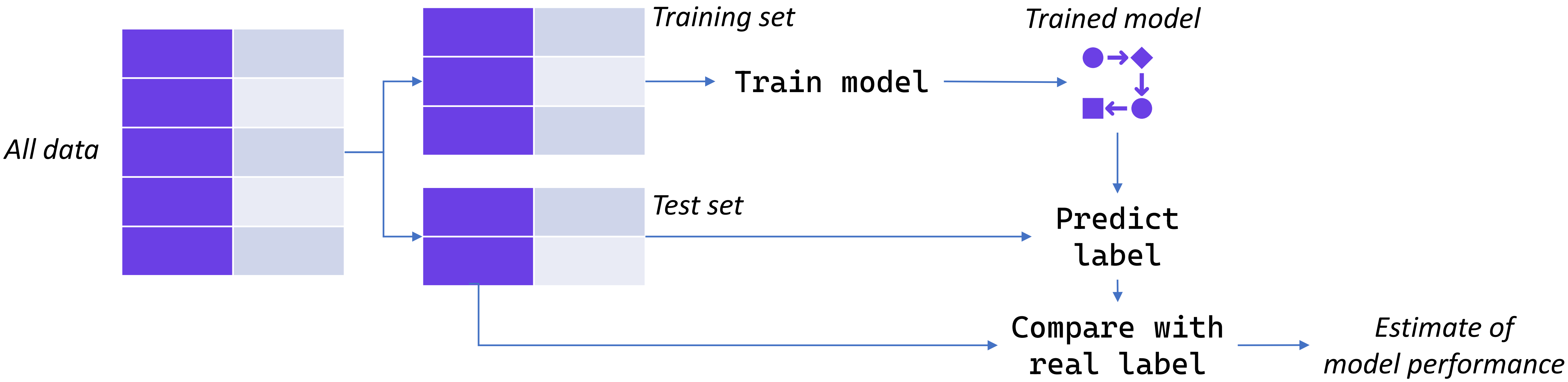
*Please note that this describes neural networks in their simplest form. Many variations exist.

# Using Neural Networks in scikit-learn

| Task | Code | Output |
|---|---|---|
| Import simple neural network for classification | `from sklearn.neural_network import MLPClassifier` | |
| Declare Multi-layer Perceptron (MLP) classifier, a simple form of a neural network | `classifier = MLPClassifier(random_state=1)` | |
| Train classifier | `classifier.fit(X_train, y_train)` | |
| Use classifier for predictions | `classifier.predict(X_test)` | `array` |

# Model Evaluation

- After training your model, it can be used for predictions (the process of generating a predicting is also called **inference**)
- To know how “good” your model is, one typically **splits** the data before training
    - One part is used for training (**training set**)
    - The other part is saved to later generate predictions which can be compared to the real labels (**test set**)
    - This way, we have a good estimate of how well our model performs on **“unseen” data**



# Evaluation metrics

| | | Predicted condition | |
|---|---|---|---|
| | Total population = P + N | Positive (PP) | Negative (PN) |
| Actual condition | Positive (P) | True positive (TP) | False negative (FN) |
| | Negative (N) | False positive (FP) | True negative (TN) |

Wikimedia Commons

| Metric | Description | Formula |
|---|---|---|
| **Accuracy** | What percentage of the model predictions were correct? | $= \frac{TP + TN}{all}$ |
| **Precision** | What percentage of predicted category A (e. g. positive) were actually category A?<br>*“How many predicted items are relevant?”* | $= \frac{TP}{TP + FP}$ |
| **Recall** | What percentage of category A were correctly predicted as category A?<br>*“How many relevant items are predicted?* | $= \frac{TP}{TP + FN}$ |
| **F1 score** | Combination of precision and recall | $= 2 * \frac{precision * recall}{precision + recall}$ |

# Evaluating models with scikit learn

| Task | Code | Output |
| --- | --- | --- |
| Split data into training and test sets | `from sklearn.model_selection import train_test_split`<br><br>`X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=SEED)` | 4x `array` |
| Generate classification report | `from sklearn.metrics import classification_report`<br><br>`print(classification_report(y_true=y_test, y_pred=predictions))` | `str` |

# Ways to improve your model's performance

**Hyperparameter tuning**

Neural networks and other algorithms have many parameters that can be adjusted to influence results

**Try other Machine Learning algorithms**

For any given ML problem, plenty of algorithms apart from neural networks exist

**Use more data**

If available, try using more data so the model can learn better/more insights

# let's get started!

## Open the Colab link

# Report back
# Q & A

# Summary

Module 3
NLP Modelling & Algorithms

- **scikit-learn** is an open-source Python library for Machine Learning
- Machine Learning accomplishes many tasks such as **classification**, **regression** and **clustering**
- **TF-IDF** is a method to **represent text numerically** by using word counts
- **Neural Networks** are a versatile ML method that learns weights to produce a desired outcome
- scikit-learn supports simple forms of neural networks (`MLPClassifier`)
- Models can be **evaluated** by using separate datasets for **training** and **testing**
- There is a range of **evaluation metrics**, incl. accuracy, precision, and recall
- Approaches to **improve your models** include tuning hyperparameters, trying out other ML algorithms, and using more data

# Training Summary

**Module 1** Basic Data Wrangling with Pandas

**Module 2** Text Pre-Processing Basics

**Module 3** NLP Modelling & Algorithms

## You understand:

- what Jupyter Notebooks are and how to use them
- what Natural Language Processing (NLP) is and which applications it has
- basic data structures such as DataFrames and Series and how they relate to data such as parallel datasets
- why text pre-processing is important for NLP applications

## You can:

- use Pandas for importing, inspecting, filtering, combining, and manipulating data
- clean noisy text data using methods such as regular expressions
- use scikit-learn to transform text into a numeric format (TF-IDF), train a neural network for classification, evaluate and improve the model

# Outlook – what else is there?

## Other NLP techniques & algorithms

- Embeddings
- Transformer models

## Other ML/ Deep Learning libraries

- TensorFlow (Keras)
- PyTorch
- Hugging Face
- spaCy, Gensim, NLTK

## Other NLP topics

- Machine Translation
- Speech processing (STT, TTS)
- Chatbots
- Sentiment Analysis

# Join the Mbaza NLP Community!

## WhatsApp

https://chat.whatsapp.com/BRlxzsFiZgsLmK5SBT2XUo

## Slack

https://join.slack.com/t/mbazanlpcommunity/shared_invite/zt-19ie5idhj-f0yWfOBgTKzs7VOKCcr_pw

## GitHub

https://github.com/MBAZA-NLP

## Hugging Face

https://huggingface.co/organizations/mbazaNLP/share/mUKyOkYpSRisRpspbfuwUvoQgWyfdiJYqU