Healthcare NLP Summit

4-5 april 2023

Shortcuts for bootstrapping NLP pipelines

David Berenstein
Developer Advocate Engineer

Agenda

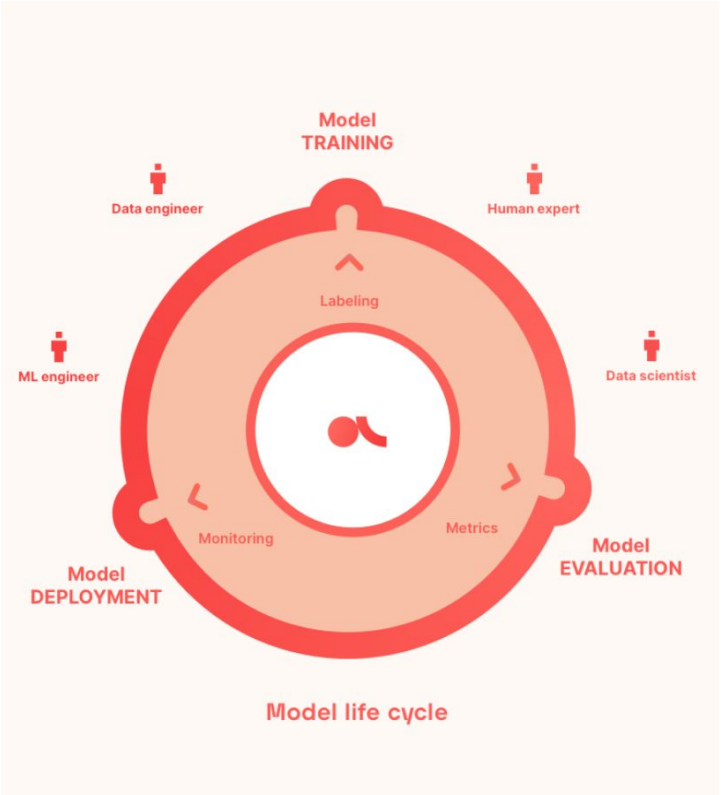
1. Argilla
2. What are shortcuts?
3. Data Exploration
4. Cool Shortcuts
    a. Weak supervision
    b. Few-shot learning
    c. Active learning
5. Wrap-up

Argilla

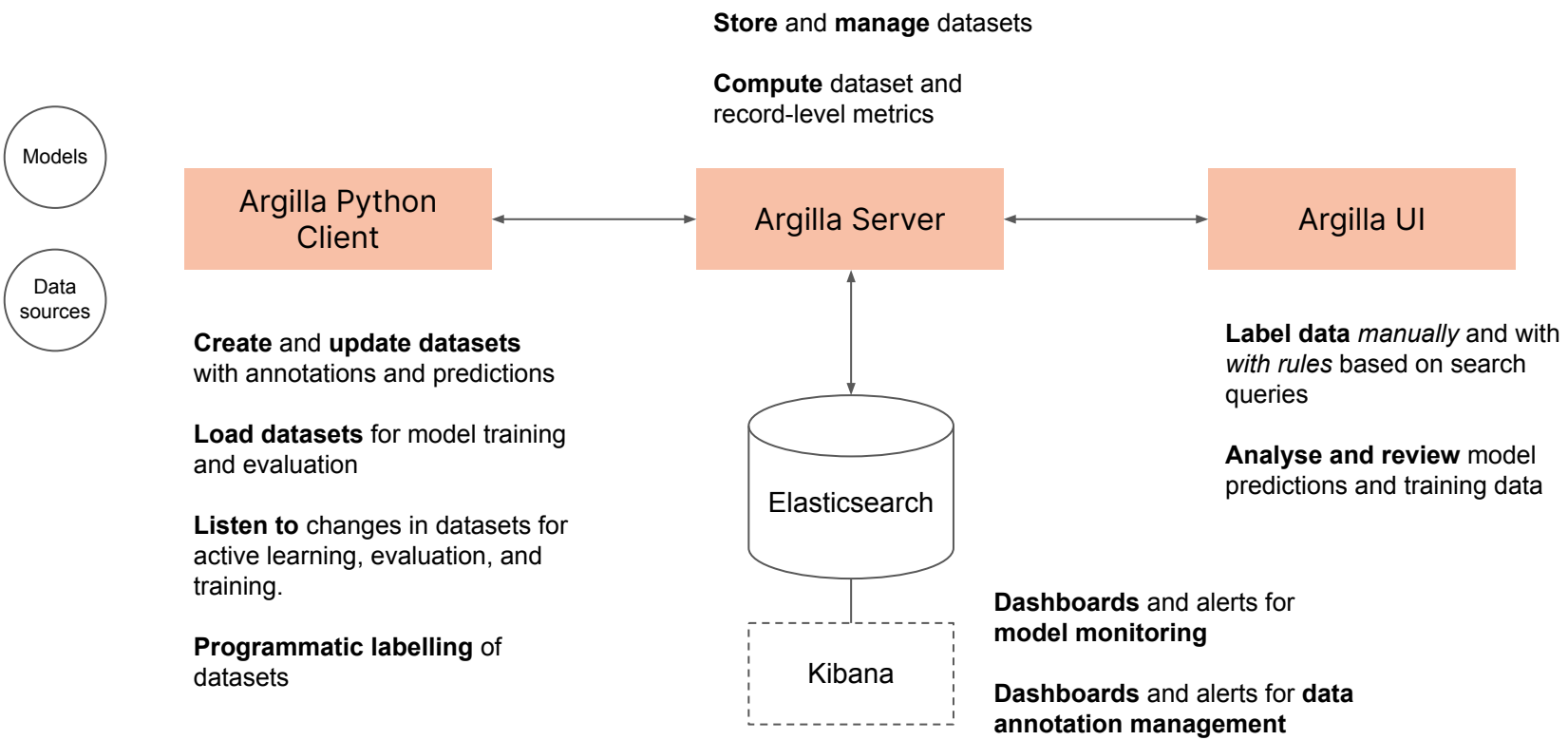
What is Argilla?

Open-source labelling platform for data-centric NLP:

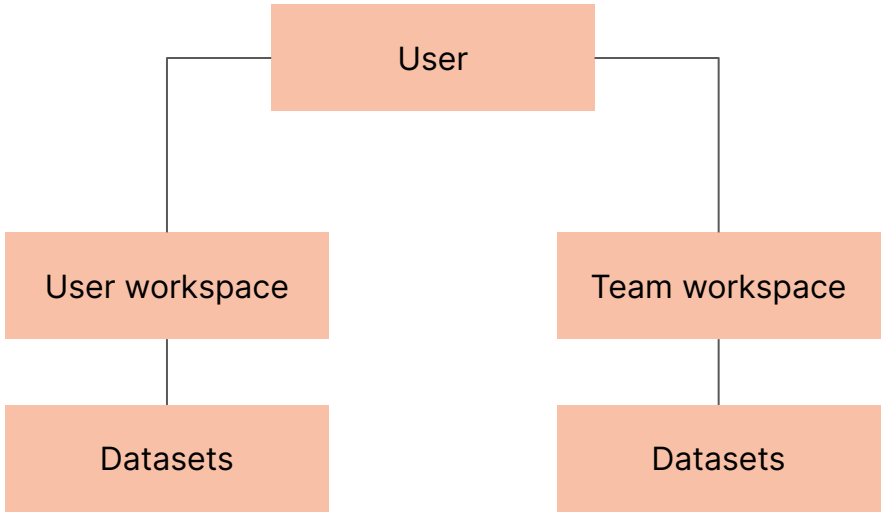
- Quickly build **high quality training data**
- Evaluate and **improve models over time**
- **Enable collaboration** between data teams and domain experts



Argilla components



Argilla Users and Workspaces



`User`

A Argilla user is defined by the following fields:

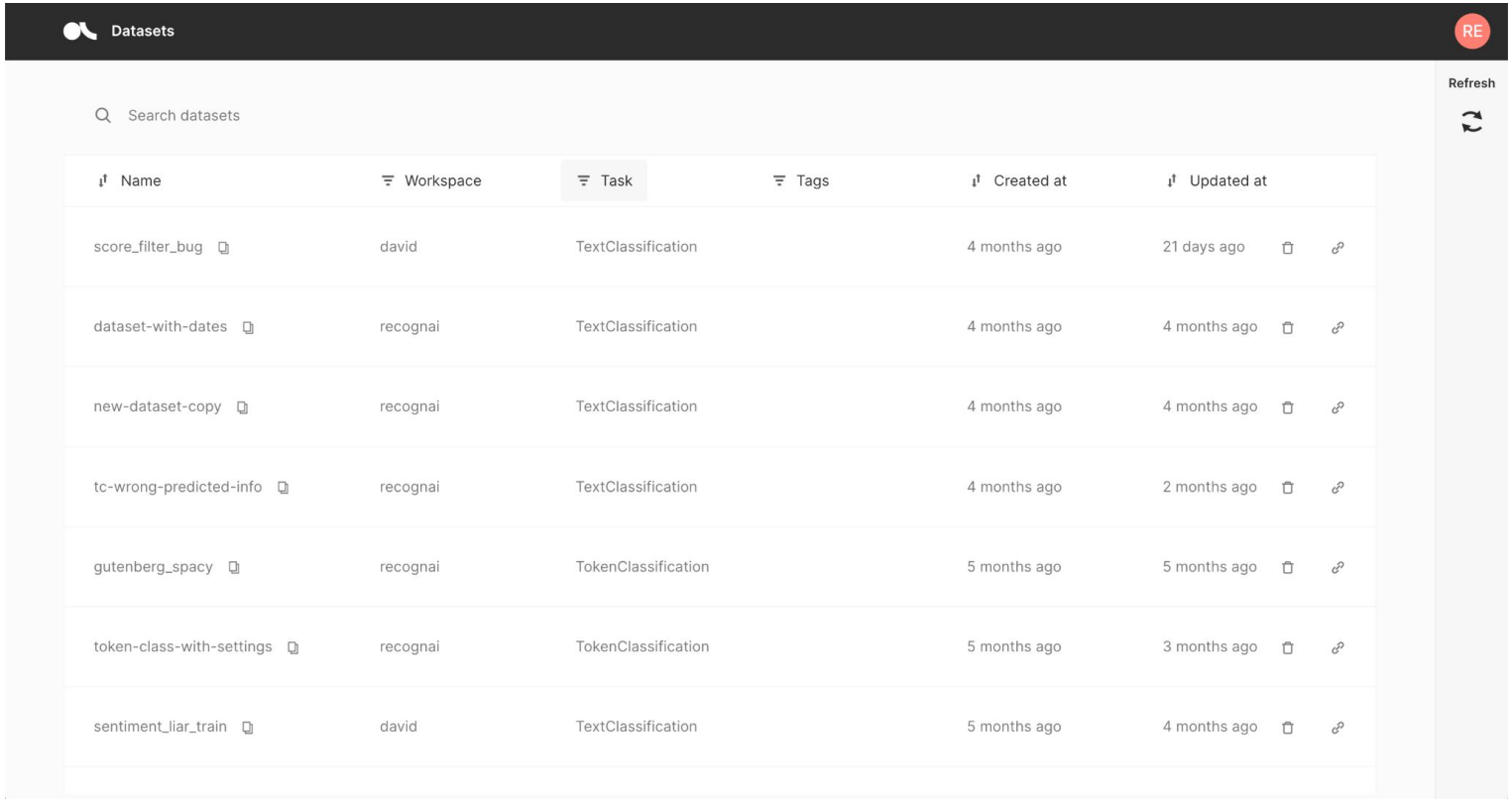
- `username`: The username to use for login into the Webapp.
- `email` (optional): The user's email.
- `fullname` (optional): The user's full name
- `disabled` (optional): Whether this use is enabled (and can interact with Argilla), this might be useful for disabling user access temporarily.
- `workspaces` (optional): The team workspaces where the user has read and write access (both from the Webapp and the Python client). If this field is not defined the user will be a super-user and have access to all datasets in the instance. If this field is set to an empty list `[]` the user will only have access to her user workspace. Read more about workspaces and users below.
- `api_key`: The API key to interact with Argilla API, mainly through the Python client but also via HTTP for advanced users.

`Workspace`

A workspace is a Argilla "space" where users can collaborate, both using the Webapp and the Python client. There are two types of workspace:

- `Team workspace`: Where one or several users have read/write access.
- `User workspace`: Every user gets its own user workspace. This workspace is the default workspace when users log and load data with the Python client. The name of this workspace corresponds to the username.

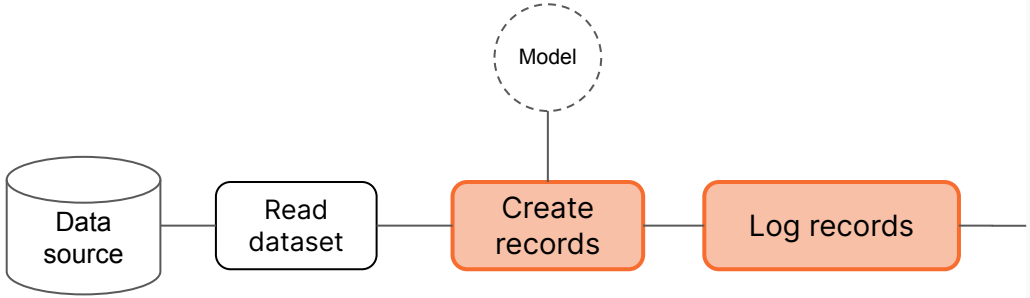
Argilla Users and Workspaces

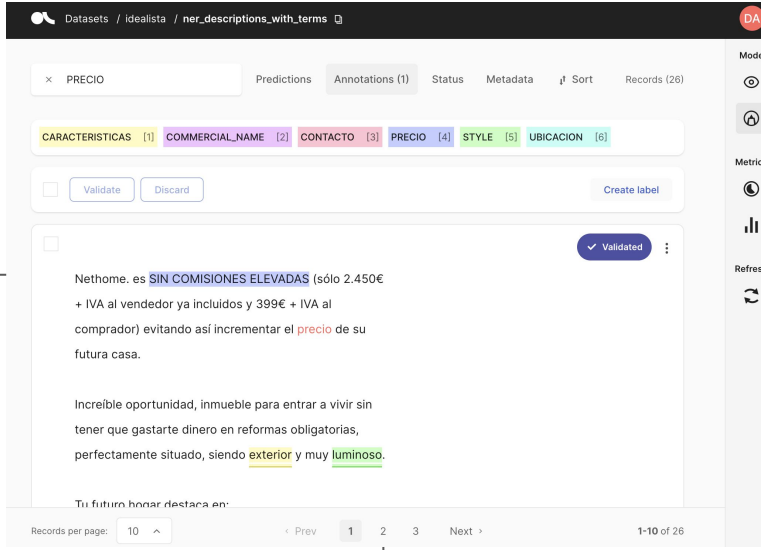


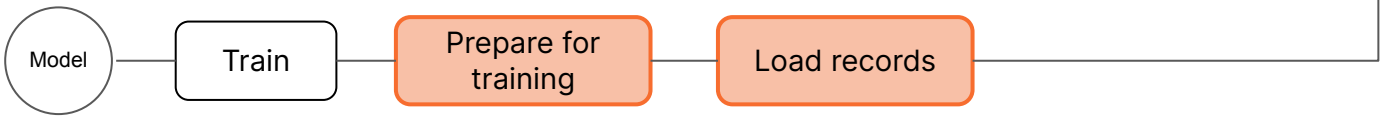
Argilla Workflow: Training

Explore

Label







Argilla Workflow: Monitoring

Explore

Label

Model requests

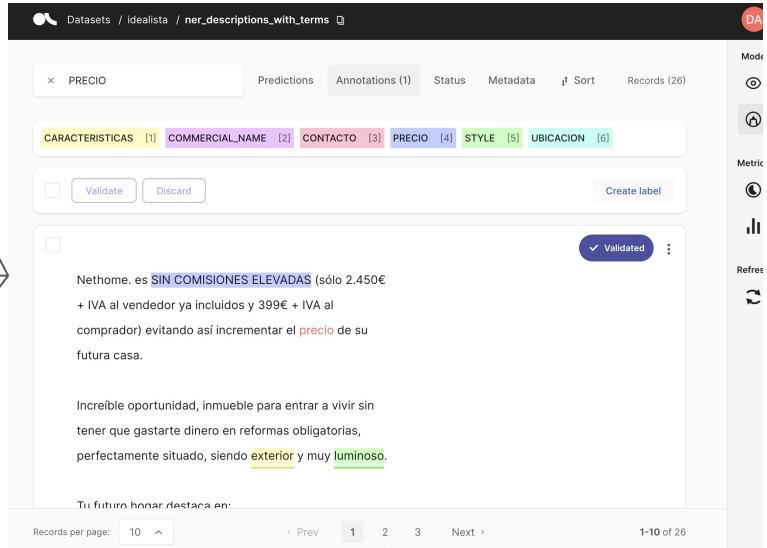
Model

Create records

Log records (async)

monitor

ASGI middleware



Metrics

prepare for training

Load records

Argilla Python: Create records

```
>>> # Single text input
>>> import argilla as rg
>>> record = rg.TextClassificationRecord(
...     text="My first argilla example",
...     prediction=[('eng', 0.9), ('esp', 0.1)]
... )
>>>
>>> # Various inputs
>>> record = rg.TextClassificationRecord(
...     inputs={
...         "subject": "Has ganado 1 million!",
...         "body": "Por usar argilla te ha tocado este premio: <link>"
...     },
...     prediction=[('spam', 0.99), ('ham', 0.01)],
...     annotation="spam"
... )
```

https://docs.argilla.io/en/latest/reference/python/python_client.html#argilla.client.models.TextClassificationRecord

Argilla Python: Log records (write data)

```
>>> import argilla as rg
>>> record = rg.TextClassificationRecord(
...     text="my first argilla example",
...     prediction=[('spam', 0.8), ('ham', 0.2)]
... )
>>> rg.log(record, name="example-dataset")
1 records logged to http://localhost:6900/datasets/argilla/example-dataset
BulkResponse(dataset='example-dataset', processed=1, failed=0)
>>>
>>> # Logging records in the background
>>> rg.log(record, name="example-dataset", background=True, verbose=False)
<Future at 0x7f675a1fffa0 state=pending>
```

https://docs.argilla.io/en/latest/reference/python/python_client.html#argilla.log

Argilla Python: Load records (read data)

```python
import argilla as rg

dataset = rg.load("my_annotated_dataset")
```

```python
dataset = rg.load("my_annotated_dataset")

dataset_for_training = dataset.prepare_for_training()
```

https://docs.argilla.io/en/latest/reference/python/python_client.html#argilla.load

Argilla Python: Prepare dataset for training

```
>>> import argilla as rg
>>> rb_dataset = rg.DatasetForTextClassification([
...     rg.TextClassificationRecord(
...         inputs={"header": "my header", "content": "my content"},
...         annotation="SPAM",
...     )
... ])
>>> rb_dataset.prepare_for_training().features
{'header': Value(dtype='string'),
 'content': Value(dtype='string'),
 'label': ClassLabel(num_classes=1, names=['SPAM'])}
```

https://docs.argilla.io/en/latest/reference/python/python_client.html#argilla.client.datasets.DatasetForTextClassification.prepare_for_training

What are shortcuts?

- “a shorter or quicker way.”
- “a method, procedure, policy, etc., that reduces the time or energy needed to accomplish something.”
- Get a **good enough baseline** and iterate from there.

Data Exploration

Data Exploration - MTSamples

- https://mtsamples.com
- https://huggingface.co/datasets/argilla/medical-domain
- Columns
  - Transcriptions
  - Keywords
- TokenClassification

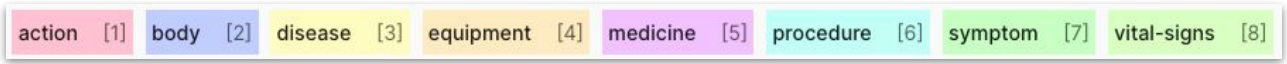
data-from-hub.py

```python
from datasets import load
import argilla as rg

my_dataset = load("argilla/medical-domain")
dataset_rg = rg.read_datasets(my_dataset)
rg.log(dataset_rg, "healthcare-nlp-summit")
```

Data Exploration - MTSamples

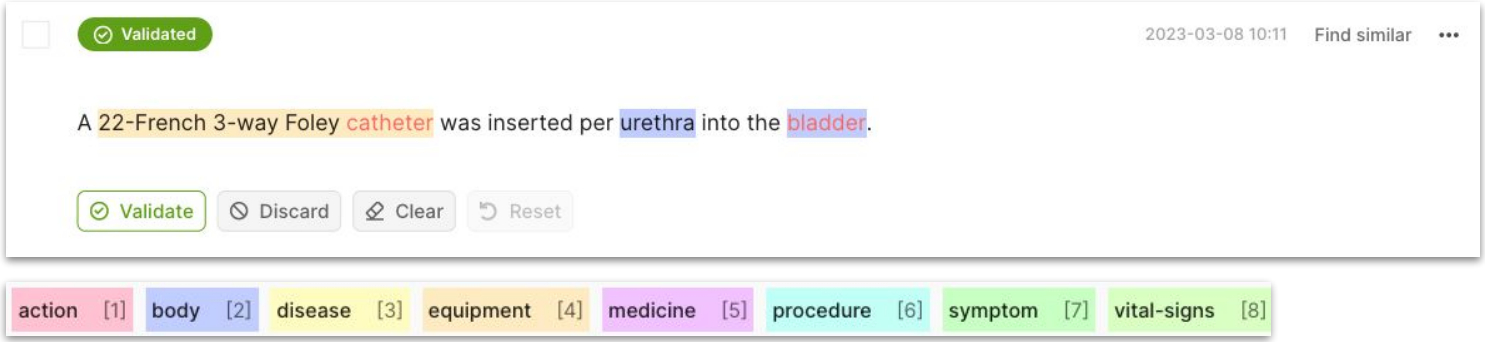
MEDICATIONS: , Her only medication currently is Ortho Tri-Cyclen and the Allegra.,ALLERGIES: , She has no known medicine allergies.,OBJECTIVE:,Vitals: Weight was 130 pounds and blood pressure 124/78.,HEENT: Her throat was mildly erythematous without exudate. Nasal mucosa was erythematous and swollen. Only clear drainage was seen. TMs were clear.,Neck: Supple without adenopathy.,Lungs: Clear.,ASSESSMENT:, Allergic rhinitis.,PLAN:,1. She will try Zyrtec instead of Allegra again. Another option will be to use loratadine. She does not think she has prescription coverage so that might be cheaper.,2. Samples of Nasonex two sprays in each nostril given for three weeks. A prescription was written as well.

['allergy', 'immunology', 'allergic rhinitis', 'allergies', 'asthma',
'nasal sprays', 'rhinitis', 'nasal', 'erythematous', 'allegra', 'sprays',
'allergic']



Data Exploration - Basic UI

- Text Queries (Lucene QL)
  - bladder AND urethra
  - tumor OR cancer
  - surg*
- Regex
- Metadata Filters



Data Exploration - Word2Vec

- Unsupervised => Semi-supervised
- word2vec model PubMed
- https://bio.nlplab.org/

word2vec.py

```python
from gensim.models import KeyedVectors

path = 'PubMed-w2v.bin'
w2v = KeyedVectors.load_word2vec_format(path, binary=True)
```

word2vec_similar.py

```python
w2v.most_similar("stomach")

# [('duodenum', 0.8277196288108826),
#  ('cardia', 0.7727073431015015),
#  ('gastric', 0.7712952494621277),
#  ('rectum', 0.7649236917495728),
#  ('colon', 0.7507137656211853),
#  ('cecum', 0.7502793073654175),
#  ('caecum', 0.7462908029556274),
#  ('jejunum', 0.7376696467399597),
#  ('pylorus', 0.7352787256240845),
#  ('esophagus', 0.7347595691680908)]
```

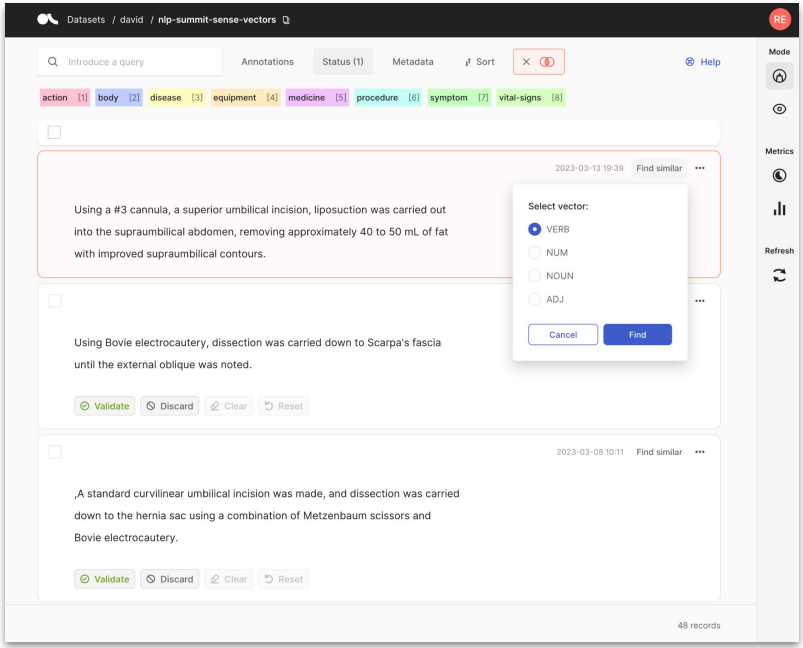
word2vec_similar_to_given.py

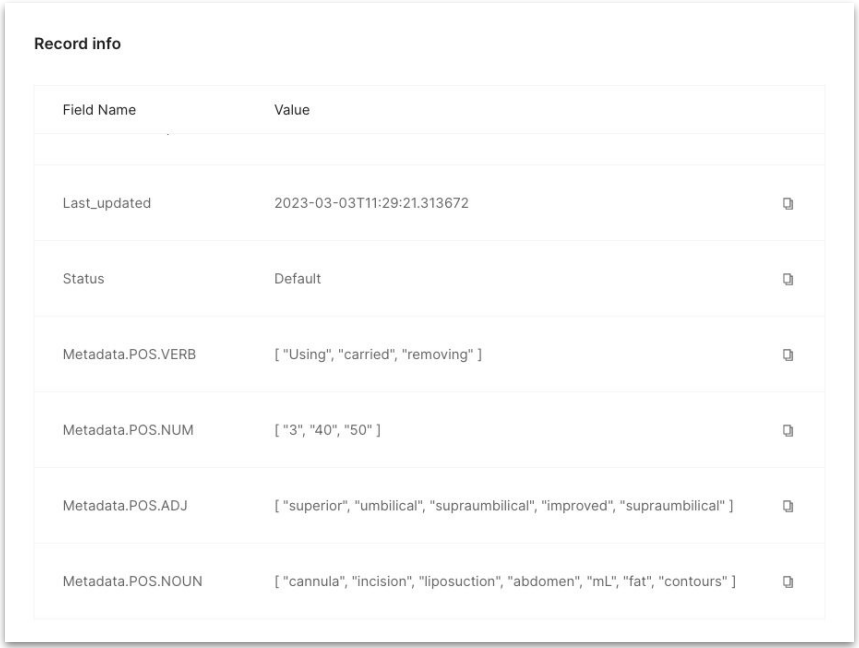
```python
w2v.most_similar_to_given("stomach", ["bladder", "brain"])

# 'bladder`
```

word2vec_similarity.py

```python
w2v["stomach"]

# array([-0.17889705, ..., 0.09800531], dtype=float32)
```

Data Exploration - (sensical) embeddings

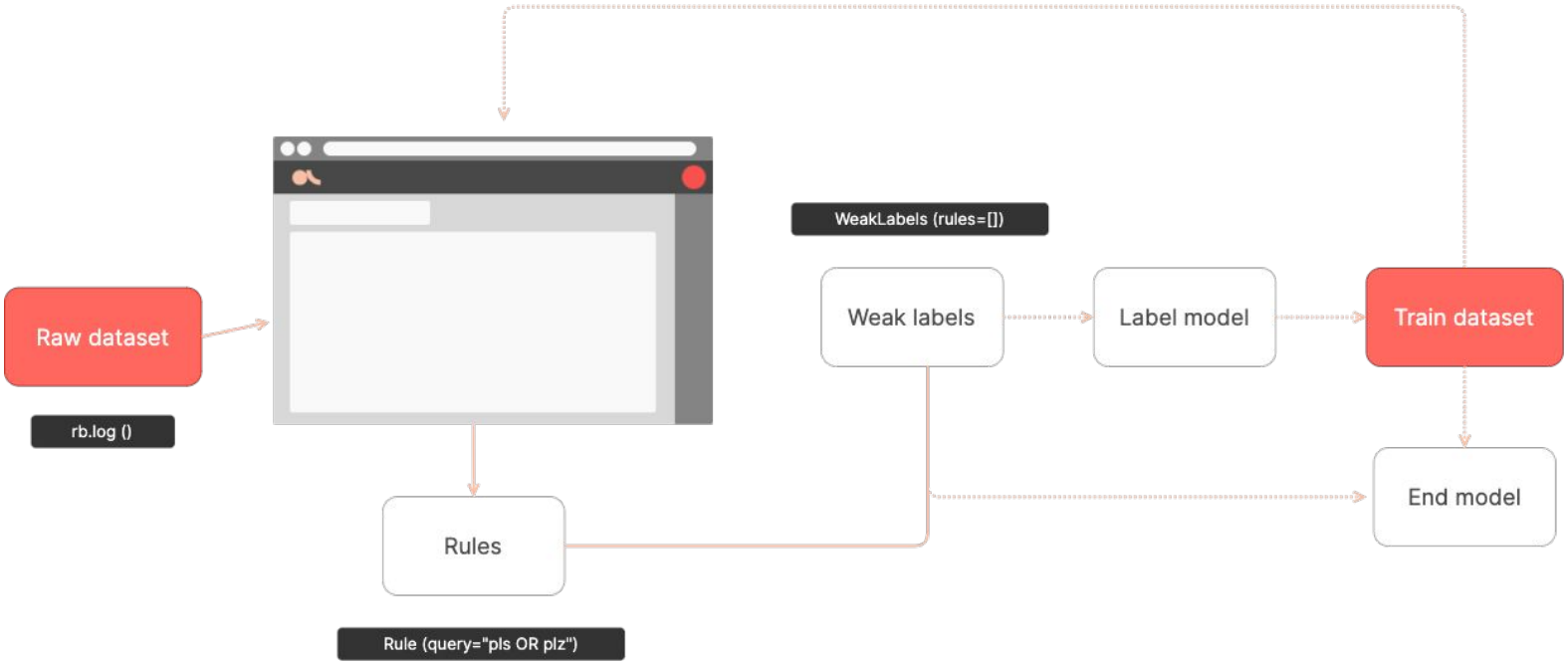




Brief Demo

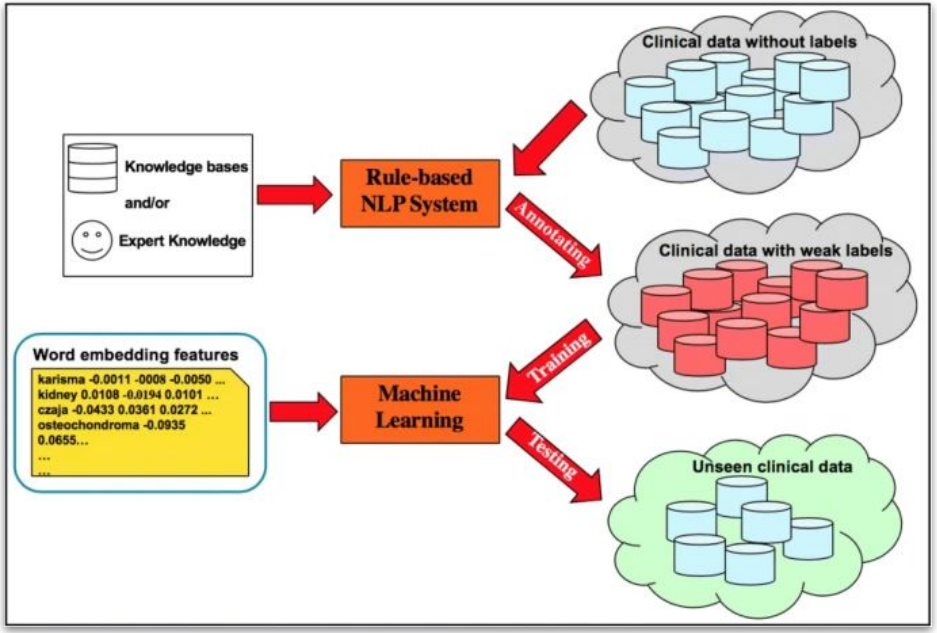
Weak Supervision

Weak supervision: an overview



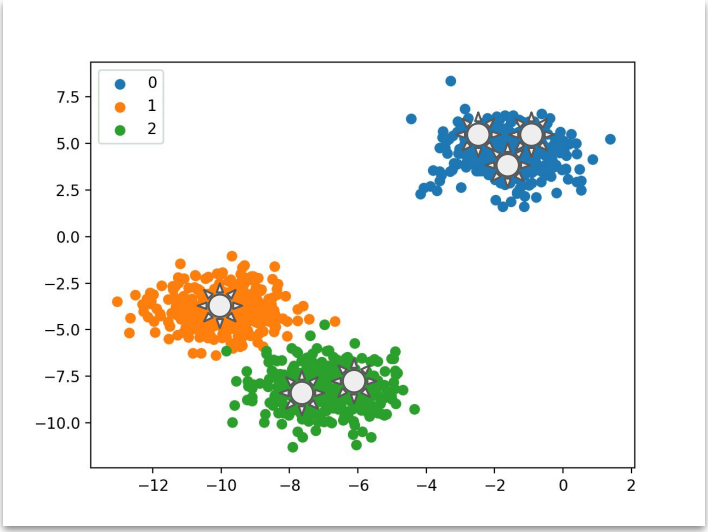
Weak supervision: lexical rules

- Exact Lexical Rules
  - foley AND catheter => equipment
  - tumor OR cancer => disease
  - surg* => operation
- Knowledge base
  - list of medication
- Regex



Weak supervision: semantic rules

- load annotated data
- create kb-word-to-label index
- annotate based on similarity threshold
- add annotations to kb



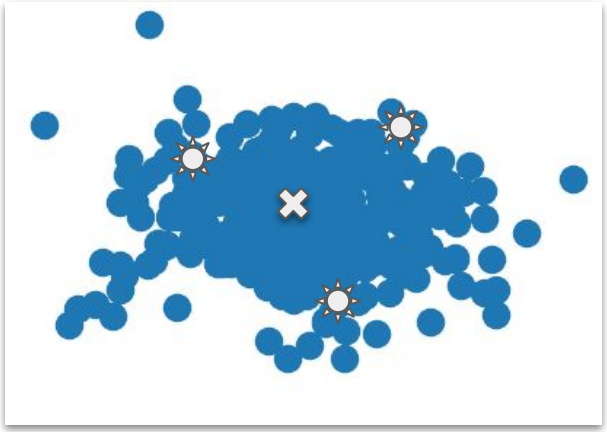
semantic-weak-supervision.py

```python
threshold = 0.75
kb_to_labels = {"Tenckhoff": "EQUIPMENT", "cannula": "EQUIPMENT"}
kb_words = list(kb_to_labels.keys())
text = ["The", "catheter", "was", "removed"]

for token in text:
    if token in w2v:
        most_similar = w2v.most_similar_to_given(token, kb_words)
        similarity = w2v.similarity(most_similar_word, token_text)
        if  similarity ⩾ threshold:
            label = kb_to_labels[most_similar]
            print(f"{token} matches ({most_similar}, {label})")

# "catheter matches (Tenckhoff, EQUIPMENT)"
```

Few-shot learning

Few shot learning - concise-concepts

- groups of words
- create n-most-similar groups
- spaCy or Gensim vectors
- from 30 input words to 15K words
- use as matching patterns



concise-concepts.py

```python
from concise_concepts import Conceptualizer
import spacy
from gensim.models import Word2Vec, KeyedVectors


data = {
    "disease": ["diabetes", "influenza", "pneumonia", "lymphadenopathy"],
    "symptom": ["headache", "fever", "vomiting", "dizzyness"],
    "procedure": ["radiotherapy", "therapy", "chemotherapy"],
    "body": ["elbow", "torso", "foot", "eye", "bladder", "neck"],
    "medicine": ["xanax", "anesthesia", "antibiotics", "insulin"],
    "action": ["incision", "surgery", "operating", "dissection", "cut"],
    "equipment": ["blade", "syringe", "catheter", "pacemaker"],
    "vital-signs": ["temperature", "pulse", "pressure", "sugar"],
}

w2v_path = 'PubMed-w2v.bin'
nlp = spacy.load("en_core_web_md", disable=["ner"])
w2v = KeyedVectors.load_word2vec_format(w2v_path, binary=True)
model = Conceptualizer(
    nlp, data, w2v, json_path="medical-patterns.json"
)
text = "Drowsiness and polyarthralgia are symptoms of the disease Diphtheric."
print(", ".join([f"{ent.text}: {ent.label_}" for ent in model(text).ents]))
# Drowsiness: SYMPTOM, polyarthralgia: SYMPTOM, disease Diphtheric: DISEASE
```

Argilla

Few shot learning - concise-concepts + semantic rules

- 30 words => 15K words => semantic 15K words
- threshold for similarity

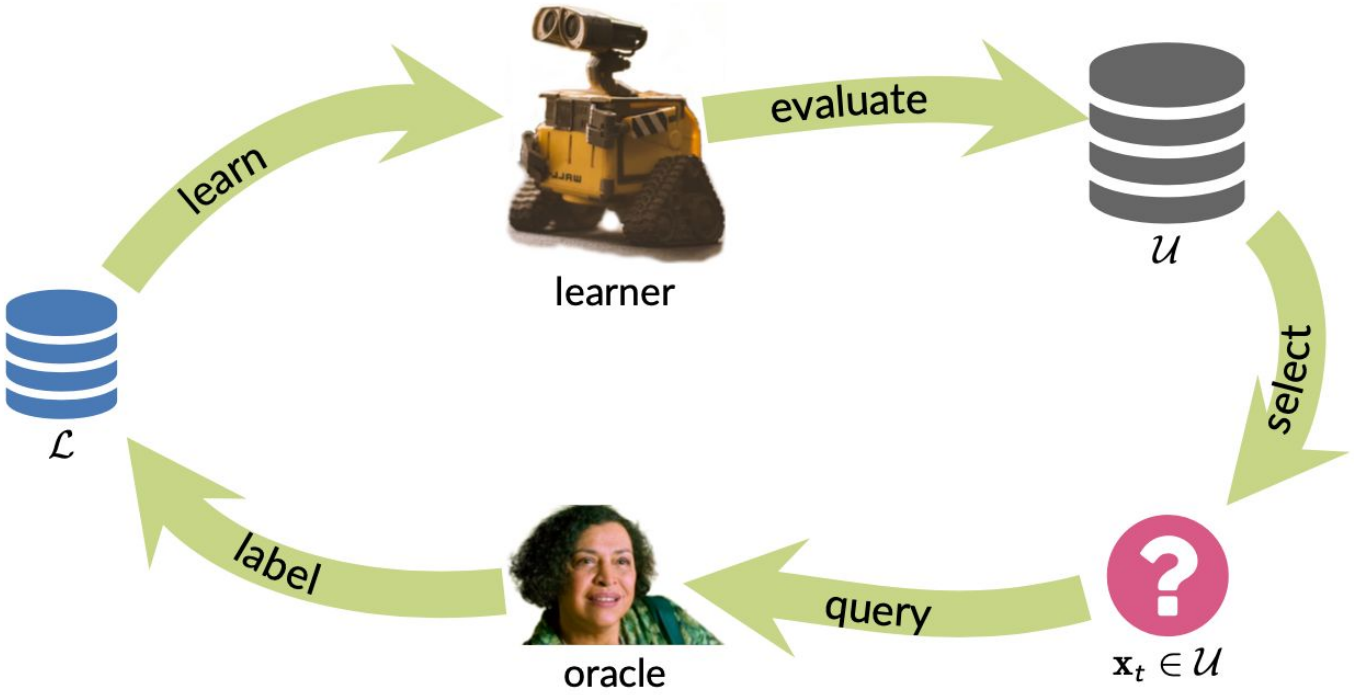
Biopro sizer. The bone was noted to be significantly hardened. The sizer was placed and a large Biopro was deemed to be the correct size implant. The sizer was removed and bar drill was then again used to ream the medullary canal. The hand reamer with a Biopro set was then used to complete the process. The Biopro implant was then inserted and tamped with a hammer and rubber mallet to ensure tight fit. There was noted to be distally increased range of motion after insertion of the implant.,Attention was then directed to the first metatarsal. A long dorsal arm Austin osteotomy was then created. A second osteotomy was then created just plantar and parallel to the first osteotomy site. The wedge was then removed in toto. The area was feathered to ensure high compression of the osteotomy site. The head was noted to be in a more plantar flexed position. The capital fragment was then temporarily fixated with two 0.45 K-wires. A 2.7 × 16 mm screw was then inserted in the standard AO fashion. A second more proximal 2.7 × 60 mm screw was also inserted in a standard AO fashion. With both screws, there was noted to be tight compression at the osteotomy sites.,The K-wires were removed and the areas were then smoothed with reciprocating rash. A screw driver was then used to check and ensure screw tightness. The area was then flushed with copious amounts of sterile saline. Subchondral drilling was performed with a 1.5 drill bit. The area was then flushed with copious amounts of sterile saline. Closure consisted of capsular closure with #3-0 Vicryl followed by subcutaneous closure with #4-0 Vicryl, followed by running subcuticular stitch of #5-0 Vicryl. Dressings consisted of Steri-Strips, Owen silk, 4×4s, Kling, Kerlix, and Coban. A total of 10 cc of 1:1 mixture of 1% lidocaine plain and 0.5% Marcaine plain was injected intraoperatively for further anesthesia. The pneumatic ankle tourniquet was released and immediate hyperemic flush was noted to all five digits of the left foot. The patient tolerated the above procedure and anesthesia well. The patient was transported to PACU with vital signs stable and vascular status intact to the right foot. The patient was given postoperative pain prescription for Vicodin ES and instructed to take 1 q. 4-6h. p.o. p.r.n. pain. The patient was instructed to ice and elevate his left lower extremity as much as possible to help decrease postoperative edema. The patient is to follow up with Dr. X in his office as directed.

word2vec_similarity.py

```
w2v.most_similar("diabetes")

# [('T2DM', 0.8334260582923889),
#  ('T1DM', 0.7870343923568726),
#  ('DM-2', 0.7731763124465942),
#  ('NIDDM', 0.769320547580719),
#  ('DM2', 0.7682093977928162),
#  ('mellitus', 0.7621428966522217),
#  ('non-insulin-dependent', 0.757671058177948),
#  ('prediabetes', 0.751226007938385),
#  ('PIDDM', 0.746285617351532),
#  ('T2D', 0.7360433340072632)]

w2v.most_similar(["diabetes", "pneumonia", "influenza"])

# [('postinfluenza', 0.6467840075492859),
#  ('ARVI', 0.6458147168159485),
#  ('chickenpox', 0.638774037361145),
#  ('A(H1N1)2009', 0.6266157031059265),
#  ('pH1N1-09', 0.6264188289642334),
#  ('A/H1N1v', 0.6234767436981201),
#  ('coronavirus-associated', 0.6231486201286316),
#  ('pH1N1', 0.6223902106285095),
#  ('encephalitis-associated', 0.6206263899803162),
#  ('Influenza', 0.6194723844528198)]
```

Active learning

Active learning: an overview



Active learning

- work smarter
- listen to newly annotated samples
- update your model
- copy annotations

demo

token_copycat.py

```python
from argilla_plugins import token_copycat

plugin = token_copycat(
    "healthcare-nlp-summit",
    copy_annotations=True,
    case_sensitive=True
)
plugin.start()
```

semantic_token_copycat.py

```python
import argilla as rg
from argilla_plugins import semantic_token_copycat
from gensim.models import Word2Vec, KeyedVectors
import spacy

kyu_path = 'PubMed-w2v.bin'
w2v = KeyedVectors.load_word2vec_format(kyu_path, binary=True)
nlp = spacy.load("en_core_web_md", disable=["ner"])
plugin = semantic_token_copycat(
    "nlp-summit-weak-supervisor",
    nlp,
    w2v,
    execution_interval_in_seconds=1
)
plugin.start()
```

Train a model

train_with_spark_nlp.py

```python
import argilla as rg
import sparknlp

ds = rg.load("healthcare-nlp-summit", query="status: Validated")
df = ds.prepare_for_training(framework="spark-nlp", train_size=0.75)

spark = sparknlp.start()
spark_df=spark.createDataFrame(df)
df.columns

# CoNLL format
# ["id", "text", "token", "label"]
```

Wrap up

Argilla

Crosslingual Coreference

crosslingual-coreference.py

```python
from crosslingual_coreference import Predictor

text = (
    "Do not forget about Momofuku Ando! He created instant noodles in Osaka. At"
    " that location, Nissin was founded. Many students survived by eating these"
    " noodles, but they don't even know him."
)

# choose minilm for speed/memory and info_xlm for accuracy
predictor = Predictor(
    language="en_core_web_sm", device=-1, model_name="minilm"
)

predictor.predict(text)["resolved_text"]
predictor.pipe([text])[0]["resolved_text"]
```

English

Do not forget about Momofuku Ando 0 ! He 0 created instant noodles 2 in Osaka 1 . At that location 1 , Nissin was founded. Many students survived by eating these noodles 2 , but they don't even know him 0 .

German

Vergiss Momofuku Ando 0 nicht! Er 0 kreierte Instantnudeln in Osaka 1 . An diesem Standort 1 wurde Nissin 3 gegründet. Viele Studenten 2 überlebten, indem sie 2 diese Nudeln aßen, aber sie 2 kennen ihn 3 nicht einmal.

Danish

Glem ikke Momofuku Ando 0 ! Han 0 skabte instant nudler 2 i Osaka 1 . På det sted 1 blev Nissin grundlagt. Mange elever 3 overlevede ved at spise disse nudler 2 , men de 3 kender ham 0 ikke engang.

Spanish

¡No te olvides de Momofuku Ando! Creó fideos instantáneos 1 en Osaka 0 . En ese lugar 0 , se fundó Nissin. Muchos estudiantes sobrevivieron comiendo estos fideos 1 , pero ni siquiera lo conocen.

French

N'oubliez pas Momofuku Ando 0 ! Il 0 a créé des nouilles instantanées 2 à Osaka 1 . À cet endroit 1 , Nissin a été fondée. De nombreux étudiants 3 ont survécu en mangeant ces nouilles 2 , mais ils 3 ne le connaissent même pas.

Italian

Non dimenticare Momofuku Ando! Ha creato spaghetti istantanei 1 a Osaka 0 . In quel luogo 0 è stata fondata Nissin. Molti studenti sono sopravvissuti mangiando questi noodles 1 , ma non lo 1 conoscono nemmeno.

Dutch

Vergeet Momofuku Ando 0 niet! Hij 0 maakte instantnoedels 2 in Osaka 1 . Op die locatie 1 werd Nissin opgericht. Veel studenten 3 overleefden door deze noedels 2 te eten, maar ze 3 kennen hem 0 niet eens.

Classy-classification

classy-classification.py

```python
from classy_classification import ClassyClassifier

data = {
    "furniture": ["This text is about chairs.",
                  "Couches, benches and televisions.",
                  "I really need to get a new sofa."],
    "kitchen": ["There also exist things like fridges.",
                "I hope to be getting a new stove today.",
                "Do you also have some ovens."]
}

classifier = ClassyClassifier(data=data)
classifier("I am looking for kitchen appliances.")
classifier.pipe(["I am looking for kitchen appliances."])
```

classy_learner.py

```python
from argilla_plugins import classy_learner

plugin = classy_learner(
    name="plugin-test",
    model="all-MiniLM-L6-v2",
    overwrite_predictions=True,
    sample_strategy="fifo",
    min_n_samples=6,
    max_n_samples=20,
    batch_size=1000,
    execution_interval_in_seconds=5,
)
plugin.start()
```

Sources

- Argilla
  - GitHub https://github.com/argilla-io/argilla
  - HuggingFace Deploy Argilla with the click of a button
- Me
  - LinkedIn https://www.linkedin.com/in/david-berenstein-1bab11105/
  - My packages https://github.com/Pandora-Intelligence
  - Volunteering https://bonfari.nl/
- Other Packages
  - SetFit - Few shot learning
  - Skweak - Weak supervision
  - small-text - Active Learning

Argilla

Feedback and questions