Machine Learning Record Matching with Similarity Encoding

with Marius Liebald (Goethe University)

May, 4th 2023

Digitisation and OCR Workshop
Groningen

Pantelis Karapanagiotis (EBS University)

karapanagiotis@ebs.edu

@pi_kappa_

Matching Data is Hard

- Adam et al., 2021 . Data extraction and matching The EurHisFirm experience. Methodological Advances in the Extraction and Analysis of Historical Data.
- Cule, Buelens, Poukens, Annaert, & Richer, 2020 . Data Connecting Case Study (EurHisFirm M6.2).
- Poukens, 2018 . Report on the Inventory of Data and Sources (EurHisFirm D4.2).
- Karapanagiotis, 2019 . Technical document on national data models (EurHisFirm D5.1).

Can we Develop Reusable Tools?

Can we Develop Reusable Tools?

User requirements:

1. meaningful match suggestions
2. in reasonable execution time
3. using information from multiple entity characteristics
4. applicable in different matching contexts
    - in particular for heterogeneous country data

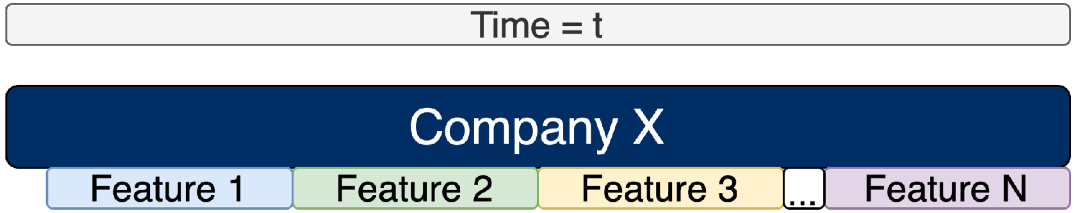
Can we Develop Reusable Tools?

User requirements:

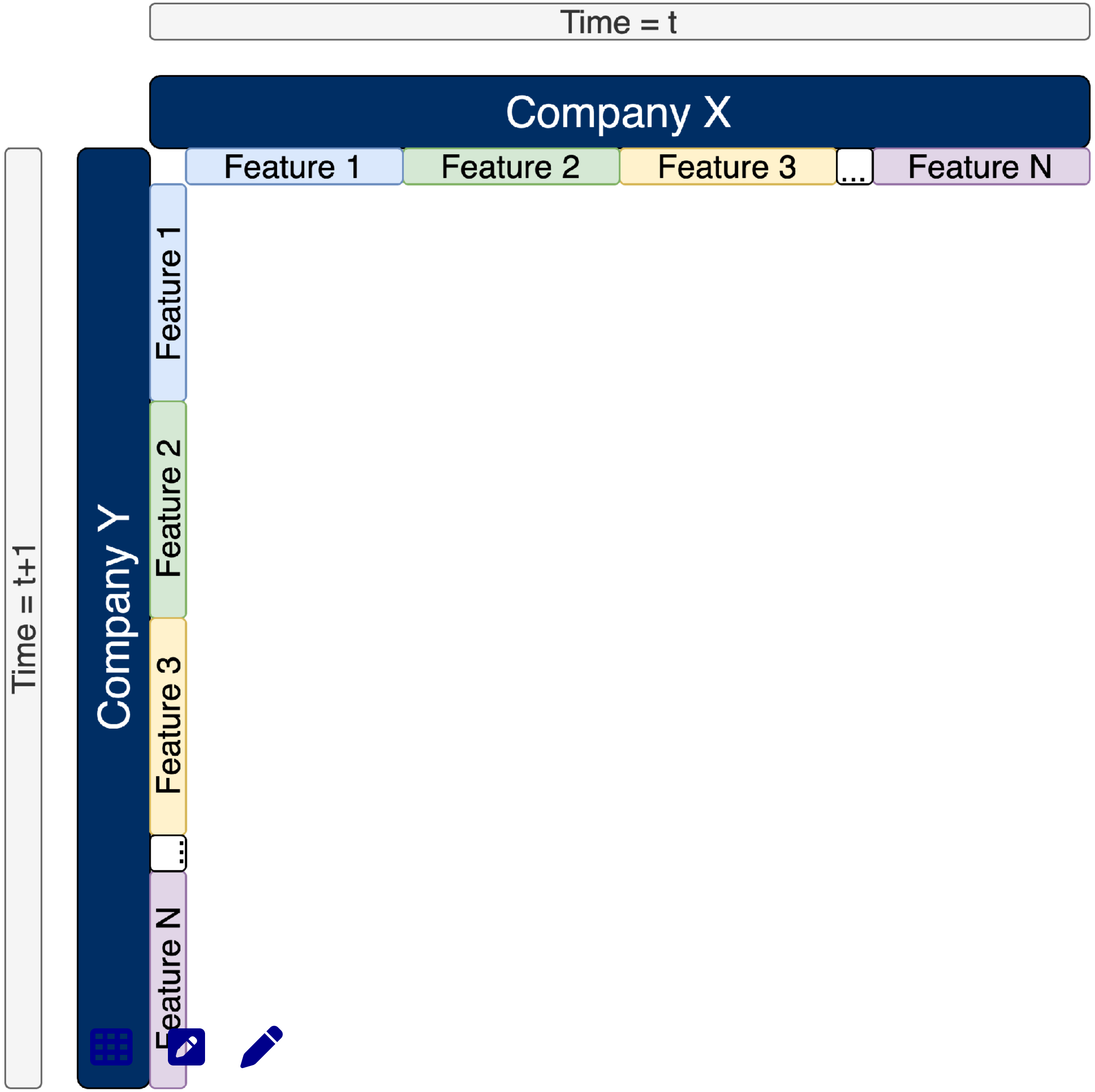
1. meaningful match suggestions
2. in reasonable execution time
3. using information from multiple entity characteristics
4. applicable in different matching contexts
    - in particular for heterogeneous country data

A Matching Problem Example

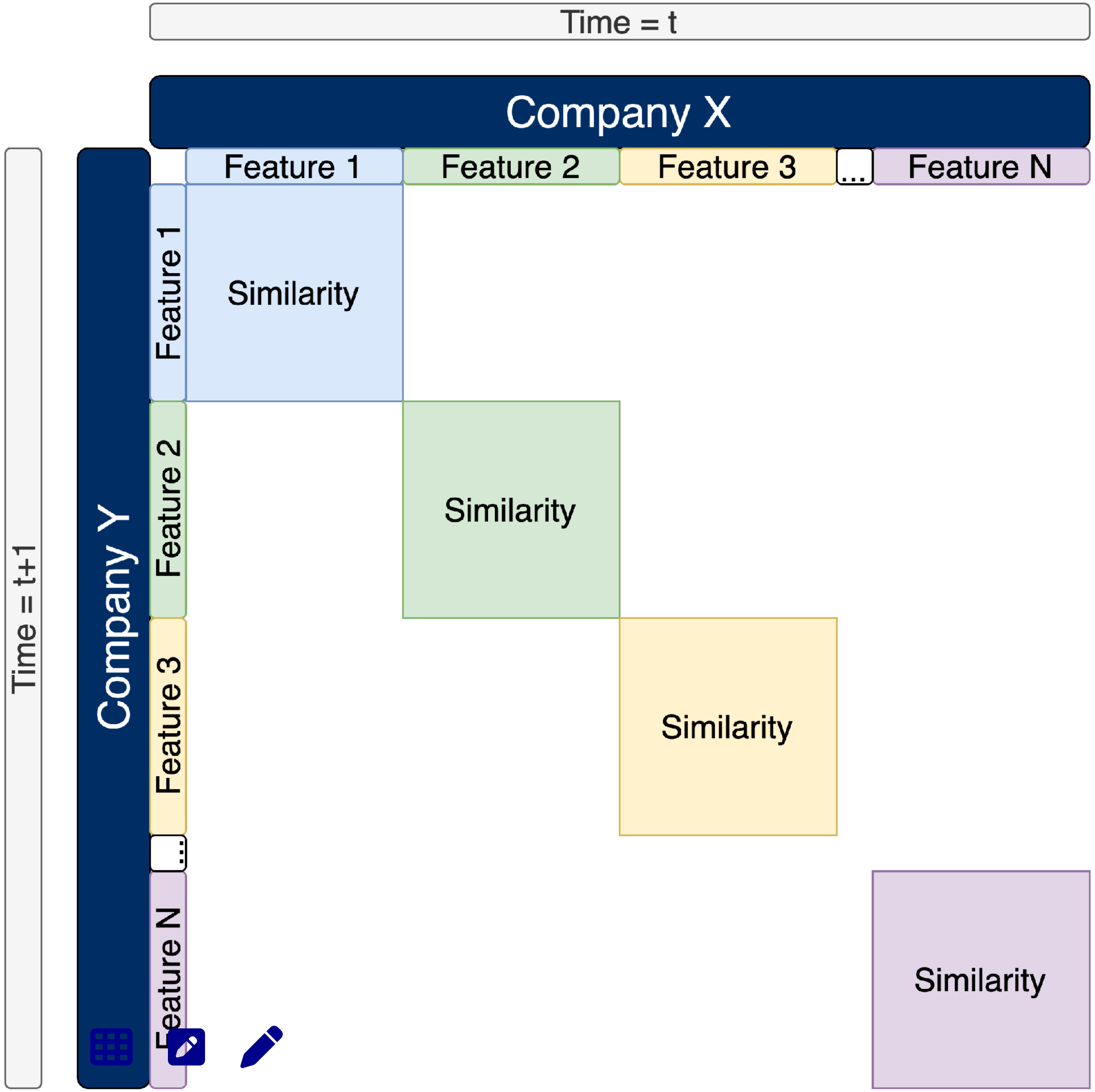
A Matching Problem Example



A Matching Problem Example



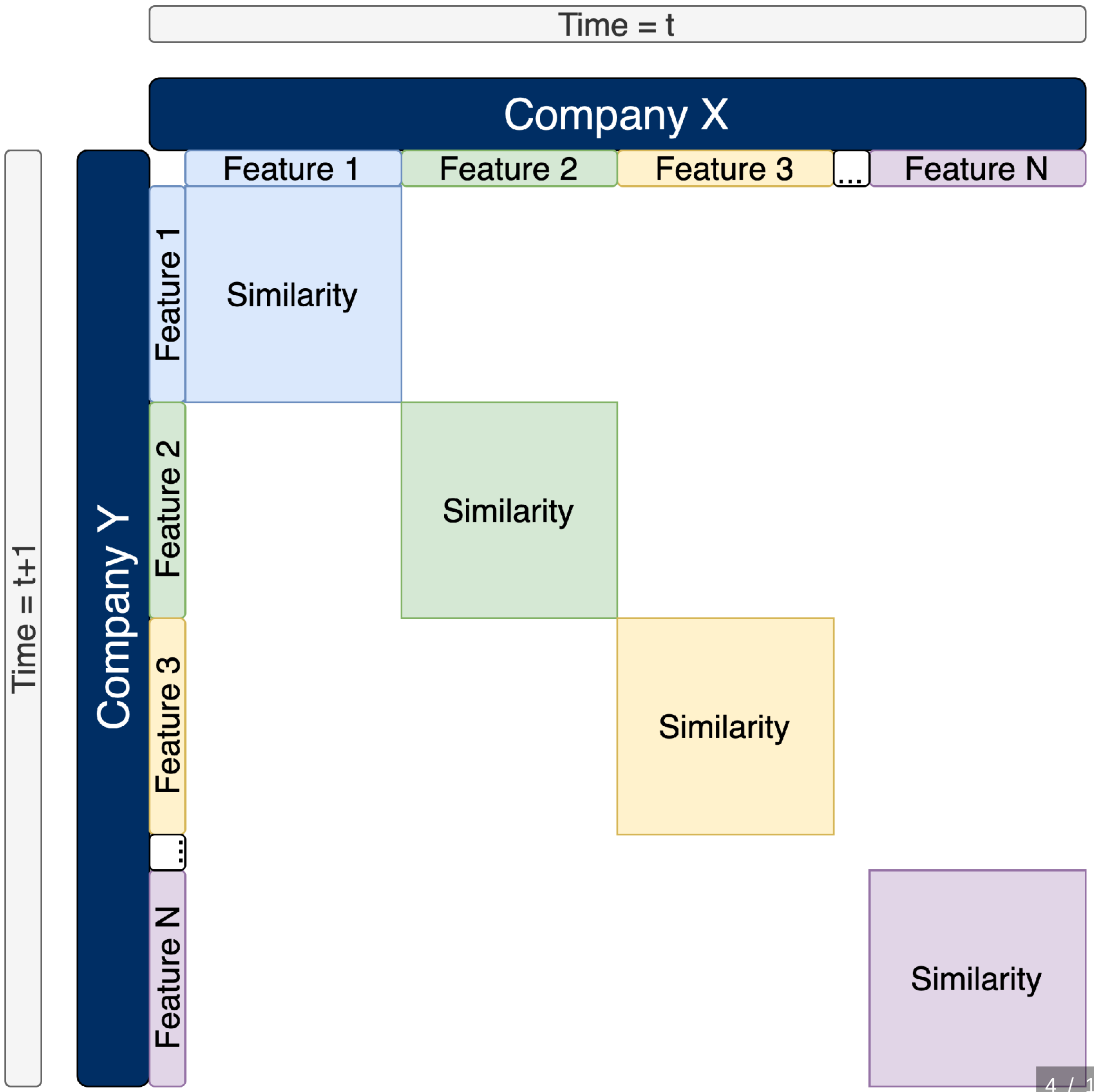
A Matching Problem Example



A Matching Problem Example

Using multiclass classification

- Use the firms of $Right$ as output classes.
- Train a model to classify each f_l from $Left$ to one of the output classes.
- But:
  - Say, $Right$ is the master database.
  - It gets updated and now contains additional firms.
  - The model will not give classification probabilities for the new firms.



Matching as a Binary Classification Problem

- Instead, we can approach the problem from another perspective.
- Make pairs of records for each firm f_l in the $Left$ and f_r in the $Right$ data.
- Classify the pairs (f_l, f_r) as a match $label = 1$ or no match $label = 0$.
- But:
  - This requires cross-joining the $Left$ and $Right$ data.
  - The memory requirements to store the transformed data can quickly render the solution infeasible.
  - Even for small data sources, say $N_L = 5 \cdot 10^3$ and $N_R = 10^4$, the matching pairs $N_{LM} = 5 \cdot 10^7$ require 100s of Gb.

Matching with Blocking

- One solution is to use blocking (Doll, Gabor-Toth, & Schild, 2021).
- Exclude some potential pairs based on pre-defined criteria before training.
- E.g., match $Left$ with firms from $Right$ that have the save foundation year.
- This effectively reduces the memory requirements problem.
- But:
    - The blocking criteria require having already expertise with the data,
    - are not re-usable in different contexts, and
    - might even be different for heterogeneous country data.
- Can we do better?

Matching with a Similarity Encoder

- Encoders are used in natural language processing models (see e.g. Vaswani et al., 2017).
- They reduce text data to vectors used to train models.
- These models use a huge amount of data.
- The encoding is calculated on the fly.
- How can we use this idea?

Matching with a Similarity Encoder

Using a similarity encoder

1. Pick a pair (f_l, f_r) of $Left$ and $Right$ firms.
2. Instruct how the features of f_l and f_r are associated.

```python
similarity_map = {
    "company_name": [
        "discrete",
        "partial",
        my_custom_awesome_similarity
    ],
    "address": [ "partial" ],
    "purpose": [
        "sort",
        lambda x, y: x*y + 42 - y*x
    ],
    "foundation": [
        "discrete",
        "partial",
    ],
}
```

Matching with a Similarity Encoder

Using a similarity encoder

1. Pick a pair (f_l, f_r) of $Left$ and $Right$ firms.
2. Instruct how the features of f_l and f_r are associated.

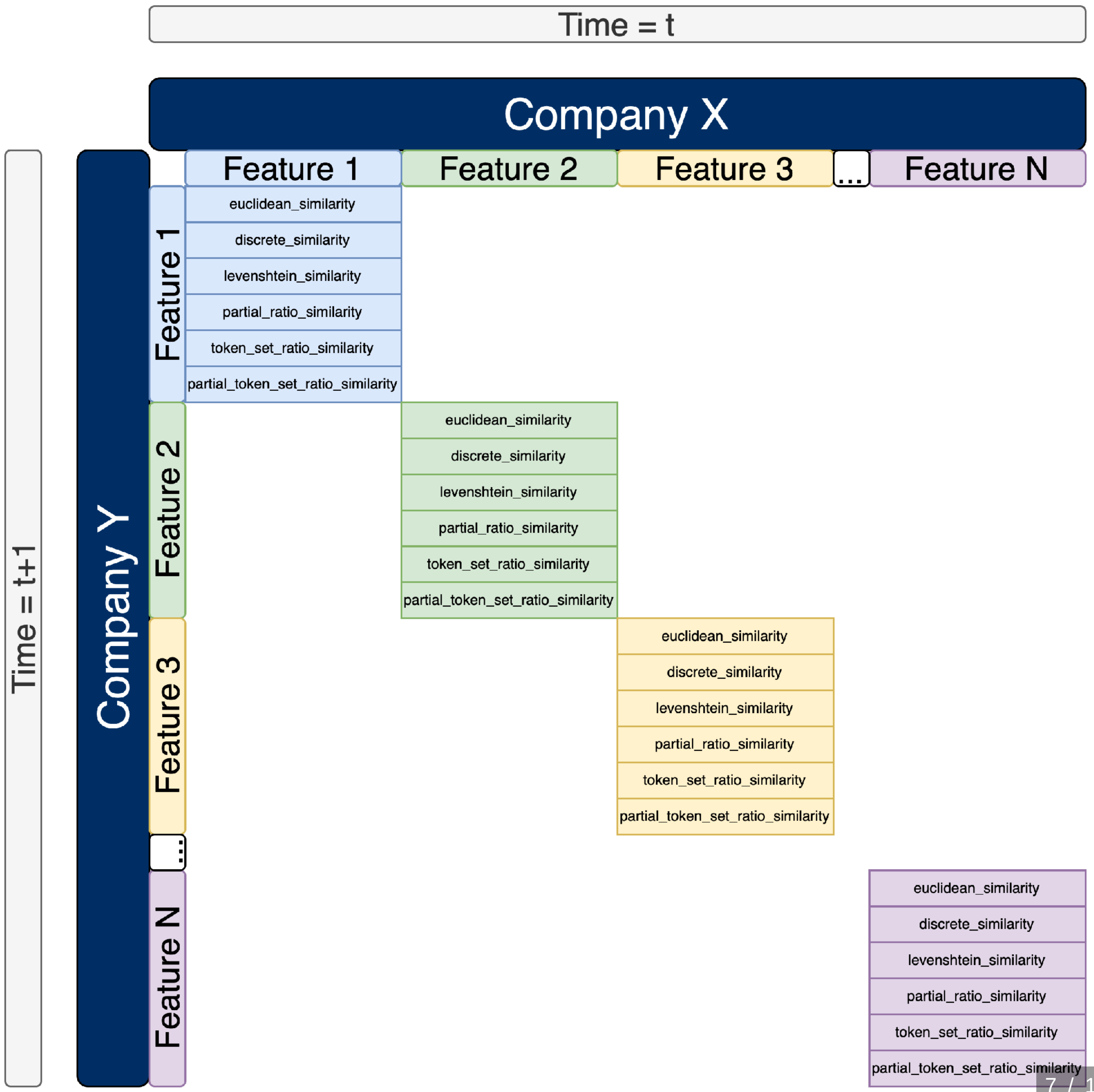
```python
similarity_map = {
    "company_name": [
        "discrete",
        "partial",
        my_custom_awesome_similarity
    ],
    "address": [ "partial" ],
    "purpose": [
        "sort",
        lambda x, y: x*y + 42 - y*x
    ],
    "foundation": [
        "discrete",
        "partial",
    ],
}
```

```python
similarity_map = {
    ...
    "address~address1": [ "partial" ],
    "address~address2": [ "partial" ],
    ...
}
```

Matching with a Similarity Encoder

Using a similarity encoder

1. Pick a pair (f_l, f_r) of $Left$ and $Right$ firms.
2. Instruct how the features of f_l and f_r are associated.
3. Calculate various similarities for each feature on the fly.
   - Train the binary matching model using these similarities.



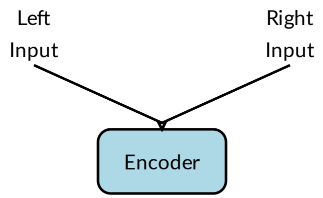
Network Architecture

Network Architecture

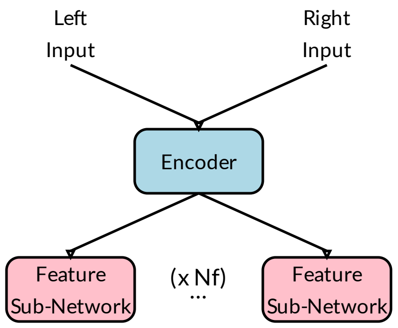
Left
Input

Right
Input

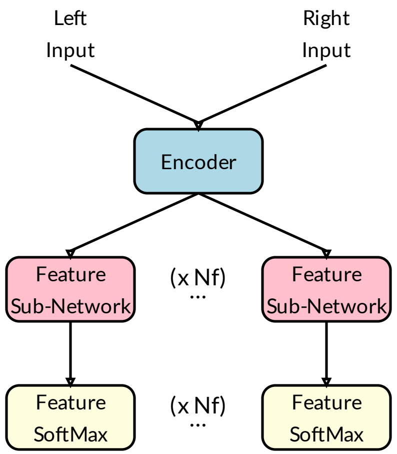
Network Architecture



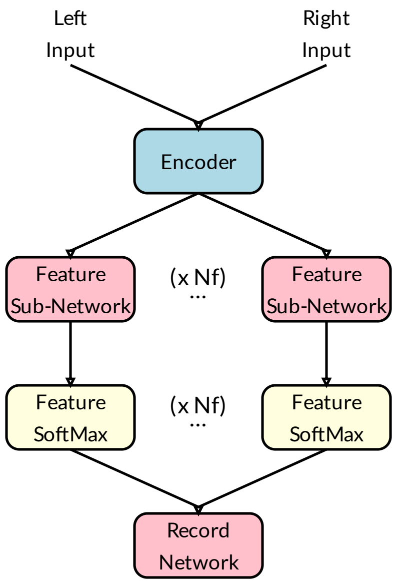
Network Architecture



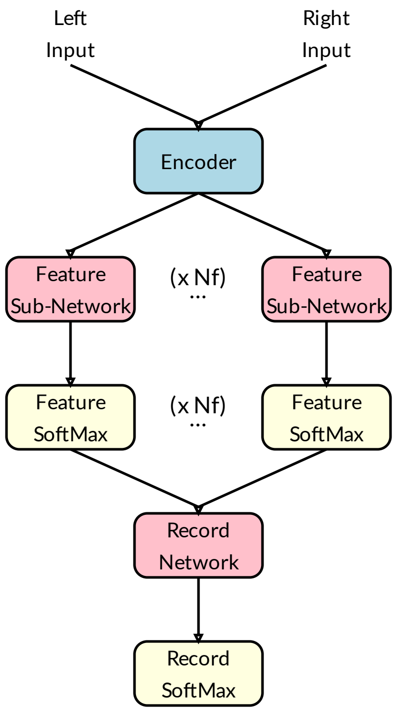
Network Architecture



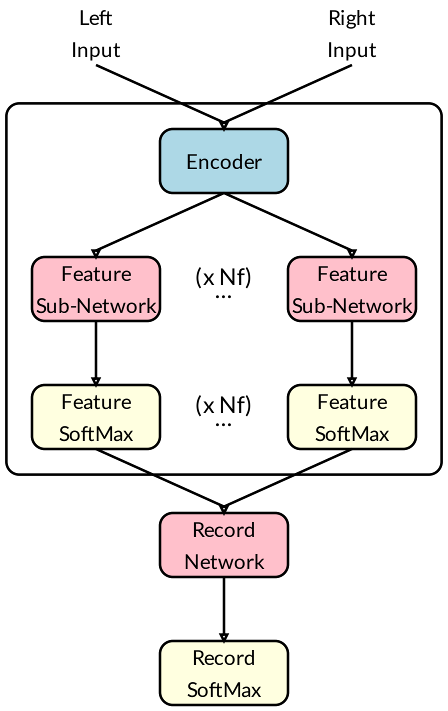
Network Architecture



Network Architecture



Network Architecture



Usage (Pilot)

Usage (Pilot)

```
model = match.MatchingModel(similarity_map)

model.compile(
    loss="binary_crossentropy",
    optimizer=tensorflow.keras.optimizers.Adam(learning_rate=0.01),
    metrics=evaluation_metrics
)

train_left, train_right, train_matches = load_train_data()
model.fit(train_left, train_right, train_matches, epochs=100)

train_evaluation = model.evaluate(train_left, train_right, train_matches)

predictions = model.predict(train_left, train_right)
suggestions = model.suggest(train_left, train_right, 3)
```

Usage (Pilot)

```
model = match.MatchingModel(similarity_map)

model.compile(
    loss="binary_crossentropy",
    optimizer=tensorflow.keras.optimizers.Adam(learning_rate=0.01),
    metrics=evaluation_metrics
)

train_left, train_right, train_matches = load_train_data()
model.fit(train_left, train_right, train_matches, epochs=100)

train_evaluation = model.evaluate(train_left, train_right, train_matches)

predictions = model.predict(train_left, train_right)
suggestions = model.suggest(train_left, train_right, 3)
```

Usage (Pilot)

```
model = match.MatchingModel(similarity_map)

model.compile(
    loss="binary_crossentropy",
    optimizer=tensorflow.keras.optimizers.Adam(learning_rate=0.01),
    metrics=evaluation_metrics
)

train_left, train_right, train_matches = load_train_data()
model.fit(train_left, train_right, train_matches, epochs=100)

train_evaluation = model.evaluate(train_left, train_right, train_matches)

predictions = model.predict(train_left, train_right)
suggestions = model.suggest(train_left, train_right, 3)
```

Usage (Pilot)

```
model = match.MatchingModel(similarity_map)

model.compile(
    loss="binary_crossentropy",
    optimizer=tensorflow.keras.optimizers.Adam(learning_rate=0.01),
    metrics=evaluation_metrics
)

train_left, train_right, train_matches = load_train_data()
model.fit(train_left, train_right, train_matches, epochs=100)

train_evaluation = model.evaluate(train_left, train_right, train_matches)

predictions = model.predict(train_left, train_right)
suggestions = model.suggest(train_left, train_right, 3)
```

Usage (Pilot)

```
model = match.MatchingModel(similarity_map)

model.compile(
    loss="binary_crossentropy",
    optimizer=tensorflow.keras.optimizers.Adam(learning_rate=0.01),
    metrics=evaluation_metrics
)

train_left, train_right, train_matches = load_train_data()
model.fit(train_left, train_right, train_matches, epochs=100)

train_evaluation = model.evaluate(train_left, train_right, train_matches)

predictions = model.predict(train_left, train_right)
suggestions = model.suggest(train_left, train_right, 3)
```

Usage (Pilot)

```
model = match.MatchingModel(similarity_map)

model.compile(
    loss="binary_crossentropy",
    optimizer=tensorflow.keras.optimizers.Adam(learning_rate=0.01),
    metrics=evaluation_metrics
)

train_left, train_right, train_matches = load_train_data()
model.fit(train_left, train_right, train_matches, epochs=100)

train_evaluation = model.evaluate(train_left, train_right, train_matches)

predictions = model.predict(train_left, train_right)
suggestions = model.suggest(train_left, train_right, 3)
```

Machine Learning Record Matching with Similarity Encoding

- Conceptualize record matching via similarity maps.
- Introduce the similarity encoder.
- Express database linking as a binary ML classification problem with similarity encoding.
- Compared to previous approaches:
  - Can match (right) data for which is not trained (unlike the multi-class approach).
  - Has feasible memory requirements than ML classification with similarity pre-processing.
  - Requires minimal expertise with the application data (unlike the blocking approach).
  - It is context-independent and can be used when linking different types of data.

References

Adam, S., Annaert, J., Buelens, F., Coüasnon, B. B., Cule, B., de Vicq, A., Guerry, C., et al. (2021). Data extraction and matching The EurHisFirm experience. *Methodological advances in the extraction and analysis of historical data*, Methodological advances in the extraction and analysis of historical data. Chicago/Virtual, United States: Kellogg School of Management - Northwestern University.

Cule, B., Buelens, F., Poukens, J., Annaert, J., & Richer, J. (2020, December). EurHisFirm M6.2: Data connecting case study. Zenodo.

Doll, H., Gabor-Toth, E., & Schild, C.-J. (2021, May). Linking deutsche bundesbank company data. Deutsche Bundesbank, Research Data and Service Centre.

Karapanagiotis, P. (2019). *EurHisFirm D5.1: Technical document on national data models*. Zenodo.

Poukens, J. (2018). *EurHisFirm D4.2: Report on the inventory of data and sources*. Zenodo.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. u., et al. (2017). Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 30). Curran Associates, Inc.