SOUTENANCE : Recherche

Classification et extraction de données dans les documents administratifs

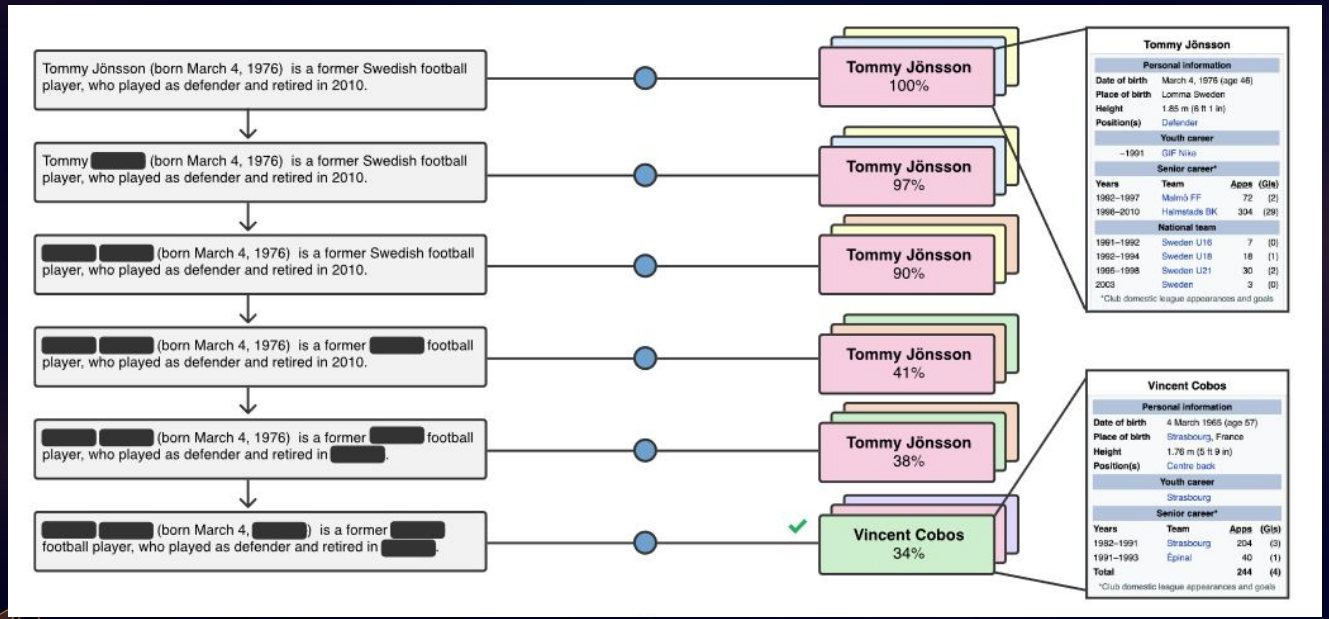
BERGERE Florian
EYNARD Maxime

8 janvier 2024

SOMMAIRE

01 - Contexte

Papier motivant : ***Unsupervised Text Deidentification - John X. Morris Justin T. Chiu Ramin Zabih Alexander M.*** **Rush**



02 - Preprocessing et Datasets

```python
# Preprocessing of a sample
def preprocessing(sample, model, tokenizer, index, max_length=512):
    # Tokenize the target_text using the tokenizer
    target_tokens = tokenizer(sample['target_text'], max_length=max_length, padding='max_length', truncation=True, return_tensors='pt', return_attention_mask=True)

    # Tokenize the input_text using the tokenizer
    input_tokens = tokenizer(sample['input_text'], max_length=max_length, padding='max_length', truncation=True, return_tensors='pt', return_attention_mask=True)

    # Embed target_tokens
    target_embeddings = get_target_embeddings(model, target_tokens["input_ids"], target_tokens["attention_mask"])

    return {
        "target_tokens": target_tokens["input_ids"][0],
        "target_attention_mask": target_tokens["attention_mask"][0],
        "target_embeddings": target_embeddings['target_embeddings'],
        "input_tokens": input_tokens["input_ids"][0],
        "input_attention_mask": input_tokens["attention_mask"][0],
        "label": index
    }

# Preprocessing of a dataset
def preprocessing_dataset(dataset: Dataset, model, tokenizer):
    return dataset.map(lambda sample, index: preprocessing(sample, model, tokenizer, index), with_indices=True, batched=False)
```

02 - Preprocessing et Datasets

```python
# Generation of masked dataset (with/without data augmentation depending of nb_candidates value)
def create_masked_dataset(dataset, model, tokenizer, nb_candidates):
    masked_dataset_dict = {
        'target_tokens': [],
        'target_attention_mask': [],
        'label': []
    }
    for sample in dataset:
        target_tokens = sample['target_tokens']
        special_tokens = [tokenizer.bos_token_id, tokenizer.eos_token_id, tokenizer.pad_token_id, tokenizer.unk_token_id, tokenizer.mask_token_id]
        masked_indices = [i for i, token_id in enumerate(target_tokens) if token_id not in special_tokens]

        for _ in range(nb_candidates):
            l = random.randint(0, len(masked_indices) - 1)
            masked_indices_to_replace = random.sample(masked_indices, l)
            masked_target_tokens = target_tokens.copy()

            for idx in masked_indices_to_replace:
                masked_target_tokens[idx] = tokenizer.mask_token_id

            masked_dataset_dict['target_tokens'].append(masked_target_tokens)
            masked_dataset_dict['target_attention_mask'].append(sample['target_attention_mask'])
            masked_dataset_dict['label'].append(sample['label'])

    masked_dataset = Dataset.from_dict(masked_dataset_dict)
    masked_dataset = masked_dataset.map(lambda sample: get_target_embeddings(model, sample['target_tokens'], sample['target_attention_mask']), batched=False)
    return masked_dataset
```

02 - Preprocessing et Datasets

```
Name: Unmasked
Dataset({
    features: ['target_text', 'input_text', 'target_tokens', 'target_attention_mask', 'target_embeddings', 'input_tokens', 'input_attention_mask', 'label'],
    num_rows: 1000
})
Name: Masked
Dataset({
    features: ['target_tokens', 'target_attention_mask', 'label', 'target_embeddings'],
    num_rows: 2000
})
Name: Augmented Masked
Dataset({
    features: ['target_tokens', 'target_attention_mask', 'label', 'target_embeddings'],
    num_rows: 6000
})
Name: Ner
Dataset({
    features: ['target_text', 'input_text', 'target_tokens', 'target_attention_mask', 'target_embeddings', 'input_tokens', 'input_attention_mask', 'label', 'ner_text',
 'ner_tokens', 'ner_attention_mask', 'ner_embeddings'],
    num_rows: 1000
})
Name: Lexical
Dataset({
    features: ['target_text', 'input_text', 'target_tokens', 'target_attention_mask', 'target_embeddings', 'input_tokens', 'input_attention_mask', 'label', 'lexical_te
xt', 'lexical_tokens', 'lexical_attention_mask', 'lexical_embeddings'],
    num_rows: 1000
})
Name: Idf
Dataset({
    features: ['target_text', 'input_text', 'target_tokens', 'target_attention_mask', 'target_embeddings', 'input_tokens', 'input_attention_mask', 'label', 'idf_text',
 'idf_tokens', 'idf_attention_mask', 'idf_embeddings'],
    num_rows: 1000
})
Name: Idf_table_aware
Dataset({
    features: ['target_text', 'input_text', 'target_tokens', 'target_attention_mask', 'target_embeddings', 'input_tokens', 'input_attention_mask', 'label', 'idf_table_
aware_text', 'idf_table_aware_tokens', 'idf_table_aware_attention_mask', 'idf_table_aware_embeddings'],
    num_rows: 1000
})
```

02 - Preprocessing et Datasets

Unsmasked	Masked	Augmented Masked	Ner	Lexical	IDF	IDF-table-aware
~20min	~25min	~40min	~30min	~35min	~45min	~45min

-> Processus coûteux: avec des moyens sophistiqués (problème de RAM sur nos machines)

03 - Modèles de désidentification

```
def ner(text):
    doc = nlp(text)
    deidentified_text = []

    for token in doc:
        # Masking the text of named entities with a certain probability
        if token.ent_type_:
            deidentified_text.append("<mask>")
        else:
            deidentified_text.append(token.text)

    ner_text = " ".join(deidentified_text)

    return {"ner_text": ner_text}

# LEXICAL
def lexical(text, table_text):
    doc = nlp(text)
    deidentified_text = []
    for token in doc:
        if str(token) in table_text and str(token) not in string.punctuation and str(token) != "\n":
            deidentified_text.append("<mask>")
        else:
            deidentified_text.append(token.text)

    lexical_text = " ".join(deidentified_text)

    return {"lexical_text": lexical_text}
```

03 - Modèles de désidentification

```python
def idf(text, corpus):
    # Tokenization
    tokenized_text = simple_preprocess(text.replace("\n", "").replace("-lrb-", "").replace("-rrb-", ""))
    tokenized_corpus = [simple_preprocess(doc) for doc in corpus]

    # Create a Gensim Dictionary and Corpus and Text
    dct = Dictionary(tokenized_corpus)  # fit dictionary
    corpus_bow = [dct.doc2bow(doc) for doc in tokenized_corpus]
    text_bow = dct.doc2bow(tokenized_text)

    # TF-IDF model
    tfidf_model = TfidfModel(corpus_bow)

    # Applying the TF-IDF model
    tfidf_vector = tfidf_model[text_bow]
    token_tfidf_dict = dict(tfidf_vector)

    # Mask the text based on IDF values
    masked_text = []
    for token in tokenized_text:
        token_index = dct.token2id.get(token, -1)
        tfidf_value = token_tfidf_dict.get(token_index, 0.0)
        if tfidf_value < IDF_THRESHOLD:
            masked_text.append("<mask>")
        else:
            masked_text.append(token)

    return {"idf_text": " ".join(masked_text)}
```

03 - Modèles de désidentification

```python
def idf_table_aware(text, profile, corpus):
    # Tokenization
    tokenized_text = simple_preprocess(text.replace("\n", "").replace("-lrb-", "").replace("-rrb-", ""))
    tokenized_profile = simple_preprocess(profile)
    tokenized_corpus = [simple_preprocess(doc) for doc in corpus]

    # Create a Gensim Dictionary and Corpus
    dct = Dictionary(tokenized_corpus)
    corpus_bow = [dct.doc2bow(doc) for doc in tokenized_corpus]

    # TF-IDF model
    tfidf_model = TfidfModel(corpus_bow)

    # Applying the TF-IDF model
    tfidf_vector = tfidf_model[dct.doc2bow(tokenized_text)]
    token_tfidf_dict = dict(tfidf_vector)

    # Mask the text based on IDF values and overlapping words
    masked_text = []
    for token in tokenized_text:
        token_index = dct.token2id.get(token, -1)
        tfidf_value = token_tfidf_dict.get(token_index, 0.0)
        if token in tokenized_profile or tfidf_value < IDF_TABLE_AWARE_THRESHOLD:
            masked_text.append("<mask>")
        else:
            masked_text.append(token)

    return {"idf_table_aware_text": " ".join(masked_text)}
```

03 - Modèles de désidentification

SANS MASQUE

```
nora reiche -lrb- born 16 september 1983 in leipzig -rrb- is
a german handballer playing for thüringer hc and the german
national team .
she won the champions league with viborg in 2009 .
reiche made her debut on the german team in 2004 .
she received a bronze medal at the 2007 world championship .
```

03 - Modèles de désidentification

NER

<mask> <mask> -lrb- born <mask> <mask> <mask> in leipzig -rrb- is a <mask> handballer playing for thüringer hc and the <mask> national team .
she won the champions league with viborg in <mask> .
reiche made her debut on the <mask> team in <mask> .
she received a bronze medal at the <mask> world championship .

LEXICAL

<mask> <mask> -lrb- born <mask> <mask> <mask> <mask> <mask> -rrb- is <mask> <mask> handballer playing for <mask> <mask> and the <mask> <mask> <mask> .
she won the champions league with <mask> <mask> <mask> .
<mask> made her debut <mask> the <mask> <mask> <mask> <mask> .
she received <mask> bronze medal <mask> the <mask> world championship .

IDF Seuil = 0.1

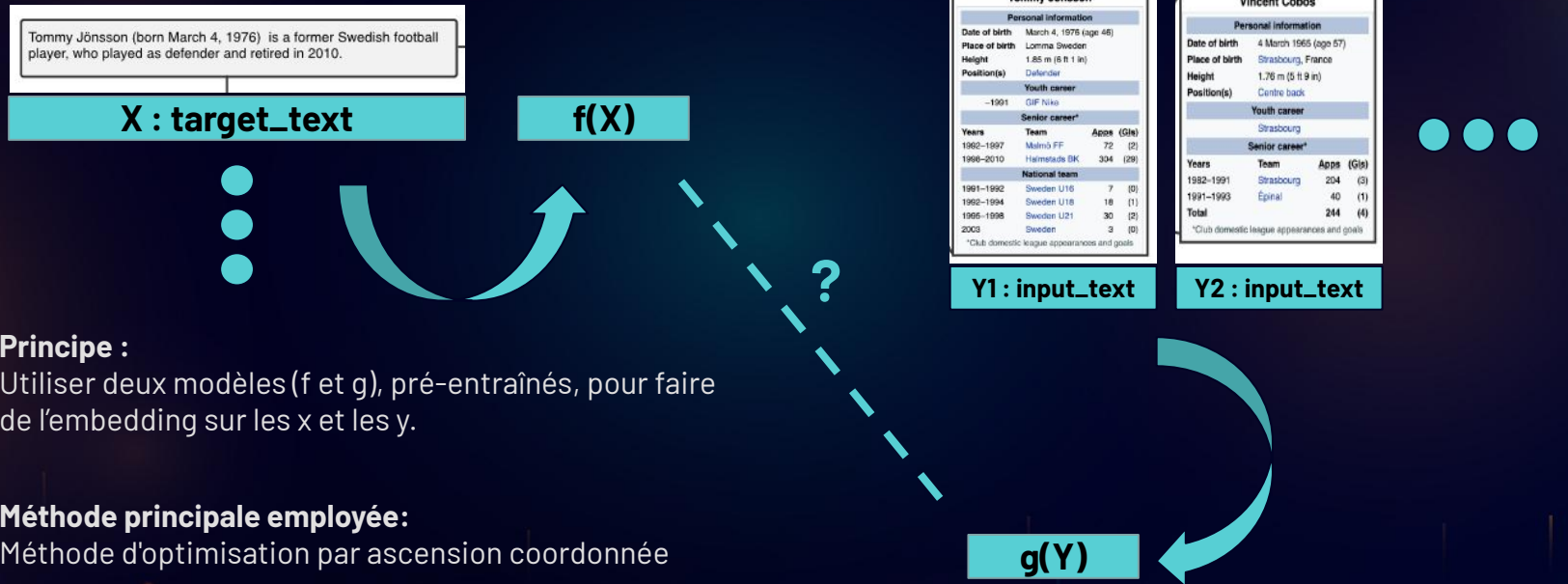
<mask> <mask> -lrb- born 16 september 1983 in <mask> -rrb- is a german <mask> playing for <mask> <mask> and the german national team .
she won the <mask> league with <mask> in 2009 .
<mask> made her debut on the german team in 2004 .
she <mask> a <mask> medal at the 2007 world <mask> .

IDF-TABLE-AWARE Seuil = 0.05

<mask> <mask> -lrb- born <mask> <mask> <mask> <mask> <mask> -rrb- is <mask> <mask> <mask> playing for <mask> <mask> and the <mask> <mask> <mask> .
she won the <mask> league with <mask> <mask> <mask> .
<mask> made her debut <mask> the <mask> <mask> <mask> <mask> .
she <mask> <mask> <mask> medal <mask> the <mask> world <mask> .

04 - Modèles de réidentification - Approche n°1

-> Première approche envisagée selon les informations du papier :



Principe :
Utiliser deux modèles (f et g), pré-entraînés, pour faire de l'embedding sur les x et les y.

Méthode principale employée:
Méthode d'optimisation par ascension coordonnée

04 - Modèles de réidentification - Approche n°1

Méthodologie du code réalisé:

1) **Preprocessing du dataset:**
   - Formatage des données
   - Entrainement d'un tokenizer pré-entraîné sur nos données
     `<old_tokenizer.train_new_from_iterator>`
   - Tokenization des données
     `<dataset.map(lambda sample: preprocessing(sample, trained_tokenizer), batched=False)>`

2) **Réalisation de la méthode d'optimisation par ascension coordonnée**
   - Utilisation de deux modèles pré-entraîné
   - Réalisation d'embeddings à partir de tokens et d'un modèle
   - Mise en place d'un entraînement par ascension coordonnée de sorte à entraîner les modèles et modifier les embeddings de sorte à minimiser une fonction de perte
   - La fonction de perte (limite) `<torch.nn.functional.cosine_similarity>`

```
Requirements :
torch==1.10.0
numpy==1.21.5
datasets==1.16.0
transformers==4.11.2
```

Limites :
- *Evolution très lente des poids*
- *Stagnation de la loss*

Causes Possibles :
- Nécessite un entraînement plus long sur gpu
- Mauvaise fonction loss
- Hyperparamètres

04 - Modèles de réidentification - Approche n°1

```python
# Train the model, minimizing the similarity loss
def train(optimizer, dataset: Dataset):
    losses = []

    input_embeddings_list = []
    for entry in dataset:
        input_embedding = torch.FloatTensor(entry['input_embeddings']).to(device).requires_grad_(True)
        input_embeddings_list.append(input_embedding)

    input_embeddings = torch.stack(input_embeddings_list)

    for positive_index, sample in enumerate(dataset):
        target_embedding = torch.FloatTensor(sample['target_embeddings']).to(device).requires_grad_(True)
        input_embedding = torch.FloatTensor(sample['input_embeddings']).to(device).requires_grad_(True)

        # Calculate similarities
        similarities = calculate_similarities(target_embedding, input_embeddings)
        print_colored(similarities, "red")

        # Similarity-based loss
        loss = multi_loss(similarities, positive_index)
        print_colored(loss, "blue")
        losses.append(loss)

        # Backward pass and optimization
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    total_loss = torch.sum(torch.stack(losses))
    print_colored(total_loss, "green")
```

04 - Modèles de réidentification - Approche n°1

```python
# Calculates the loss -> The less of (neg_similarities - pos_similarity), the best
def multi_loss(similarities, positive_index):
    neg_similarities = torch.cat((similarities[:positive_index], similarities[positive_index + 1:]))
    loss = torch.sum(neg_similarities) - similarities[positive_index]
    return loss

def calculate_similarities(target_embedding, input_embeddings):
    similarities = torch.nn.functional.cosine_similarity(target_embedding.unsqueeze(0), input_embeddings, dim=1)
    return similarities
```

```python
# Calls the train() function, in order to train target and input models each at a time
def coordinate_ascent_optimization(target_model, input_model, target_optimizer, input_optimizer, dataset: Dataset):
    for epoch in range(NB_EPOCHS):
        print_colored(epoch, "red")
        dataset = update_embeddings(target_model, input_model, dataset)
        if epoch % 10 : # Input model
            train(input_optimizer, dataset)
            save_model(model=input_model, location=PRETRAINED_INPUT_MODEL_LOCATION)
            save_model(model=target_model, location=PRETRAINED_TARGET_MODEL_LOCATION)
        else : # Train the target model
            train(target_optimizer, dataset)
# endregion
```

04 - Modèles de réidentification - Approche n°2

Méthodologie du code réalisé:

Préambule :
On souhaite avoir un ou plusieurs modèles qui fonctionne réellement afin d'aboutir à notre objectif : comparer, d'une part, les performances de différents modèles entraînés sur différents pourcentages de masquage sur les X et, d'autre part, leurs robustesses face aux différentes méthodes de désidentification

1) **Preprocessing du dataset:**
   - Formatage des données
   - Entrainement d'un tokenizer pré-entraîné sur nos données
     `<old_tokenizer.train_new_from_iterator>`
   - Tokenization des données

```
<dataset.map(lambda sample: preprocessing(sample, trained_tokenizer), batched=False)>
```

2) **Réalisation d'un réseau de neurones pour la réidentification (à partir d'embeddings pour les X et des labels pour Y)**
   - Utilisation d'un modèle pré-entraîné
   - Réalisation d'embeddings à partir de tokens et du modèle
   - Mise en place de l'entraînement

05 - Comparaison des méthodes de désidentification

-> Input : Embedding du target_text avec le modèle pré-entraîné de roberta-base

Datasets ***Models***	Unmasked **(0%)**	Masked **(~46%)**	Masked (Unmasked) **(~23%)**	Augmented **(~46%)**	Augmented (Unmasked) **(~39%)**	**NER (~20%)**	**LEXICAL (~32%)**	**IDF (~46%)**	**IDF-table (~49%)**
Unmasked	1	0.345	0.5435	0.0992	0.24	**0.073**	**0.02**	**0.017**	**0.007**
Masked	0.18	1	0.18	0.14	0.18	**0.034**	**0.005**	**0.005**	**0.001**
Augmented	0.11	0.05	0.11	0.18	0.18	**0.05**	**0.015**	**0.014**	**0.008**

05 - Comparaison des méthodes de désidentification

-> Input : Target_text tokenizé de taille 512 avec le tokenizer pré-entraîné de roberta-base

Datasets ***Models***	Unmasked **(0%)**	Masked **(~46%)**	Masked (Unmasked) **(~23%)**	Augmented **(~46%)**	Augmented (Unmasked) **(~39%)**	**NER (~20%)**	**LEXICAL (~32%)**	**IDF (~46%)**	**IDF-table (~49%)**
Unmasked	1	0.46	0.72	0.46	0.54	**0.12**	**0.067**	**0.018**	**0.008**
Masked	0.407	0.976	0.29	0.18	0.21	**0.05**	**0.019**	**0.006**	**0.002**
Augmented	0.12	0.09	0.12	0.18	0.18	**0.05**	**0.044**	**0.015**	**0.009**

05 - Comparaison des méthodes de désidentification

-> Nos résultats tendent à confirmer une certaine tendance, que ce soit sur des modèles basés sur les embeddings ou les input_ids des tokens:

- Dans ce type d'exercice, la **data augmentation** est fondamentale mais elle est coûteuse en ressource et en temps.

- Même si les **modèles entraînés sur** le(s) datasets de training **Augmented_Masked** (with/without Unmasked) ne sont pas aboutis et auraient nécessités des améliorations (en augmentant le nombre d' époques, en baissant le learning rate, en augmentant les ressources de calculs et en ayant plus de temps d'entraînement), nous sommes convaincus que la **réidentification** par ce modèle sur la plupart des méthodes de désidentification serait **meilleure que** celle du modèle train sur le dataset **Unmasked**

=> **Apprentissage avec masques** par rapport à notre problème est obligatoire pour réaliser un bon modèle de réidentification !

- L'embedding semble être une moins bonne stratégie que le travail sur les ids des tokens dans cette approche au vu des résultats

-> **NER <= LEXICAL << IDF < IDF-Table-Aware** (vu en égalisant au maximum les % de masquage)

06 - Limites, Ouvertures - Limites

Modèle complexe

Modèle de réidentification difficile à mettre en place,

Bi-encodeurs sur une méthode d'optimisation par ascension coordonnée difficile à maîtriser

Embeddings -> Abstraction parfois "trop abstraite" des textes

Calibrage des hyperparamètres

Problème avec les hyperparamètres du modèle de réidentification que l'on a pas réussi à bien optimiser

Tests effectués :
Jusqu'à 5 couches allant de 100 à 5000 neurones
Lr de 1e-2 à 1e-5

Problème avec le loss

Fonction loss qui ne décroît pas, pas bien définie pour le bi-encodeur

06 - Limites, Ouvertures - Ouvertures

Modèle de paraphrase

"leonard shenoff randle -lrb- born february 12 , 1949 -rrb- is a former major league baseball player . he was the first-round pick of the washington senators in the secondary phase of the june 1970 major league baseball draft , tenth overall ."

"in the june 1970 major league baseball draft, leonard shenoff randle, who was born on february 12, 1949, and lived in rural vermont, was selected as the first-round pick by the washington senators and finished tenth overall."