02_repr_tradicionales

October 27, 2025

[1]:

```
# Generar representaciones tradicionales:
# - TF-IDF de palabras.
# - TF-IDF de n-gramas de caracteres.
# Usaremos solo TRAIN para ajustar y transformaremos TRAIN y VALIDATION.
# Guardamos matrices .npz, vectorizers .pkl y un índice con doc_id/sent_id.
```

Imports y config

[2]:

```
from pathlib import Path
import pandas as pd
import numpy as np
import pickle, json
from scipy import sparse
from sklearn.feature_extraction.text import TfidfVectorizer

pd.set_option("display.max_colwidth", 120)
SEED = 42
```

Rutas

[3]:

```
def find_root():
    p = Path.cwd()
    for cand in [p, *p.parents]:
        if (cand / "data" / "processed").exists():
            return cand
    raise FileNotFoundError("No encuentro data/processed.")

ROOT = find_root()
PROC = ROOT / "data" / "processed"
FEAT = ROOT / "features" / "tfidf"
FEAT.mkdir(parents=True, exist_ok=True)

NIVELES = ["easy","medium","hard"]
SPLITS = ["train","validation"]
```

Carga de processed → DataFrame

[4]:

```
def cargar_processed():
    filas = []
    for level in NIVELES:
        for split in SPLITS:
            p = PROC / level / split / "sentences.jsonl"
            if not p.exists():
                continue
            df = pd.read_json(p, lines=True)
            df["level"] = level
            df["split"] = split
            filas.append(df[["doc_id","sent_id","level","split","text_norm"]])
    return pd.concat(filas, ignore_index=True)

df = cargar_processed().sort_values(["level","split","doc_id","sent_id"]).
 ↪reset_index(drop=True)
print(df.shape, "filas")
df.head(3)
```

```
(208160, 5) filas
```

[4]:

```
      doc_id  sent_id level  split  \
0  problem-1        0  easy  train
1  problem-1        1  easy  train
2  problem-1        2  easy  train

                                 text_norm
0  there s also incidents of testosterone insensitive males that have either
ambiguous genitals or are phenotypically f…
1                    female is the human default body plan so a number of
conditions exist that cause female looking males
2                             what about people born appearing num female
complete with the bits but are genetically males
```

TF-IDF de palabras

[5]:

```
# Ajuste en TRAIN
df_train = df[df["split"] == "train"]
corpus_train = df_train["text_norm"].astype(str).tolist()

vec_word = TfidfVectorizer(
    lowercase=False,          # ya normalizado
    analyzer="word",
    ngram_range=(1,2),        # unigrams + bigrams
    min_df=5,                 # filtra ruido
    max_features=100_000
)
Xtr_w = vec_word.fit_transform(corpus_train)

# Transform VALIDATION
df_val = df[df["split"] == "validation"]
Xva_w = vec_word.transform(df_val["text_norm"].astype(str).tolist())
```

```

print("WORD TF-IDF")
print("train:", Xtr_w.shape, "nnz:", Xtr_w.nnz)
print("val  :", Xva_w.shape, "nnz:", Xva_w.nnz)
```

```
WORD TF-IDF
train: (171602, 85932) nnz: 4449125
val  : (36558, 85932) nnz: 937263
```

TF-IDF de n-gramas de caracteres

[6]:
```
vec_char = TfidfVectorizer(
    analyzer="char",
    ngram_range=(3,5),
    min_df=5,
    max_features=200_000
)
Xtr_c = vec_char.fit_transform(corpus_train)
Xva_c = vec_char.transform(df_val["text_norm"].astype(str).tolist())

print("CHAR TF-IDF")
print("train:", Xtr_c.shape, "nnz:", Xtr_c.nnz)
print("val  :", Xva_c.shape, "nnz:", Xva_c.nnz)
```

```
CHAR TF-IDF
train: (171602, 134419) nnz: 45449143
val  : (36558, 134419) nnz: 9754529
```

Guardado de matrices y metadatos

[7]:
```
# Índices para mapear filas → (level, split, doc_id, sent_id)
idx_train = df_train[["level","split","doc_id","sent_id"]].
 ↪reset_index(drop=True)
idx_val   = df_val[["level","split","doc_id","sent_id"]].reset_index(drop=True)

# Carpetas
( FEAT / "word" ).mkdir(parents=True, exist_ok=True)
( FEAT / "char" ).mkdir(parents=True, exist_ok=True)

# WORD
sparse.save_npz(FEAT / "word" / "X_train_word.npz", Xtr_w)
sparse.save_npz(FEAT / "word" / "X_val_word.npz",   Xva_w)
pickle.dump(vec_word, open(FEAT / "word" / "vectorizer_word.pkl","wb"))
idx_train.to_csv(FEAT / "word" / "index_train.csv", index=False)
idx_val.to_csv(  FEAT / "word" / "index_val.csv",   index=False)

# CHAR
sparse.save_npz(FEAT / "char" / "X_train_char.npz", Xtr_c)
sparse.save_npz(FEAT / "char" / "X_val_char.npz",   Xva_c)
```

```
pickle.dump(vec_char, open(FEAT / "char" / "vectorizer_char.pkl","wb"))
idx_train.to_csv(FEAT / "char" / "index_train.csv", index=False)
idx_val.to_csv(  FEAT / "char" / "index_val.csv",   index=False)

# Informe JSON mínimo
reporte = {
    "word": {
        "shape_train": Xtr_w.shape, "nnz_train": int(Xtr_w.nnz),
        "shape_val":   Xva_w.shape, "nnz_val":   int(Xva_w.nnz),
        "vocab_size":  len(vec_word.vocabulary_)
    },
    "char": {
        "shape_train": Xtr_c.shape, "nnz_train": int(Xtr_c.nnz),
        "shape_val":   Xva_c.shape, "nnz_val":   int(Xva_c.nnz),
        "vocab_size":  len(vec_char.vocabulary_)
    }
}
(Path(FEAT) / "tfidf_resumen.json").write_text(json.dumps(reporte, indent=2), encoding="utf-8")
print("Guardado en features/tfidf/")
```

```
Guardado en features/tfidf/
```

Tests FInales

[8]:
```
# Terminos más pesados por TF-IDF medio en TRAIN (palabras)
col_means = np.asarray(Xtr_w.mean(axis=0)).ravel()
top_idx = col_means.argsort()[-10:][::-1]
inv_vocab = {j:i for i,j in vec_word.vocabulary_.items()}
top_terms = [inv_vocab[i] for i in top_idx]
pd.DataFrame({"term": top_terms, "tfidf_mean": col_means[top_idx].round(6)})
```

[8]:
```
    term  tfidf_mean
0    the    0.034300
1     to    0.026840
2    and    0.022177
3     of    0.020698
4     in    0.018026
5     it    0.017368
6     is    0.017342
7   that    0.016891
8   they    0.013748
9    you    0.013222
```

1 Informe breve — `02_representaciones_tradicionales.ipynb`

1.1 Objetivo

Crear representaciones TF-IDF sobre frases normalizadas en `processed` para alimentar baselines y modelos posteriores.

1.2 Configuración

- Ajuste en **train**. Transformación en **train** y **validation**.
- Dos variantes:
  - Palabras: `ngram_range=(1,2)`, `min_df=5`, `max_features=100k`, `lowercase=False`.
  - Caracteres: `ngram_range=(3,5)`, `min_df=5`, `max_features=200k`.

1.3 Salidas

- `features/tfidf/word/`
  - `X_train_word.npz`, `X_val_word.npz`
  - `vectorizer_word.pkl`
  - `index_train.csv`, `index_val.csv`
- `features/tfidf/char/`
  - `X_train_char.npz`, `X_val_char.npz`
  - `vectorizer_char.pkl`
  - `index_train.csv`, `index_val.csv`
- Resumen: `features/tfidf/tfidf_resumen.json` con shapes, nnz y vocabulario.

1.4 Uso previsto

- Comparar baselines de cambio de autor calculando distancias coseno entre frases o ventanas.
- Servir de entrada a clasificadores ligeros en E3/E4.

1.5 Notas

- El texto ya viene normalizado desde `01_preprocesamiento`.
- Los índices CSV permiten mapear cada fila a `(level, split, doc_id, sent_id)`.
- Los hiperparámetros están pensados para un primer pase. Ajustables según memoria y rendimiento.