

1. Setup and Imports

Import necessary libraries and project modules. Set up paths to access `src` code.

```
import os, json
from pathlib import Path
import pandas as pd

# Make src importable
import sys
sys.path.append(str(Path("..").resolve())) # if notebook inside
/notebooks
sys.path.append(str((Path(..) / "src").resolve()))

from src.common.io import load_docs, load_rules, load_gold
from src.baselines.tfidf_ir import tfidf_ir_predict
from src.baselines.logistic_baseline import logistic_predict
from src.baselines.fasttext_05a import fasttext_predict_05A
from src.eval.metrics import evaluate
```

2. Load Data

Load documents, compliance rules, and gold standard labels from the `data` directory.

```
DOCS_DIR = "../data/docs"
RULES_PATH = "../rules/rules.json"
GOLD_PATH = "../data/gold_labels.json"

docs = load_docs(DOCS_DIR)
rules = load_rules(RULES_PATH)
gold = load_gold(GOLD_PATH)

print("Loaded docs:", [d.doc_id for d in docs])
print("Loaded rules:", [r.id for r in rules])

Loaded docs: ['doc_001', 'doc_002', 'doc_003', 'DOC_004', 'DOC_005',
'DOC_006', 'DOC_007', 'DOC_008', 'DOC_009', 'DOC_010', 'DOC_011',
'DOC_012', 'DOC_013', 'DOC_014', 'DOC_015']
Loaded rules: ['01', '02', '03', '05A', '08', '10A', '12', '13']
```

3. Baseline: TF-IDF Information Retrieval

Run a simple vector space model baseline. It computes Cosine Similarity between the document text and the rule description. If similarity < threshold, it predicts VIOLATED.

```
preds_tfidf = tfidf_ir_predict(
    docs=docs,
```

```

    rules=rules,
    model_name="TFIDF-IR",
    threshold=0.10 # you can tune this
)

eval_tfidf = evaluate(preds_tfidf, gold)
eval_tfidf["overall"], list(eval_tfidf["per_rule"].items())[:2]

({'macro_precision': 0.2395833333333334,
 'macro_recall': 0.75,
 'macro_f1': 0.3374295467174724},
 [(
  '01',
  {'tp': 3,
   'fp': 9,
   'fn': 0,
   'precision': 0.25,
   'recall': 1.0,
   'f1': 0.4,
   'coverage': 1.0}),
 (
  '02',
  {'tp': 0,
   'fp': 12,
   'fn': 0,
   'precision': 0.0,
   'recall': 0.0,
   'f1': 0.0,
   'coverage': 1.0)])

```

4. Baseline: Logistic Regression

Train a Logistic Regression classifier on TF-IDF features. Note: This is purely illustrative due to the very small dataset size.

```

preds_logreg = logistic_predict(
    docs=docs,
    rules=rules,
    gold=gold,
    model_name="TFIDF+LogReg"
)

eval_logreg = evaluate(preds_logreg, gold)
eval_logreg["overall"], list(eval_logreg["per_rule"].items())[:2]

({'macro_precision': 0.25, 'macro_recall': 0.25, 'macro_f1': 0.25},
 [(
  '01',
  {'tp': 0,
   'fp': 0,
   'fn': 3,
   'precision': 0.0,
   'recall': 0.0,
   'f1': 0.0,
   'coverage': 1.0}),
 (
  '02',
  {'tp': 0,
   'fp': 12,
   'fn': 0,
   'precision': 0.0,
   'recall': 0.0,
   'f1': 0.0,
   'coverage': 1.0)])

```

```

'recall': 0.0,
'f1': 0.0,
'coverage': 1.0}),
('02',
{'tp': 0,
'fp': 0,
'fn': 0,
'precision': 0.0,
'recall': 0.0,
'f1': 0.0,
'coverage': 1.0}])

```

5. Compare Results

Aggregate evaluation metrics (Precision, Recall, F1) for both baselines into a Pandas DataFrame for easy comparison.

```

def eval_to_df(eval_obj, model_name):
    rows = []
    for rid, m in eval_obj["per_rule"].items():
        rows.append({
            "model": model_name,
            "rule": rid,
            "precision": m["precision"],
            "recall": m["recall"],
            "f1": m["f1"],
            "coverage": m["coverage"]
        })
    return pd.DataFrame(rows)

df = pd.concat([
    eval_to_df(eval_tfidf, "TFIDF-IR"),
    eval_to_df(eval_logreg, "TFIDF+LogReg"),
], ignore_index=True)

df.sort_values(["rule", "model"])

      model rule  precision  recall      f1  coverage
8  TFIDF+LogReg  01  0.000000   0.0  0.000000  1.0
0      TFIDF-IR  01  0.250000   1.0  0.400000  1.0
9  TFIDF+LogReg  02  0.000000   0.0  0.000000  1.0
1      TFIDF-IR  02  0.000000   0.0  0.000000  1.0
10  TFIDF+LogReg  03  0.000000   0.0  0.000000  1.0
2      TFIDF-IR  03  0.083333   1.0  0.153846  1.0
11  TFIDF+LogReg  05A 0.000000   0.0  0.000000  1.0
3      TFIDF-IR  05A  0.416667   1.0  0.588235  1.0
12  TFIDF+LogReg  08  0.000000   0.0  0.000000  1.0
4      TFIDF-IR  08  0.083333   1.0  0.153846  1.0
13  TFIDF+LogReg  10A 1.000000   1.0  1.000000  1.0

```

5	TFIDF-IR	10A	0.583333	1.0	0.736842	1.0
14	TFIDF+LogReg	12	1.000000	1.0	1.000000	1.0
6	TFIDF-IR	12	0.500000	1.0	0.666667	1.0
15	TFIDF+LogReg	13	0.000000	0.0	0.000000	1.0
7	TFIDF-IR	13	0.000000	0.0	0.000000	1.0

6. Save Results

Save the summary table to a CSV file in the `results` directory.

```
OUT_DIR = Path("../results")
OUT_DIR.mkdir(parents=True, exist_ok=True)

df.to_csv(OUT_DIR / "summary.csv", index=False)
print("Saved:", OUT_DIR / "summary.csv")

Saved: ..\results\summary.csv
```

7. Baseline: FastText Embeddings (Rule 05A)

Run a semantic similarity baseline specifically for Rule 05A using pre-trained Italian FastText embeddings. This approach matches keywords like 'resistente' or 'refrattario', but may fail on negations.

```
from pathlib import Path

p = Path("../models/cc.it.300.bin")
print("exists:", p.exists())
print("absolute:", p.resolve())
print("suffix:", p.suffix)
print("size MB:", round(p.stat().st_size / (1024*1024), 2) if
p.exists() else None)

exists: True
absolute: C:\Users\loren\Desktop\Uni\Statale\NLP\Exam_LLM\models\
cc.it.300.bin
suffix: .bin
size MB: 6903.07

ft_preds = fasttext_predict_05A(
    docs=docs,
    model_path="../models/cc.it.300.bin",
    threshold=0.60
)

eval_ft = evaluate(ft_preds, gold)
eval_ft["overall"], eval_ft["per_rule"].get("05A")

Loading vectors from ../models/cc.it.300.bin...
```

```
({'macro_precision': 0.75,
 'macro_recall': 0.6,
 'macro_f1': 0.6666666666666665},
 {'tp': 3,
 'fp': 1,
 'fn': 2,
 'precision': 0.75,
 'recall': 0.6,
 'f1': 0.6666666666666665,
 'coverage': 1.0})
```

8. FastText Results

View the performance of the FastText baseline on Rule 05A.

```
df_ft = eval_to_df(eval_ft, "fastText-sim-05A")
df_ft
```

	model	rule	precision	recall	f1	coverage
0	fastText-sim-05A	05A	0.75	0.6	0.666667	1.0