2_models

December 27, 2025

1 Models training

[5]:
```
# --- Set up environment ---
import pandas as pd
import spacy
import numpy as np
import re

from collections import Counter
from collections import defaultdict
from tqdm import tqdm
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk.tokenize import sent_tokenize
from transformers import AutoTokenizer, AutoModelForSequenceClassification
from torch.nn.functional import softmax
from torch import argmax
from pathlib import Path
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from scipy import sparse

tqdm.pandas()
```

[2]:
```
!python -m spacy download en_core_web_sm
nlp = spacy.load("en_core_web_sm")
import features
```

```
Defaulting to user installation because normal site-packages is not writeable
Collecting en-core-web-sm==3.8.0
  Downloading https://github.com/explosion/spacy-
models/releases/download/en_core_web_sm-3.8.0/en_core_web_sm-3.8.0-py3-none-
any.whl (12.8 MB)
     ---------------------------------------- 0.0/12.8 MB ? eta -:--:--
     ---- ----------------------------------- 1.6/12.8 MB 39.8 MB/s eta 0:00:01
     ---------------------------------------- 12.8/12.8 MB 54.4 MB/s  0:00:00
 Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
```

[3]:
```
training_data_pan = "C://Users//andre//OneDrive//Desktop//PAN Task 2025//train//
 ↪train.jsonl"
testing_data_pan = "C://Users//andre//OneDrive//Desktop//PAN Task 2025//train//
 ↪val.jsonl"
training_data_migration = "C://Users//andre//OneDrive//Desktop//PAN Task 2025//
 ↪SOCC_data//migration_texts_both_train.xlsx"
testing_data_migration = "C://Users//andre//OneDrive//Desktop//PAN Task 2025//
 ↪SOCC_data//migration_texts_both_test.xlsx"
```

2 Baseline model on PAN dataset

[8]:
```
# Keep original and metadata
train_path = Path(training_data_pan)
val_path = Path(testing_data_pan)

df_train = pd.read_json(train_path, lines=True)
df_train = df_train[["id", "text", "label", "genre", "model"]].copy()
df_train["label"] = df_train["label"].astype(int)

# --- Train full baseline model ---

# Apply spaCy to get all tokens, POS tags, etc.
df_train["spacy_doc"] = df_train["text"].progress_apply(nlp)

# Baseline features for TRAIN
baseline_df = pd.concat([
    df_train["spacy_doc"].progress_apply(features.extract_length_features),
    df_train["spacy_doc"].progress_apply(features.extract_stopword_rate),
    df_train["text"].progress_apply(features.extract_punct_rates),
    df_train["text"].progress_apply(features.extract_char_class_ratios),
    df_train["spacy_doc"].progress_apply(features.extract_ttr),
    df_train["spacy_doc"].progress_apply(features.extract_upos_freq),
    df_train["spacy_doc"].progress_apply(features.compute_5gram_repetition),
    df_train["spacy_doc"].progress_apply(features.compute_self_similarity)
], axis=1)

# Scale features
scaler = StandardScaler()
X_handcrafted_scaled = scaler.fit_transform(baseline_df)
X_handcrafted_sparse = sparse.csr_matrix(X_handcrafted_scaled)

# Train classifier
clf = LogisticRegression(max_iter=1000, verbose=1)
clf.fit(X_handcrafted_sparse, df_train["label"])

# --- Validate ---
```

```
df_val = pd.read_json(val_path, lines=True)
df_val["spacy_doc"] = df_val["text"].progress_apply(nlp)

# baseline features for VAL
baseline_val = pd.concat([
    df_val["spacy_doc"].progress_apply(features.extract_length_features),
    df_val["spacy_doc"].progress_apply(features.extract_stopword_rate),
    df_val["text"].progress_apply(features.extract_punct_rates),
    df_val["text"].progress_apply(features.extract_char_class_ratios),
    df_val["spacy_doc"].progress_apply(features.extract_ttr),
    df_val["spacy_doc"].progress_apply(features.extract_upos_freq),
    df_val["spacy_doc"].progress_apply(features.compute_5gram_repetition),
    df_val["spacy_doc"].progress_apply(features.compute_self_similarity)
], axis=1)

# Scale features with SAME scaler
X_handcrafted_scaled_val = scaler.transform(baseline_val)
X_handcrafted_sparse_val = sparse.csr_matrix(X_handcrafted_scaled_val)

# Predict
y_val_pred = clf.predict(X_handcrafted_sparse_val)

df_val = df_val.copy()
df_val["y_true"] = df_val["label"]
df_val["y_pred"] = y_val_pred

# Evaluate
report_df = pd.DataFrame(
    classification_report(
        df_val["label"], y_val_pred,
        target_names=["Human", "AI"],
        output_dict=True)
).transpose()

# Sanity check
print(baseline_df.isna().sum().sum())
print(baseline_val.isna().sum().sum())
print(baseline_df.shape, baseline_val.shape)
print(confusion_matrix(df_val["label"], y_val_pred))
```

```
100%|      | 23707/23707 [32:48<00:00, 12.05it/s]
100%|      | 23707/23707 [00:32<00:00, 732.24it/s]
100%|      | 23707/23707 [00:08<00:00, 2944.71it/s]
100%|      | 23707/23707 [00:05<00:00, 4585.32it/s]
100%|      | 23707/23707 [00:19<00:00, 1222.54it/s]
100%|      | 23707/23707 [00:13<00:00, 1798.40it/s]
100%|      | 23707/23707 [00:10<00:00, 2325.50it/s]
```

```
100%|      | 23707/23707 [00:24<00:00, 964.43it/s]
100%|      | 23707/23707 [00:17<00:00, 1371.26it/s]
100%|      | 3589/3589 [04:49<00:00, 12.40it/s]
100%|      | 3589/3589 [00:02<00:00, 1235.43it/s]
100%|      | 3589/3589 [00:01<00:00, 2629.44it/s]
100%|      | 3589/3589 [00:00<00:00, 6417.69it/s]
100%|      | 3589/3589 [00:02<00:00, 1289.63it/s]
100%|      | 3589/3589 [00:01<00:00, 2008.22it/s]
100%|      | 3589/3589 [00:01<00:00, 2160.12it/s]
100%|      | 3589/3589 [00:03<00:00, 1060.48it/s]
100%|      | 3589/3589 [00:02<00:00, 1481.65it/s]

0
0
(23707, 38) (3589, 38)
[[1169  108]
 [  64 2248]]
```

[9]:
```python
# --- Analysis ---
# Full report
full_results_output_path = Path("1_baseline_model_pan_results.xlsx")
report_df.to_excel(full_results_output_path, index=True)

print(f"\nSaved baseline model report to: {full_results_output_path}")
print(report_df)

# Misclassifications
misclassified = df_val[df_val["y_true"] != df_val["y_pred"]]
misclassified_feats = baseline_val.loc[misclassified.index]

misclassified_full = pd.concat(
    [misclassified[["id", "text", "y_true", "y_pred", "model", "genre"]],
     misclassified_feats],
    axis=1
)

missclassifications_output_path = Path("baseline_pan_misclassifications.xlsx")
misclassified_full.to_excel(missclassifications_output_path, index=False)

print(f"Saved {len(misclassified_full)} misclassifications to␣
 ↪{missclassifications_output_path}")

# Check what the baseline classifier learned
handcrafted_df = baseline_df
handcrafted_feat_names = list(handcrafted_df.columns)
coefs = clf.coef_[0]   # shape = (n_features,)
```

```
coef_df = pd.DataFrame({
    "feature": handcrafted_feat_names,
    "weight": coefs
})

top_ai = coef_df.sort_values("weight", ascending=False).head(20)
top_human = coef_df.sort_values("weight", ascending=True).head(20)

print("\nTop features pushing toward AI:")
print(top_ai)

print("\nTop features pushing toward Human:")
print(top_human)
```

```
Saved baseline model report to: 1_baseline_model_pan_results.xlsx
              precision    recall  f1-score      support
Human          0.948094  0.915427  0.931474  1277.000000
AI             0.954160  0.972318  0.963153  2312.000000
accuracy       0.952076  0.952076  0.952076     0.952076
macro avg      0.951127  0.943873  0.947314  3589.000000
weighted avg   0.952001  0.952076  0.951882  3589.000000
Saved 172 misclassifications to baseline_pan_misclassifications.xlsx

Top features pushing toward AI:
             feature    weight
26         upos_PRON  2.843443
5            punct_,  1.470360
21               ttr  1.406904
28         upos_DET  1.124349
2      mean_word_len  1.086647
36   rep_5gram_ratio  0.904316
37  self_sim_jaccard  0.888103
0      mean_sent_len  0.652200
19       digit_ratio  0.556874
11           punct_'  0.435686
34         upos_PART  0.391804
3       std_word_len  0.355190
9            punct_?  0.114997
17       upper_ratio  0.057353
30        upos_SCONJ  0.033016
12           punct_"  0.023750
6            punct_. -0.013307
16           punct_… -0.022215
8            punct_: -0.022722
13           punct_( -0.047620
```

```
Top features pushing toward Human:
           feature    weight
4   stopword_ratio -3.621120
7          punct_; -1.504417
35      upos_PROPN -1.285680
15         punct_- -1.240677
1     std_sent_len -1.150756
20     space_ratio -1.017590
31        upos_NUM -0.902325
24        upos_ADJ -0.695302
23       upos_VERB -0.620994
25        upos_ADV -0.478760
33       upos_INTJ -0.463896
32        upos_AUX -0.370075
22       upos_NOUN -0.338835
10         punct_! -0.248612
14         punct_) -0.110865
29      upos_CCONJ -0.094989
27        upos_ADP -0.085844
18     title_ratio -0.062135
13         punct_( -0.047620
8          punct_: -0.022722
```

3 Baseline model on migration dataset

[10]:
```python
# Keep original and metadata
train_path = Path(training_data_migration)
val_path = Path(testing_data_migration)

df_train = pd.read_excel(train_path)
df_train = df_train[["id", "text", "label", "genre", "model"]].copy()
df_train["label"] = df_train["label"].astype(int)

# --- Train full baseline model ---

# Apply spaCy to get all tokens, POS tags, etc.
df_train["spacy_doc"] = df_train["text"].progress_apply(nlp)

# Baseline features for TRAIN
baseline_df = pd.concat([
    df_train["spacy_doc"].progress_apply(features.extract_length_features),
    df_train["spacy_doc"].progress_apply(features.extract_stopword_rate),
    df_train["text"].progress_apply(features.extract_punct_rates),
    df_train["text"].progress_apply(features.extract_char_class_ratios),
    df_train["spacy_doc"].progress_apply(features.extract_ttr),
    df_train["spacy_doc"].progress_apply(features.extract_upos_freq),
    df_train["spacy_doc"].progress_apply(features.compute_5gram_repetition),
```

```
    df_train["spacy_doc"].progress_apply(features.compute_self_similarity)
], axis=1)

# Scale features
scaler = StandardScaler()
X_handcrafted_scaled = scaler.fit_transform(baseline_df)
X_handcrafted_sparse = sparse.csr_matrix(X_handcrafted_scaled)

# Train classifier
clf = LogisticRegression(max_iter=1000, verbose=1)
clf.fit(X_handcrafted_sparse, df_train["label"])

# --- Validate ---
df_val = pd.read_excel(val_path)

df_val["spacy_doc"] = df_val["text"].progress_apply(nlp)

# baseline features for VAL
baseline_val = pd.concat([
    df_val["spacy_doc"].progress_apply(features.extract_length_features),
    df_val["spacy_doc"].progress_apply(features.extract_stopword_rate),
    df_val["text"].progress_apply(features.extract_punct_rates),
    df_val["text"].progress_apply(features.extract_char_class_ratios),
    df_val["spacy_doc"].progress_apply(features.extract_ttr),
    df_val["spacy_doc"].progress_apply(features.extract_upos_freq),
    df_val["spacy_doc"].progress_apply(features.compute_5gram_repetition),
    df_val["spacy_doc"].progress_apply(features.compute_self_similarity)
], axis=1)

# Scale features with SAME scaler
X_handcrafted_scaled_val = scaler.transform(baseline_val)
X_handcrafted_sparse_val = sparse.csr_matrix(X_handcrafted_scaled_val)

# Predict
y_val_pred = clf.predict(X_handcrafted_sparse_val)

# Evaluate
report_df = pd.DataFrame(
    classification_report(
        df_val["label"], y_val_pred,
        target_names=["Human", "AI"],
        output_dict=True)
).transpose()

# Sanity check results
print(baseline_df.isna().sum().sum())
print(baseline_val.isna().sum().sum())
```

```
print(baseline_df.shape, baseline_val.shape)

print(confusion_matrix(df_val["label"], y_val_pred))
```

```
100%|      | 522/522 [00:47<00:00, 11.09it/s]
100%|      | 522/522 [00:00<00:00, 712.02it/s]
100%|      | 522/522 [00:00<00:00, 1994.16it/s]
100%|      | 522/522 [00:00<00:00, 4790.03it/s]
100%|      | 522/522 [00:00<00:00, 955.76it/s]
100%|      | 522/522 [00:00<00:00, 1454.03it/s]
100%|      | 522/522 [00:00<00:00, 2113.83it/s]
100%|      | 522/522 [00:00<00:00, 694.19it/s]
100%|      | 522/522 [00:00<00:00, 833.19it/s]
100%|      | 78/78 [00:06<00:00, 11.53it/s]
100%|      | 78/78 [00:00<00:00, 1032.32it/s]
100%|      | 78/78 [00:00<00:00, 2686.93it/s]
100%|      | 78/78 [00:00<00:00, 5521.52it/s]
100%|      | 78/78 [00:00<00:00, 1109.97it/s]
100%|      | 78/78 [00:00<00:00, 1770.61it/s]
100%|      | 78/78 [00:00<00:00, 2381.62it/s]
100%|      | 78/78 [00:00<00:00, 979.61it/s]
100%|      | 78/78 [00:00<00:00, 1299.25it/s]
```

```
0
0
(522, 38) (78, 38)
[[38  0]
 [ 0 40]]
```

[11]:
```
# --- Analysis ---
# Full report
output_path = Path("1_baseline_model_migration_db_results.xlsx")
report_df.to_excel(output_path, index=True)

print(f"\nSaved full classification report to: {output_path}")
print(report_df)

# Check what the baseline classifier learned
handcrafted_df = baseline_df
handcrafted_feat_names = list(handcrafted_df.columns)
coefs = clf.coef_[0] # shape = (n_features,)

coef_df = pd.DataFrame({
    "feature": handcrafted_feat_names,
    "weight": coefs
})
```

```python
top_ai = coef_df.sort_values("weight", ascending=False).head(20)
top_human = coef_df.sort_values("weight", ascending=True).head(20)

print("\nTop features pushing toward AI:")
print(top_ai)
print("\nTop features pushing toward Human:")
print(top_human)
```

```
Saved full classification report to: 1_baseline_model_migration_db_results.xlsx
              precision  recall  f1-score  support
Human               1.0     1.0       1.0     38.0
AI                  1.0     1.0       1.0     40.0
accuracy            1.0     1.0       1.0      1.0
macro avg           1.0     1.0       1.0     78.0
weighted avg        1.0     1.0       1.0     78.0

Top features pushing toward AI:
            feature    weight
5           punct_,  0.657379
22        upos_NOUN  0.626643
3      std_word_len  0.621383
26        upos_PRON  0.582140
2     mean_word_len  0.461726
12          punct_"  0.426033
28         upos_DET  0.340927
29       upos_CCONJ  0.300897
23        upos_VERB  0.230893
21              ttr  0.206724
24         upos_ADJ  0.134342
8           punct_:  0.083334
6           punct_.  0.042621
16          punct_…  0.000000
7           punct_;  0.000000
10          punct_! -0.037683
33        upos_INTJ -0.060109
9           punct_? -0.067773
25         upos_ADV -0.085508
36  rep_5gram_ratio -0.195342

Top features pushing toward Human:
             feature    weight
35        upos_PROPN -0.735148
15           punct_- -0.734589
11           punct_' -0.727390
18       title_ratio -0.663105
17       upper_ratio -0.632454
31          upos_NUM -0.561319
```

```
1        std_sent_len -0.509759
32           upos_AUX -0.446126
19        digit_ratio -0.444154
37  self_sim_jaccard -0.405192
20       space_ratio -0.394657
30         upos_SCONJ -0.387993
27           upos_ADP -0.386094
34          upos_PART -0.373883
0       mean_sent_len -0.316033
4      stopword_ratio -0.296349
14            punct_) -0.243967
13            punct_( -0.243967
36   rep_5gram_ratio -0.195342
25           upos_ADV -0.085508
```

4 Bias model on PAN dataset

[12]:
```
# Keep original and metadata
train_path = Path(training_data_pan)
val_path = Path(testing_data_pan)

# --- Train bias model ---
df_train = pd.read_json(train_path, lines=True)
df_train = df_train[["id", "text", "label", "genre", "model"]].copy()
df_train["label"] = df_train["label"].astype(int)

# Load lexicons
positive_lexicon = features.load_lexicon(r"C:\Users\andre\OneDrive\Desktop\PAN␣
 ↪Task 2025\positive-words-dictionary.txt")
negative_lexicon = features.load_lexicon(r"C:\Users\andre\OneDrive\Desktop\PAN␣
 ↪Task 2025\negative-words-dictionary.txt")
profanity_lexicon = features.load_profanity_lexicon(r"C:
 ↪\Users\andre\OneDrive\Desktop\PAN Task 2025\toxic-words-dictionary.txt")

# Bias features for TRAIN
bias_df = pd.concat([
    df_train["text"].progress_apply(features.compute_assertive_rate).
 ↪rename("assertive_rate"),
    df_train["text"].progress_apply(features.compute_sentiment_features),  #␣
 ↪already multi-col
    df_train["text"].progress_apply(lambda t: features.
 ↪compute_profanity_rate(t, profanity_lexicon)).rename("profanity_rate"),
    df_train["text"].progress_apply(features.compute_subjectivity_score).
 ↪rename("subjectivity_score"),
    df_train["text"].progress_apply(lambda t: features.compute_hedge_rate(t,␣
 ↪features.hedge_lexicon)).rename("hedge_rate"),
```

```
    df_train["text"].progress_apply(features.compute_identity_term_rates),  # multi-col
    df_train["text"].progress_apply(lambda t: features.compute_emotional_tone_from_lexicons(
        t, positive_lexicon, negative_lexicon
    )),  # multi-col (pos_rate, neg_rate, polarity_score)
    df_train["text"].progress_apply(features.compute_categorical_rate).rename("categorical_rate"),
], axis=1)

# Scale features
scaler_bias = StandardScaler()
X_train_bias = scaler_bias.fit_transform(bias_df)

# Train classifier
clf_bias = LogisticRegression(max_iter=1000, verbose=1)
clf_bias.fit(X_train_bias, df_train["label"])

# Bias features for VAL
df_val = pd.read_json(val_path, lines=True)

bias_val = pd.concat([
    df_val["text"].progress_apply(features.compute_assertive_rate).rename("assertive_rate"),
    df_val["text"].progress_apply(features.compute_sentiment_features),
    df_val["text"].progress_apply(lambda t: features.compute_profanity_rate(t, profanity_lexicon)).rename("profanity_rate"),
    df_val["text"].progress_apply(features.compute_subjectivity_score).rename("subjectivity_score"),
    df_val["text"].progress_apply(lambda t: features.compute_hedge_rate(t, features.hedge_lexicon)).rename("hedge_rate"),
    df_val["text"].progress_apply(features.compute_identity_term_rates),
    df_val["text"].progress_apply(lambda t: features.compute_emotional_tone_from_lexicons(
        t, positive_lexicon, negative_lexicon
    )),
    df_val["text"].progress_apply(features.compute_categorical_rate).rename("categorical_rate"),
], axis=1)

# Scale using SAME scaler
X_val_bias = scaler_bias.transform(bias_val)

# Predict
y_val_pred = clf_bias.predict(X_val_bias)
```

```
df_val = df_val.copy()
df_val["y_true"] = df_val["label"]
df_val["y_pred_bias"] = y_val_pred

# Evaluate
report_df = pd.DataFrame(
    classification_report(
        df_val["label"], y_val_pred,
        target_names=["Human", "AI"],
        output_dict=True
    )
).transpose()

# Sanity check
print(bias_df.isna().sum().sum())
print(bias_val.isna().sum().sum())
print(bias_df.shape, bias_val.shape)
print(confusion_matrix(df_val["label"], y_val_pred))
```

```
100%|      | 23707/23707 [00:15<00:00, 1506.26it/s]
100%|      | 23707/23707 [02:51<00:00, 138.20it/s]
100%|      | 23707/23707 [00:07<00:00, 3296.95it/s]
100%|      | 23707/23707 [00:14<00:00, 1671.21it/s]
100%|      | 23707/23707 [00:08<00:00, 2864.95it/s]
100%|      | 23707/23707 [00:16<00:00, 1412.47it/s]
100%|      | 23707/23707 [00:14<00:00, 1659.57it/s]
100%|      | 23707/23707 [00:16<00:00, 1421.65it/s]
100%|      | 3589/3589 [00:02<00:00, 1565.04it/s]
100%|      | 3589/3589 [00:24<00:00, 144.47it/s]
100%|      | 3589/3589 [00:01<00:00, 2899.33it/s]
100%|      | 3589/3589 [00:02<00:00, 1696.64it/s]
100%|      | 3589/3589 [00:01<00:00, 2804.05it/s]
100%|      | 3589/3589 [00:02<00:00, 1513.17it/s]
100%|      | 3589/3589 [00:01<00:00, 1944.07it/s]
100%|      | 3589/3589 [00:02<00:00, 1444.99it/s]

0
0
(23707, 17) (3589, 17)
[[ 967  310]
 [ 252 2060]]
```

[13]:
```
# --- Analysis ---
# Full report
full_results_output_path = Path("2_bias_model_pan_results.xlsx")
report_df.to_excel(full_results_output_path, index=True)
```

```
print(f"\nSaved bias model report to: {full_results_output_path}")
print(report_df)

# Misclassifications
misclassified_bias = df_val[df_val["y_true"] != df_val["y_pred_bias"]]
misclassified_bias_feats = bias_val.loc[misclassified_bias.index]

misclassified_bias_full = pd.concat(
    [misclassified_bias[["id", "text", "y_true", "y_pred_bias", "model",␣
 ↪"genre"]],
    misclassified_bias_feats],
    axis=1
)

missclassifications_output_path = Path("bias_pan_misclassifications.xlsx")
misclassified_bias_full.to_excel(missclassifications_output_path, index=False)

print(f"Saved {len(misclassified_bias_full)} misclassifications to␣
 ↪{missclassifications_output_path}")

# --- Check what the bias classifier learned ---
handcrafted_df = bias_df.copy()
handcrafted_feat_names = list(handcrafted_df.columns)
coefs = clf_bias.coef_[0]   # shape = (n_features,)

coef_df = pd.DataFrame({
    "feature": handcrafted_feat_names,
    "weight": coefs
})

top_ai = coef_df.sort_values("weight", ascending=False).head(20)
top_human = coef_df.sort_values("weight", ascending=True).head(20)

print("\nTop bias features pushing toward AI:")
print(top_ai)

print("\nTop bias features pushing toward Human:")
print(top_human)
```

```
Saved bias model report to: 2_bias_model_pan_results.xlsx
              precision    recall  f1-score     support
Human          0.793273  0.757244  0.774840  1277.00000
AI             0.869198  0.891003  0.879966  2312.00000
accuracy       0.843410  0.843410  0.843410     0.84341
macro avg      0.831236  0.824123  0.827403  3589.00000
```

```
weighted avg   0.842183  0.843410  0.842561  3589.00000
Saved 562 misclassifications to bias_pan_misclassifications.xlsx

Top bias features pushing toward AI:
                         feature    weight
14                      neg_rate  1.482674
13                      pos_rate  1.199948
1                 sentiment_mean  0.408142
0                 assertive_rate  0.391881
10     orientation_identity_rate  0.133668
11      disability_identity_rate  0.102077
8         religion_identity_rate  0.027694
9      nationality_identity_rate -0.011181
7   race_ethnicity_identity_rate -0.195390
15                polarity_score -0.268695
4              subjectivity_score -0.299980
12              age_identity_rate -0.397164
2                  sentiment_var -0.422614
6           gender_identity_rate -0.475158
5                     hedge_rate -0.537363
3                 profanity_rate -0.684859
16              categorical_rate -1.162688

Top bias features pushing toward Human:
                         feature    weight
16              categorical_rate -1.162688
3                 profanity_rate -0.684859
5                     hedge_rate -0.537363
6           gender_identity_rate -0.475158
2                  sentiment_var -0.422614
12              age_identity_rate -0.397164
4              subjectivity_score -0.299980
15                polarity_score -0.268695
7   race_ethnicity_identity_rate -0.195390
9      nationality_identity_rate -0.011181
8         religion_identity_rate  0.027694
11      disability_identity_rate  0.102077
10     orientation_identity_rate  0.133668
0                 assertive_rate  0.391881
1                 sentiment_mean  0.408142
13                      pos_rate  1.199948
14                      neg_rate  1.482674
```

5 Bias model on migration dataset

[14]:
```
# Keep original and metadata
train_path = Path(training_data_migration)
val_path = Path(testing_data_migration)

df_train = pd.read_excel(train_path)
df_train = df_train[["id", "text", "label", "genre", "model"]].copy()
df_train["label"] = df_train["label"].astype(int)

# --- Train bias model ---

# Load lexicons
positive_lexicon = features.load_lexicon(r"C:\Users\andre\OneDrive\Desktop\PAN␣
 ↪Task 2025\positive-words-dictionary.txt")
negative_lexicon = features.load_lexicon(r"C:\Users\andre\OneDrive\Desktop\PAN␣
 ↪Task 2025\negative-words-dictionary.txt")
profanity_lexicon = features.load_profanity_lexicon(r"C:
 ↪\Users\andre\OneDrive\Desktop\PAN Task 2025\toxic-words-dictionary.txt")

# Bias features for TRAIN
bias_df = pd.concat([
    df_train["text"].progress_apply(features.compute_assertive_rate).
 ↪rename("assertive_rate"),
    df_train["text"].progress_apply(features.compute_sentiment_features),  #␣
 ↪already multi-col
    df_train["text"].progress_apply(lambda t: features.
 ↪compute_profanity_rate(t, profanity_lexicon)).rename("profanity_rate"),
    df_train["text"].progress_apply(features.compute_subjectivity_score).
 ↪rename("subjectivity_score"),
    df_train["text"].progress_apply(lambda t: features.compute_hedge_rate(t,␣
 ↪features.hedge_lexicon)).rename("hedge_rate"),
    df_train["text"].progress_apply(features.compute_identity_term_rates),  #␣
 ↪multi-col
    df_train["text"].progress_apply(lambda t: features.
 ↪compute_emotional_tone_from_lexicons(
        t, positive_lexicon, negative_lexicon
    )),  # multi-col (pos_rate, neg_rate, polarity_score)
    df_train["text"].progress_apply(features.compute_categorical_rate).
 ↪rename("categorical_rate"),
], axis=1)

# Scale features
scaler_bias = StandardScaler()
X_train_bias = scaler_bias.fit_transform(bias_df)
```

```
# Train classifier
clf_bias = LogisticRegression(max_iter=1000, verbose=1)
clf_bias.fit(X_train_bias, df_train["label"])

# --- Bias Feature Classifier Training ---
df_val = pd.read_excel(val_path)

# Bias features for VAL
bias_val = pd.concat([
    df_val["text"].progress_apply(features.compute_assertive_rate).
 ↪rename("assertive_rate"),
    df_val["text"].progress_apply(features.compute_sentiment_features),
    df_val["text"].progress_apply(lambda t: features.compute_profanity_rate(t,␣
 ↪profanity_lexicon)).rename("profanity_rate"),
    df_val["text"].progress_apply(features.compute_subjectivity_score).
 ↪rename("subjectivity_score"),
    df_val["text"].progress_apply(lambda t: features.compute_hedge_rate(t,␣
 ↪features.hedge_lexicon)).rename("hedge_rate"),
    df_val["text"].progress_apply(features.compute_identity_term_rates),
    df_val["text"].progress_apply(lambda t: features.
 ↪compute_emotional_tone_from_lexicons(
        t, positive_lexicon, negative_lexicon
    )),
    df_val["text"].progress_apply(features.compute_categorical_rate).
 ↪rename("categorical_rate"),
], axis=1)

# Scale using SAME scaler
X_val_bias = scaler_bias.transform(bias_val)

# Predict
y_val_pred = clf_bias.predict(X_val_bias)

# Evaluate
report_df = pd.DataFrame(
    classification_report(
        df_val["label"], y_val_pred,
        target_names=["Human", "AI"],
        output_dict=True
    )
).transpose()

# Sanity check
print(bias_df.isna().sum().sum())
print(bias_val.isna().sum().sum())
print(bias_df.shape, bias_val.shape)
```

```
print(confusion_matrix(df_val["label"], y_val_pred))
```

```
100%|      | 522/522 [00:00<00:00, 1351.43it/s]
100%|      | 522/522 [00:03<00:00, 140.72it/s]
100%|      | 522/522 [00:00<00:00, 3359.57it/s]
100%|      | 522/522 [00:00<00:00, 1733.85it/s]
100%|      | 522/522 [00:00<00:00, 2855.02it/s]
100%|      | 522/522 [00:00<00:00, 1521.81it/s]
100%|      | 522/522 [00:00<00:00, 2007.11it/s]
100%|      | 522/522 [00:00<00:00, 1455.84it/s]
100%|      | 78/78 [00:00<00:00, 1628.84it/s]
100%|      | 78/78 [00:00<00:00, 145.94it/s]
100%|      | 78/78 [00:00<00:00, 3290.11it/s]
100%|      | 78/78 [00:00<00:00, 1706.73it/s]
100%|      | 78/78 [00:00<00:00, 2828.04it/s]
100%|      | 78/78 [00:00<00:00, 1484.40it/s]
100%|      | 78/78 [00:00<00:00, 1960.94it/s]
100%|      | 78/78 [00:00<00:00, 1469.44it/s]
```

```
0
0
(522, 17) (78, 17)
[[36  2]
 [ 1 39]]
```

[15]:
```
# --- Analysis ---
# Full report
output_path = Path("2_bias_model_migration_results.xlsx")
report_df.to_excel(output_path, index=True)

print(f"\nSaved bias model report to: {output_path}")
print(report_df)

# Check what the bias classifier learned
handcrafted_df = bias_df.copy()
handcrafted_feat_names = list(handcrafted_df.columns)
coefs = clf_bias.coef_[0]   # shape = (n_features,)

coef_df = pd.DataFrame({
    "feature": handcrafted_feat_names,
    "weight": coefs
})

top_ai = coef_df.sort_values("weight", ascending=False).head(20)
top_human = coef_df.sort_values("weight", ascending=True).head(20)
```

```
print("\nTop bias features pushing toward AI:")
print(top_ai)

print("\nTop bias features pushing toward Human:")
print(top_human)
```

```
Saved bias model report to: 2_bias_model_migration_results.xlsx
              precision    recall  f1-score    support
Human          0.972973  0.947368  0.960000  38.000000
AI             0.951220  0.975000  0.962963  40.000000
accuracy       0.961538  0.961538  0.961538   0.961538
macro avg      0.962096  0.961184  0.961481  78.000000
weighted avg   0.961817  0.961538  0.961519  78.000000

Top bias features pushing toward AI:
                         feature    weight
14                     neg_rate  2.183094
13                     pos_rate  1.806948
1                 sentiment_mean  1.392883
12             age_identity_rate  0.101530
16              categorical_rate -0.012834
15                polarity_score -0.040461
5                     hedge_rate -0.076977
2                  sentiment_var -0.082457
4             subjectivity_score -0.117547
0                 assertive_rate -0.138438
11      disability_identity_rate -0.325031
8         religion_identity_rate -0.735063
10     orientation_identity_rate -0.886660
9      nationality_identity_rate -1.166516
7   race_ethnicity_identity_rate -1.577769
3                 profanity_rate -2.122194
6           gender_identity_rate -2.185796

Top bias features pushing toward Human:
                         feature    weight
6           gender_identity_rate -2.185796
3                 profanity_rate -2.122194
7   race_ethnicity_identity_rate -1.577769
9      nationality_identity_rate -1.166516
10     orientation_identity_rate -0.886660
8         religion_identity_rate -0.735063
11      disability_identity_rate -0.325031
0                 assertive_rate -0.138438
4             subjectivity_score -0.117547
2                  sentiment_var -0.082457
5                     hedge_rate -0.076977
```

```
15                polarity_score -0.040461
16              categorical_rate -0.012834
12            age_identity_rate  0.101530
1                sentiment_mean  1.392883
13                     pos_rate  1.806948
14                     neg_rate  2.183094
```

6 Merged model on PAN dataset

[16]:
```
# Keep original and metadata
train_path = Path(training_data_pan)
val_path = Path(testing_data_pan)

df_train = pd.read_json(train_path, lines=True)
df_train = df_train[["id", "text", "label", "genre", "model"]].copy()
df_train["label"] = df_train["label"].astype(int)

# --- Train merged model ---

# Load lexicons
positive_lexicon = features.load_lexicon(r"C:\Users\andre\OneDrive\Desktop\PAN␣
 ↪Task 2025\positive-words-dictionary.txt")
negative_lexicon = features.load_lexicon(r"C:\Users\andre\OneDrive\Desktop\PAN␣
 ↪Task 2025\negative-words-dictionary.txt")
profanity_lexicon = features.load_profanity_lexicon(r"C:
 ↪\Users\andre\OneDrive\Desktop\PAN Task 2025\toxic-words-dictionary.txt")

# Features from reduced baseline model
df_train["spacy_doc"] = df_train["text"].progress_apply(nlp)

baseline_train_full = pd.concat([
    df_train["spacy_doc"].progress_apply(features.extract_length_features),
    df_train["spacy_doc"].progress_apply(features.extract_stopword_rate),
    df_train["text"].progress_apply(features.extract_punct_rates),
    df_train["text"].progress_apply(features.extract_char_class_ratios),
    df_train["spacy_doc"].progress_apply(features.extract_ttr),
    df_train["spacy_doc"].progress_apply(features.extract_upos_freq),
    df_train["spacy_doc"].progress_apply(features.compute_5gram_repetition),
    df_train["spacy_doc"].progress_apply(features.compute_self_similarity)
], axis=1)

# Bias features
bias_train_full = pd.concat([
    df_train["text"].progress_apply(features.compute_assertive_rate).
 ↪rename("assertive_rate"),
```

```
    df_train["text"].progress_apply(features.compute_sentiment_features),  # already multi-col
    df_train["text"].progress_apply(lambda t: features.compute_profanity_rate(t, profanity_lexicon)).rename("profanity_rate"),
    df_train["text"].progress_apply(features.compute_subjectivity_score).rename("subjectivity_score"),
    df_train["text"].progress_apply(lambda t: features.compute_hedge_rate(t, features.hedge_lexicon)).rename("hedge_rate"),
    df_train["text"].progress_apply(features.compute_identity_term_rates),  # multi-col
    df_train["text"].progress_apply(lambda t: features.compute_emotional_tone_from_lexicons(
        t, positive_lexicon, negative_lexicon
    )),  # multi-col (pos_rate, neg_rate, polarity_score)
    df_train["text"].progress_apply(features.compute_categorical_rate).rename("categorical_rate"),
], axis=1)

# Merge all features
merged_train_full = pd.concat([baseline_train_full, bias_train_full], axis=1)
feature_order = merged_train_full.columns

# Scale
scaler_merged = StandardScaler()
X_train_merged = scaler_merged.fit_transform(merged_train_full)

# Train classifier
clf_merged = LogisticRegression(max_iter=1000, verbose=1)
clf_merged.fit(X_train_merged, df_train["label"])

# --- Validate ---
df_val = pd.read_json(val_path, lines=True)
df_val["spacy_doc"] = df_val["text"].progress_apply(nlp)

# Features from reduced baseline model
baseline_val_full = pd.concat([
    df_val["spacy_doc"].progress_apply(features.extract_length_features),
    df_val["spacy_doc"].progress_apply(features.extract_stopword_rate),
    df_val["text"].progress_apply(features.extract_punct_rates),
    df_val["text"].progress_apply(features.extract_char_class_ratios),
    df_val["spacy_doc"].progress_apply(features.extract_ttr),
    df_val["spacy_doc"].progress_apply(features.extract_upos_freq),
    df_val["spacy_doc"].progress_apply(features.compute_5gram_repetition),
    df_val["spacy_doc"].progress_apply(features.compute_self_similarity)
], axis=1)
```

```
# Bias features
bias_val_full = pd.concat([
    df_val["text"].progress_apply(features.compute_assertive_rate).
 ↪rename("assertive_rate"),
    df_val["text"].progress_apply(features.compute_sentiment_features),
    df_val["text"].progress_apply(lambda t: features.compute_profanity_rate(t,␣
 ↪profanity_lexicon)).rename("profanity_rate"),
    df_val["text"].progress_apply(features.compute_subjectivity_score).
 ↪rename("subjectivity_score"),
    df_val["text"].progress_apply(lambda t: features.compute_hedge_rate(t,␣
 ↪features.hedge_lexicon)).rename("hedge_rate"),
    df_val["text"].progress_apply(features.compute_identity_term_rates),
    df_val["text"].progress_apply(lambda t: features.
 ↪compute_emotional_tone_from_lexicons(
        t, positive_lexicon, negative_lexicon
    )),
    df_val["text"].progress_apply(features.compute_categorical_rate).
 ↪rename("categorical_rate"),
], axis=1)

merged_val_full = pd.concat([baseline_val_full, bias_val_full], axis=1)
merged_val_full = merged_val_full.reindex(columns=feature_order)

# Scale using same scaler
X_val_merged = scaler_merged.transform(merged_val_full)

# Predict
y_val_pred = clf_merged.predict(X_val_merged)

df_val = df_val.copy()
df_val["y_true"] = df_val["label"]
df_val["y_pred"] = y_val_pred

# Evaluate
report_df = pd.DataFrame(
    classification_report(
        df_val["label"], y_val_pred,
        target_names=["Human", "AI"],
        output_dict=True)
).transpose()

# Sanity check
print(merged_train_full.isna().sum().sum())
print(merged_val_full.isna().sum().sum())
print(merged_train_full.shape, merged_val_full.shape)
```

```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(df_val["label"], y_val_pred))
```

```
100%|      | 23707/23707 [32:44<00:00, 12.07it/s]
100%|      | 23707/23707 [00:32<00:00, 725.36it/s]
100%|      | 23707/23707 [00:09<00:00, 2606.39it/s]
100%|      | 23707/23707 [00:05<00:00, 4184.88it/s]
100%|      | 23707/23707 [00:21<00:00, 1093.65it/s]
100%|      | 23707/23707 [00:16<00:00, 1479.82it/s]
100%|      | 23707/23707 [00:10<00:00, 2181.87it/s]
100%|      | 23707/23707 [00:30<00:00, 772.77it/s]
100%|      | 23707/23707 [00:22<00:00, 1074.84it/s]
100%|      | 23707/23707 [00:15<00:00, 1553.27it/s]
100%|      | 23707/23707 [02:50<00:00, 139.30it/s]
100%|      | 23707/23707 [00:13<00:00, 1703.46it/s]
100%|      | 23707/23707 [00:25<00:00, 941.93it/s]
100%|      | 23707/23707 [00:15<00:00, 1530.98it/s]
100%|      | 23707/23707 [00:29<00:00, 794.24it/s]
100%|      | 23707/23707 [00:21<00:00, 1100.11it/s]
100%|      | 23707/23707 [00:28<00:00, 827.92it/s]
100%|      | 3589/3589 [05:48<00:00, 10.31it/s]
100%|      | 3589/3589 [00:04<00:00, 797.93it/s]
100%|      | 3589/3589 [00:01<00:00, 2355.86it/s]
100%|      | 3589/3589 [00:00<00:00, 5465.54it/s]
100%|      | 3589/3589 [00:03<00:00, 948.46it/s]
100%|      | 3589/3589 [00:02<00:00, 1209.14it/s]
100%|      | 3589/3589 [00:01<00:00, 1968.78it/s]
100%|      | 3589/3589 [00:04<00:00, 750.23it/s]
100%|      | 3589/3589 [00:02<00:00, 1203.76it/s]
100%|      | 3589/3589 [00:02<00:00, 1465.40it/s]
100%|      | 3589/3589 [00:26<00:00, 133.19it/s]
100%|      | 3589/3589 [00:01<00:00, 3282.56it/s]
100%|      | 3589/3589 [00:02<00:00, 1764.81it/s]
100%|      | 3589/3589 [00:01<00:00, 2898.15it/s]
100%|      | 3589/3589 [00:02<00:00, 1282.04it/s]
100%|      | 3589/3589 [00:01<00:00, 1977.84it/s]
100%|      | 3589/3589 [00:02<00:00, 1473.10it/s]

0
0
(23707, 55) (3589, 55)
[[1189   88]
 [  50 2262]]
```

[17]:
```
# --- Analysis ---
# Full report
```

```
full_results_output_path = Path("3_merged_pan_results.xlsx")
report_df.to_excel(full_results_output_path, index=True)

print(f"\nSaved merged model report to: {full_results_output_path}")
print(report_df)

# Misclassifications
misclassified = df_val[df_val["y_true"] != df_val["y_pred"]]
misclassified_feats = merged_val_full.loc[misclassified.index]

misclassified_merged = pd.concat(
    [misclassified[["id", "text", "y_true", "y_pred", "model", "genre"]],
    misclassified_feats],
    axis=1
)

missclassifications_output_path = Path("merged_pan_misclassifications.xlsx")
misclassified_merged.to_excel(missclassifications_output_path, index=False)

print(f"Saved {len(misclassified_merged)} misclassifications to␣
 ↪{missclassifications_output_path}")

# Check what the merged classifier learned
handcrafted_df = merged_train_full.copy()
handcrafted_feat_names = list(feature_order)
coefs = clf_merged.coef_[0]   # shape = (n_features,)

coef_df = pd.DataFrame({
    "feature": handcrafted_feat_names,
    "weight": coefs
})

# Tag feature origin
coef_df["feature_type"] = np.where(
    coef_df["feature"].isin(baseline_train_full.columns),
    "baseline",
    "bias"
)

top_ai = coef_df.sort_values("weight", ascending=False).head(20)
top_human = coef_df.sort_values("weight", ascending=True).head(20)

print("\nTop merged features pushing toward AI:")
print(top_ai)

print("\nTop merged features pushing toward Human:")
print(top_human)
```

```
Saved merged model report to: 3_merged_pan_results.xlsx
              precision    recall  f1-score      support
Human          0.959645  0.931088  0.945151  1277.000000
AI             0.962553  0.978374  0.970399  2312.000000
accuracy       0.961549  0.961549  0.961549     0.961549
macro avg      0.961099  0.954731  0.957775  3589.000000
weighted avg   0.961518  0.961549  0.961416  3589.000000
Saved 138 misclassifications to merged_pan_misclassifications.xlsx

Top merged features pushing toward AI:
               feature    weight feature_type
26           upos_PRON  2.261831     baseline
2        mean_word_len  1.494952     baseline
5              punct_,  1.448293     baseline
21                 ttr  1.262385     baseline
28           upos_DET  1.110786     baseline
52           neg_rate  0.999864         bias
37    self_sim_jaccard  0.989597     baseline
36     rep_5gram_ratio  0.833820     baseline
51           pos_rate  0.822653         bias
19        digit_ratio  0.570215     baseline
0        mean_sent_len  0.566931     baseline
11             punct_'  0.525353     baseline
39      sentiment_mean  0.491009         bias
34           upos_PART  0.380480     baseline
18        title_ratio  0.215660     baseline
9              punct_?  0.149644     baseline
17        upper_ratio  0.132821     baseline
3         std_word_len  0.094634     baseline
42  subjectivity_score  0.084035         bias
13             punct_(  0.043566     baseline

Top merged features pushing toward Human:
                     feature    weight feature_type
4              stopword_ratio -2.843400     baseline
7                    punct_; -1.579051     baseline
35                upos_PROPN -1.255422     baseline
15                   punct_- -1.050674     baseline
1                std_sent_len -1.036793     baseline
20                space_ratio -1.034956     baseline
24                   upos_ADJ -0.862468     baseline
31                   upos_NUM -0.759919     baseline
25                   upos_ADV -0.564936     baseline
33                  upos_INTJ -0.443324     baseline
23                  upos_VERB -0.436430     baseline
41             profanity_rate -0.365874         bias
10                   punct_! -0.305412     baseline
```

```
32                   upos_AUX -0.244425     baseline
53             polarity_score -0.172855         bias
14                    punct_) -0.171282     baseline
54           categorical_rate -0.153850         bias
22                  upos_NOUN -0.149626     baseline
47  nationality_identity_rate -0.132754         bias
29                 upos_CCONJ -0.107629     baseline
```

7 Check a random entry from the PAN data set

[23]:
```
import random
import numpy as np
import pandas as pd
from scipy.sparse import csr_matrix
from sklearn.metrics import classification_report

# Select random sample
i = random.randint(0, len(df_val) - 1)
example = df_val.iloc[[i]].copy()
text = example["text"].values[0]

print("Ground truth:", "AI" if example["label"].values[0] == 1 else "Human")
print("\nText excerpt:\n", text[:300], "...")

# Baseline feature extraction
baseline_ex = pd.concat([
    example["spacy_doc"].progress_apply(features.extract_length_features),
    example["spacy_doc"].progress_apply(features.extract_stopword_rate),
    example["text"].progress_apply(features.extract_punct_rates),
    example["text"].progress_apply(features.extract_char_class_ratios),
    example["spacy_doc"].progress_apply(features.extract_ttr),
    example["spacy_doc"].progress_apply(features.extract_upos_freq),
    example["spacy_doc"].progress_apply(features.compute_5gram_repetition),
    example["spacy_doc"].progress_apply(features.compute_self_similarity),
], axis=1)

# scale
X_base_scaled = scaler.transform(baseline_ex)
X_base_sparse = csr_matrix(X_base_scaled)

# predict
proba_base = clf.predict_proba(X_base_sparse)[0][1]
pred_base = clf.predict(X_base_sparse)[0]

# Bias feature extraction
bias_ex = pd.concat([
```

```
    example["text"].progress_apply(features.compute_assertive_rate).
 ↪rename("assertive_rate"),
    example["text"].progress_apply(features.compute_sentiment_features),  #␣
 ↪already multi-col
    example["text"].progress_apply(lambda t: features.compute_profanity_rate(t,␣
 ↪profanity_lexicon)).rename("profanity_rate"),
    example["text"].progress_apply(features.compute_subjectivity_score).
 ↪rename("subjectivity_score"),
    example["text"].progress_apply(lambda t: features.compute_hedge_rate(t,␣
 ↪features.hedge_lexicon)).rename("hedge_rate"),
    example["text"].progress_apply(features.compute_identity_term_rates),  #␣
 ↪multi-col
    example["text"].progress_apply(lambda t: features.
 ↪compute_emotional_tone_from_lexicons(
        t, positive_lexicon, negative_lexicon
    )),  # multi-col (pos_rate, neg_rate, polarity_score)
    example["text"].progress_apply(features.compute_categorical_rate).
 ↪rename("categorical_rate"),
], axis=1)


X_bias_scaled = scaler_bias.transform(bias_ex)
proba_bias = clf_bias.predict_proba(X_bias_scaled)[0][1]
pred_bias = clf_bias.predict(X_bias_scaled)[0]

# Merged feature extraction
merged_ex = pd.concat([baseline_ex, bias_ex], axis=1)
merged_ex = merged_ex.reindex(columns=feature_order)

X_merge_scaled = scaler_merged.transform(merged_ex)
proba_merge = clf_merged.predict_proba(X_merge_scaled)[0][1]
pred_merge = clf_merged.predict(X_merge_scaled)[0]


# Results
def show_pred(name, pred, proba):
    print(f"\n{name}: {'AI' if pred==1 else 'Human'}"
          f"  (confidence={proba:.3f})")

show_pred("Baseline", pred_base, proba_base)
show_pred("Bias", pred_bias, proba_bias)
show_pred("Merged", pred_merge, proba_merge)
```

```
Ground truth: AI

Text excerpt:
 Brady and Stafford: A Tale of Two Quarterbacks
```

```
Los Angeles, CA - In a highly anticipated matchup, two of the NFL's most
accomplished quarterbacks, Tom Brady and Matthew Stafford, will face off as the
Los Angeles Rams host the Tampa Bay Buccaneers. While their paths to their
current teams have been  …
```

```
100%|      | 1/1 [00:00<00:00, 359.01it/s]
100%|      | 1/1 [00:00<00:00, 1156.09it/s]
100%|      | 1/1 [00:00<00:00, 1264.49it/s]
100%|      | 1/1 [00:00<00:00, 187.98it/s]
100%|      | 1/1 [00:00<00:00, 400.33it/s]
100%|      | 1/1 [00:00<00:00, 406.94it/s]
100%|      | 1/1 [00:00<00:00, 329.04it/s]
100%|      | 1/1 [00:00<00:00, 369.74it/s]
100%|      | 1/1 [00:00<00:00, 390.02it/s]
100%|      | 1/1 [00:00<00:00, 80.59it/s]
100%|      | 1/1 [00:00<00:00, 1546.00it/s]
100%|      | 1/1 [00:00<00:00, 1078.78it/s]
100%|      | 1/1 [00:00<00:00, 1563.29it/s]
100%|      | 1/1 [00:00<00:00, 530.72it/s]
100%|      | 1/1 [00:00<00:00, 658.76it/s]
100%|      | 1/1 [00:00<00:00, 668.20it/s]
```

```
Baseline: Human  (confidence=0.001)

Bias: Human  (confidence=0.014)

Merged: AI  (confidence=0.583)
```

8 Test it with your own input

[24]:
```
import pandas as pd
import numpy as np
from scipy.sparse import csr_matrix
from sklearn.metrics import classification_report

def predict_and_explain(model_name, clf, feature_df, scaler,␣
 ↪feature_order=None):
    # If merged model → enforce identical training column order
    if feature_order is not None:
        feature_df = feature_df.reindex(columns=feature_order).fillna(0)
    else:
        feature_df = feature_df.fillna(0)

    # Scale
```

```
    X_scaled = scaler.transform(feature_df)

    # Sparse for baseline / merged
    X_sparse = csr_matrix(X_scaled)

    # Predict
    proba = clf.predict_proba(X_sparse)[0][1]
    pred  = clf.predict(X_sparse)[0]
    pred_label = "AI" if pred == 1 else "Human"

    print(f"\n=== {model_name} ===")
    print(f"Prediction: {pred_label}  (confidence={proba:.3f})")

    # Explain influence
    coefs = clf.coef_[0]
    vec   = X_sparse.toarray().flatten()
    contribs = vec * coefs
    feature_names = feature_df.columns.tolist()

    top_pos = np.argsort(contribs)[-3:][::-1]
    top_neg = np.argsort(contribs)[:3]

    print("\nTop features pushing toward AI:")
    for idx in top_pos:
        if vec[idx] != 0:
            print(f"{feature_names[idx]:<32} {contribs[idx]:.4f}")

    print("\nTop features pushing toward Human:")
    for idx in top_neg:
        if vec[idx] != 0:
            print(f"{feature_names[idx]:<32} {contribs[idx]:.4f}")
    print("-" * 80)


# Manual input
my_text = """
Time moves differently in places wrapped by mountains and quiet valleys; one␣
 ↪day feels as patient as a slow climb, another as fleeting as a bird darting␣
 ↪through pines.
In Switzerland, people often walk to the train or stand at a tram stop and␣
 ↪glance up at snow-tipped peaks - almost as if the mountains are part of␣
 ↪daily rhythm rather than distant scenery.
The multilingual buzz in cafés, from German to French to Italian, carries that␣
 ↪subtle sense of multiple worlds coexisting side by side, each greeting the␣
 ↪next in its own voice.
```

```
Everyday order-clean streets, on-time trains, polite commuters-is underpinned␣
 ↪by a kind of silent agreement, a willingness to stay small, to notice the␣
 ↪detail.
And yet under that calm surface there's movement: ideas flow, debates happen in␣
 ↪parliaments and living rooms alike, change sometimes sneaks in as quietly as␣
 ↪fog rolling down a slope.
I often wonder if that balance-between stillness and motion, tradition and␣
 ↪change-keeps Swiss life alive in a way harder to see on glossy postcards.
Maybe the real charm lies in how ordinary moments carry a little weight, gentle␣
 ↪but firm, as steady as the mountains.
"""

# Wrap like a one-row dataset
example = pd.DataFrame({"text": [my_text]})
example["spacy_doc"] = example["text"].apply(nlp)


# === Extract each model's feature set exactly like training ===

# Baseline features
baseline_ex = pd.concat([
    example["spacy_doc"].apply(features.extract_length_features),
    example["spacy_doc"].apply(features.extract_stopword_rate),
    example["text"].apply(features.extract_punct_rates),
    example["text"].apply(features.extract_char_class_ratios),
    example["spacy_doc"].apply(features.extract_ttr),
    example["spacy_doc"].apply(features.extract_upos_freq),
    example["spacy_doc"].apply(features.compute_5gram_repetition),
    example["spacy_doc"].apply(features.compute_self_similarity),
], axis=1)

# Bias features
bias_ex = pd.concat([
    example["text"].progress_apply(features.compute_assertive_rate).
 ↪rename("assertive_rate"),
    example["text"].progress_apply(features.compute_sentiment_features),  #␣
 ↪already multi-col
    example["text"].progress_apply(lambda t: features.compute_profanity_rate(t,␣
 ↪profanity_lexicon)).rename("profanity_rate"),
    example["text"].progress_apply(features.compute_subjectivity_score).
 ↪rename("subjectivity_score"),
    example["text"].progress_apply(lambda t: features.compute_hedge_rate(t,␣
 ↪features.hedge_lexicon)).rename("hedge_rate"),
    example["text"].progress_apply(features.compute_identity_term_rates),  #␣
 ↪multi-col
```

```
    example["text"].progress_apply(lambda t: features.
 ↪compute_emotional_tone_from_lexicons(
        t, positive_lexicon, negative_lexicon
    )),  # multi-col (pos_rate, neg_rate, polarity_score)
    example["text"].progress_apply(features.compute_categorical_rate).
 ↪rename("categorical_rate"),
], axis=1)

# Full merged set (same build as training)
merged_ex = pd.concat([baseline_ex, bias_ex], axis=1)


# ===== Run all three models =====
predict_and_explain("BASELINE MODEL", clf, baseline_ex, scaler)
predict_and_explain("BIAS MODEL", clf_bias, bias_ex, scaler_bias)
predict_and_explain("MERGED MODEL", clf_merged, merged_ex, scaler_merged,␣
 ↪feature_order)
```

```
100%|      | 1/1 [00:00<00:00, 2438.55it/s]
100%|      | 1/1 [00:00<00:00, 350.49it/s]
100%|      | 1/1 [00:00<00:00, 2824.45it/s]
100%|      | 1/1 [00:00<00:00, 1952.66it/s]
100%|      | 1/1 [00:00<00:00, 2641.25it/s]
100%|      | 1/1 [00:00<00:00, 1186.51it/s]
100%|      | 1/1 [00:00<00:00, 1399.97it/s]
100%|      | 1/1 [00:00<00:00, 1974.72it/s]


=== BASELINE MODEL ===
Prediction: AI  (confidence=0.721)

Top features pushing toward AI:
upos_AUX                        1.6901
std_sent_len                    0.9171
self_sim_jaccard                0.7937

Top features pushing toward Human:
upos_PRON                       -1.2292
upos_ADP                        -1.1505
std_word_len                    -0.9150
--------------------------------------------------------------------------------

=== BIAS MODEL ===
Prediction: Human  (confidence=0.111)

Top features pushing toward AI:
pos_rate                        1.7573
profanity_rate                  1.6289
```

```
sentiment_mean                  1.2632

Top features pushing toward Human:
neg_rate                        -4.0221
nationality_identity_rate       -3.3572
hedge_rate                      -0.1590
--------------------------------------------------------------------------------

=== MERGED MODEL ===
Prediction: Human  (confidence=0.196)

Top features pushing toward AI:
ttr                             4.3853
stopword_ratio                  2.5673
std_sent_len                    0.9756

Top features pushing toward Human:
upos_PRON                       -3.5327
self_sim_jaccard                -1.8105
upos_ADJ                        -1.6912
--------------------------------------------------------------------------------
```