In [8]:
```
import os
from os.path import join
from pathlib import Path
import torch
from tqdm import tqdm
import random
import pandas as pd

import sys

sys.path.append(os.path.abspath(Path('..')))

from notebooks.profiles import MahalanobisProfile
from notebooks.feature_extractors import POSPCAExtractor, HeuristicExtr
actor
```

In [2]:
```
bawe_group_dir = Path('../data/preprocess/bawe-group')
```

Progress Report 1

This is a summary of all the work that has been done over winter break/the first few weeks of the semester.

Current Progress

We currently use profile-based methods to determine whether a new text belongs to a given author. Given a series of texts for an author, we compute a profile for that author, and then make a decision about whether a new text is from that author based on the distance from the new text to the profile. The current profile implementation is based on finding the mean feature vector given the different feature vectors for a text, and then the mahalanobis distance between the mean and a feature vector for the new text is computed. We then find the probability that the new text belongs to the profile, and output that. In the below tests, a threshold of 90% was used to

flag essays as either belonging to the same author or a different author. We currently have 3 different feature extractions methods being worked on:

1. PCA based on POS bigram counts (good accuracy, currently best model)
2. Heuristics gathering (chance accuracy, but may be able to boost performance of bigram counts)
    - Will likely have better accuracy when we use all heuristics and gather more
3. LSTM POS encoder (WIP)

Next Steps

1. Interpretability issues
    - We would like to have some kind of interface to explain to the user why an essay is being flagged.
    - Use interpretable model, but if probability is too close to threshold, use high performance model and prompt for review?
    - Use high performance model, but display most relevant heuristics to aid in diagnosis
        - Correlation does not equal causation?
    - If nothing else, we can heavily encourage the user to review any essays that are flagged and make a decision for themselves.
2. Are there any other feature extraction techniques we should try?
3. Adversarial models?
    - Train model to turn text from a different author into text that is accepted by the profile
        - May aid in assessing whether the profiles are resilient to patch writing
        - Concern: model may find a way to perfectly capture the style of an author and reform the text to that style, this new text *should* be accepted

Precision, Recall, and Accuracy on the BAWE data

We used the British Academic Written English Corpus (BAWE) to test the precision, recall, and accuracy of our profilers. Five authors are randomly selected from the set, and then they are compared with 20 random different authors. The first essay of an author is used to "profile" them, and then their essays along with the essays from the different authors are tested against this

profile. The profile should return a high probability for essays that belong to the profile and a low probability for essays that don't. For this demo, a cutoff of 90% is used.

In [3]:
```
def test_profile(profile, cutoff):
    ids = os.listdir(bawe_group_dir)
    other_ids = ids.copy()
    random.shuffle(ids)
    random.shuffle(other_ids)

    irrelevant, false_positives, relevant, true_positives = 0, 0, 0, 0
    # For performance reasons, only choose 5 authors
    for id in ids[:10]:
        profile.reset()
        texts = file_texts(id)

        first_text = texts[0]
        other_texts = texts[1:]

        # This author's first essay is used to profile that author
        profile.feed(first_text)

        # The rest of their essays are tested against this profile
        other_scores = torch.tensor([profile.score(text) for text in ot
her_texts])

        irrelevant += len(texts)
        false_positives += sum(other_scores < cutoff)

        # Compare this author to 20 random different authors
        new_relevant, new_true_positives = grade_others(profile, other_
ids[:30], id, cutoff)
        relevant += new_relevant
        true_positives += new_true_positives

    # Precision is relevant selections out of all selections
    precision = true_positives / (true_positives + false_positives)
    # Recall is all relevant selections out of all relevant items
    recall = true_positives / relevant
```

```
    false_negatives = relevant - true_positives

    # False positives and false negatives sum to the error count, and
    # irrelevant and relevant items sum to the whole set
    error_rate = false_negatives + false_positives / (irrelevant + rele
vant)

    return float(precision), float(recall), float(error_rate)


def file_texts(id):
    filenames = os.listdir(join(bawe_group_dir, id))
    texts = []
    for filename in filenames:
        with open(join(bawe_group_dir, f'{id}/{filename}'), 'r') as f:
            texts.append(f.read())

    return texts


def grade_others(profile, ids, id, cutoff):
    rest_ids = [other_id for other_id in ids if other_id != id]

    relevant, true_positives = 0, 0
    for other_id in tqdm(rest_ids):
        texts = file_texts(other_id)

        scores = torch.tensor([profile.score(text) for text in texts])

        true_positives += sum(scores < cutoff)
        relevant += len(texts)

    return relevant, true_positives
```

In [4]:
```
pospca_extractor = POSPCAExtractor(4, 10)
pospca_profile = MahalanobisProfile(pospca_extractor)

pospca_precision, pospca_recall, pospca_error = test_profile(pospca_pro
file, 0.50)
```

```
100%|##########| 30/30 [02:02<00:00,  4.09s/it]
100%|##########| 30/30 [02:05<00:00,  4.18s/it]
100%|##########| 30/30 [02:06<00:00,  4.22s/it]
100%|##########| 30/30 [02:04<00:00,  4.14s/it]
100%|##########| 30/30 [02:04<00:00,  4.16s/it]
100%|##########| 30/30 [02:07<00:00,  4.24s/it]
100%|##########| 30/30 [02:13<00:00,  4.45s/it]
100%|##########| 30/30 [02:13<00:00,  4.44s/it]
100%|##########| 30/30 [02:12<00:00,  4.41s/it]
100%|##########| 30/30 [02:08<00:00,  4.28s/it]
```

In [5]:
```
heuristics_extractor = HeuristicExtractor(4)
heuristics_profile = MahalanobisProfile(heuristics_extractor)

heuristics_precision, heuristics_recall, heuristics_error = test_profil
e(heuristics_profile, 0.90)
```

```
100%|##########| 30/30 [00:36<00:00,  1.21s/it]
100%|##########| 30/30 [00:34<00:00,  1.15s/it]
100%|##########| 30/30 [00:34<00:00,  1.15s/it]
100%|##########| 30/30 [00:34<00:00,  1.15s/it]
100%|##########| 30/30 [00:34<00:00,  1.14s/it]
100%|##########| 30/30 [00:34<00:00,  1.16s/it]
100%|##########| 30/30 [00:34<00:00,  1.15s/it]
100%|##########| 30/30 [00:35<00:00,  1.17s/it]
100%|##########| 30/30 [00:35<00:00,  1.17s/it]
100%|##########| 29/29 [00:34<00:00,  1.19s/it]
```

In [10]:
```
metric_data = [
    [pospca_precision, pospca_recall, pospca_error],
    [heuristics_precision, heuristics_recall, heuristics_error]
]

pd.DataFrame(metric_data, columns=['Precision', 'Recall', 'Error'], ind
ex=['POSPCA', 'Heuristics'])
```

Out[10]:

	Precision	Recall	Error

	Precision	Recall	Error
POSPCA	0.979986	1.000000	0.019877
Heuristics	0.977012	0.742653	324.016846