3_0_2_Naive_Bayes_Baseline_Re_Balanced_Dataset_FINAL

April 13, 2025

0.1 Setup

[1]:

```
#@title Install Packages
```

[2]:

```
!pip install -q transformers
!pip install -q torchinfo
!pip install -q datasets
!pip install -q evaluate
!pip install -q nltk
!pip install -q contractions
!pip install -q hf_xet
!pip install -q sentencepiece
```

```
                          491.2/491.2 kB
9.2 MB/s eta 0:00:00
                          116.3/116.3 kB
8.5 MB/s eta 0:00:00
                          183.9/183.9 kB
11.5 MB/s eta 0:00:00
                          143.5/143.5 kB
9.4 MB/s eta 0:00:00
                          194.8/194.8 kB
13.8 MB/s eta 0:00:00
```

```
ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of the
following dependency conflicts.
torch 2.6.0+cu124 requires nvidia-cublas-cu12==12.4.5.8; platform_system ==
"Linux" and platform_machine == "x86_64", but you have nvidia-cublas-cu12
12.5.3.2 which is incompatible.
torch 2.6.0+cu124 requires nvidia-cuda-cupti-cu12==12.4.127; platform_system ==
"Linux" and platform_machine == "x86_64", but you have nvidia-cuda-cupti-cu12
12.5.82 which is incompatible.
torch 2.6.0+cu124 requires nvidia-cuda-nvrtc-cu12==12.4.127; platform_system ==
"Linux" and platform_machine == "x86_64", but you have nvidia-cuda-nvrtc-cu12
12.5.82 which is incompatible.
torch 2.6.0+cu124 requires nvidia-cuda-runtime-cu12==12.4.127; platform_system
== "Linux" and platform_machine == "x86_64", but you have nvidia-cuda-runtime-
cu12 12.5.82 which is incompatible.
torch 2.6.0+cu124 requires nvidia-cudnn-cu12==9.1.0.70; platform_system ==
"Linux" and platform_machine == "x86_64", but you have nvidia-cudnn-cu12
9.3.0.75 which is incompatible.
torch 2.6.0+cu124 requires nvidia-cufft-cu12==11.2.1.3; platform_system ==
"Linux" and platform_machine == "x86_64", but you have nvidia-cufft-cu12
11.2.3.61 which is incompatible.
torch 2.6.0+cu124 requires nvidia-curand-cu12==10.3.5.147; platform_system ==
"Linux" and platform_machine == "x86_64", but you have nvidia-curand-cu12
10.3.6.82 which is incompatible.
torch 2.6.0+cu124 requires nvidia-cusolver-cu12==11.6.1.9; platform_system ==
"Linux" and platform_machine == "x86_64", but you have nvidia-cusolver-cu12
11.6.3.83 which is incompatible.
torch 2.6.0+cu124 requires nvidia-cusparse-cu12==12.3.1.170; platform_system ==
"Linux" and platform_machine == "x86_64", but you have nvidia-cusparse-cu12
12.5.1.3 which is incompatible.
torch 2.6.0+cu124 requires nvidia-nvjitlink-cu12==12.4.127; platform_system ==
"Linux" and platform_machine == "x86_64", but you have nvidia-nvjitlink-cu12
12.5.82 which is incompatible.
gcsfs 2025.3.2 requires fsspec==2025.3.2, but you have fsspec 2024.12.0 which is
incompatible.
                         84.0/84.0 kB
```

```
1.3 MB/s eta 0:00:00
                         289.9/289.9 kB
8.5 MB/s eta 0:00:00
                         118.3/118.3 kB
11.0 MB/s eta 0:00:00
                         53.8/53.8 MB
41.9 MB/s eta 0:00:00
```

[3]:

```
!sudo apt-get update
! sudo apt-get install tree
```

```
Hit:1 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_64
InRelease
Get:2 https://cloud.r-project.org/bin/linux/ubuntu jammy-cran40/ InRelease
[3,632 B]
Get:3 https://r2u.stat.illinois.edu/ubuntu jammy InRelease [6,555 B]
Get:4 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Hit:5 http://archive.ubuntu.com/ubuntu jammy InRelease
Get:6 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Get:7 https://r2u.stat.illinois.edu/ubuntu jammy/main amd64 Packages [2,688 kB]
Hit:8 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy InRelease
Hit:9 https://ppa.launchpadcontent.net/graphics-drivers/ppa/ubuntu jammy
InRelease
Get:10 https://r2u.stat.illinois.edu/ubuntu jammy/main all Packages [8,824 kB]
Hit:11 https://ppa.launchpadcontent.net/ubuntugis/ppa/ubuntu jammy InRelease
Get:12 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages
[2,788 kB]
Hit:13 http://archive.ubuntu.com/ubuntu jammy-backports InRelease
Get:14 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages
[1,542 kB]
Get:15 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 Packages
[1,243 kB]
Get:16 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [3,099
kB]
Fetched 20.5 MB in 2s (9,673 kB/s)
Reading package lists… Done
W: Skipping acquire of configured file 'main/source/Sources' as repository
'https://r2u.stat.illinois.edu/ubuntu jammy InRelease' does not seem to provide
it (sources.list entry misspelt?)
Reading package lists… Done
Building dependency tree… Done
Reading state information… Done
The following NEW packages will be installed:
  tree
0 upgraded, 1 newly installed, 0 to remove and 31 not upgraded.
Need to get 47.9 kB of archives.
After this operation, 116 kB of additional disk space will be used.
```

```
Get:1 http://archive.ubuntu.com/ubuntu jammy/universe amd64 tree amd64 2.0.2-1
[47.9 kB]
Fetched 47.9 kB in 0s (109 kB/s)
debconf: unable to initialize frontend: Dialog
debconf: (No usable dialog-like program is installed, so the dialog based
frontend cannot be used. at /usr/share/perl5/Debconf/FrontEnd/Dialog.pm line 78,
<> line 1.)
debconf: falling back to frontend: Readline
debconf: unable to initialize frontend: Readline
debconf: (This frontend requires a controlling tty.)
debconf: falling back to frontend: Teletype
dpkg-preconfigure: unable to re-open stdin:
Selecting previously unselected package tree.
(Reading database … 126315 files and directories currently installed.)
Preparing to unpack …/tree_2.0.2-1_amd64.deb …
Unpacking tree (2.0.2-1) …
Setting up tree (2.0.2-1) …
Processing triggers for man-db (2.10.2-1) …
```

[4]:

```python
#@title Imports
import nltk
from nltk.tokenize import RegexpTokenizer

import contractions

import evaluate
import transformers
import torch

from torchinfo import summary

from datasets import load_dataset, Dataset, DatasetDict

from transformers import AutoTokenizer, AutoModel,␣
 ↪AutoModelForSequenceClassification, TrainingArguments, Trainer

import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import sklearn

import spacy

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report,␣
 ↪precision_recall_fscore_support, accuracy_score

import sentencepiece
```

[5]:
```
# @title Mount Google Drive
```

[6]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[7]:
```
dir_root = '/content/drive/MyDrive/266-final/'
# dir_data = '/content/drive/MyDrive/266-final/data/'
# dir_data = '/content/drive/MyDrive/266-final/data/se21-t1-comp-lex-master/'
dir_data = '/content/drive/MyDrive/266-final/data/266-comp-lex-master'
dir_models = '/content/drive/MyDrive/266-final/models/'
dir_results = '/content/drive/MyDrive/266-final/results/'
```

[8]:
```
log_filename = "experiment_runs.txt"
log_filepath = os.path.join(dir_results, log_filename)
```

[9]:
```
wandbai_api_key = ""
```

[10]:
```
!tree /content/drive/MyDrive/266-final/data/266-comp-lex-master/
```

```
/content/drive/MyDrive/266-final/data/266-comp-lex-master/
  fe-test-labels
      test_multi_df.csv
      test_single_df.csv
  fe-train
      train_multi_df.csv
      train_single_df.csv
  fe-trial-val
      trial_val_multi_df.csv
      trial_val_single_df.csv
  test-labels
      lcp_multi_test.tsv
      lcp_single_test.tsv
  train
      lcp_multi_train.tsv
      lcp_single_train.tsv
  trial
      lcp_multi_trial.tsv
      lcp_single_trial.tsv

6 directories, 12 files
```

[11]:

```
!ls -R /content/drive/MyDrive/266-final/data/266-comp-lex-master/
```

```
/content/drive/MyDrive/266-final/data/266-comp-lex-master/:
fe-test-labels  fe-train  fe-trial-val  test-labels  train  trial

/content/drive/MyDrive/266-final/data/266-comp-lex-master/fe-test-labels:
test_multi_df.csv  test_single_df.csv

/content/drive/MyDrive/266-final/data/266-comp-lex-master/fe-train:
train_multi_df.csv  train_single_df.csv

/content/drive/MyDrive/266-final/data/266-comp-lex-master/fe-trial-val:
trial_val_multi_df.csv  trial_val_single_df.csv

/content/drive/MyDrive/266-final/data/266-comp-lex-master/test-labels:
lcp_multi_test.tsv  lcp_single_test.tsv

/content/drive/MyDrive/266-final/data/266-comp-lex-master/train:
lcp_multi_train.tsv  lcp_single_train.tsv

/content/drive/MyDrive/266-final/data/266-comp-lex-master/trial:
lcp_multi_trial.tsv  lcp_single_trial.tsv
```

[12]:

```
!tree /content/drive/MyDrive/266-final/data/266-comp-lex-master/
```

```
/content/drive/MyDrive/266-final/data/266-comp-lex-master/
  fe-test-labels
     test_multi_df.csv
     test_single_df.csv
  fe-train
     train_multi_df.csv
     train_single_df.csv
  fe-trial-val
     trial_val_multi_df.csv
     trial_val_single_df.csv
  test-labels
     lcp_multi_test.tsv
     lcp_single_test.tsv
  train
     lcp_multi_train.tsv
     lcp_single_train.tsv
  trial
     lcp_multi_trial.tsv
     lcp_single_trial.tsv

6 directories, 12 files
```

[13]:

```
#@title Import Data
```

[14]:

```
df_names = [
    "train_single_df",
    "train_multi_df",
    "trial_val_single_df",
    "trial_val_multi_df",
    "test_single_df",
    "test_multi_df"
]

loaded_dataframes = {}

for df_name in df_names:
    if "train" in df_name:
        subdir = "fe-train"
    elif "trial_val" in df_name:
        subdir = "fe-trial-val"
    elif "test" in df_name:
        subdir = "fe-test-labels"
    else:
        subdir = None

    if subdir:
        read_path = os.path.join(dir_data, subdir, f"{df_name}.csv")
        loaded_df = pd.read_csv(read_path)
        loaded_dataframes[df_name] = loaded_df
        print(f"Loaded {df_name} from {read_path}")

# for df_name, df in loaded_dataframes.items():
#     print(f"\n>>> {df_name} shape: {df.shape}")
#     if 'binary_complexity' in df.columns:
#         print(df['binary_complexity'].value_counts())
#         print(df.info())
#         print(df.head())

for df_name, df in loaded_dataframes.items():
    globals()[df_name] = df
    print(f"{df_name} loaded into global namespace.")
```

```
Loaded train_single_df from /content/drive/MyDrive/266-final/data/266-comp-lex-
master/fe-train/train_single_df.csv
Loaded train_multi_df from /content/drive/MyDrive/266-final/data/266-comp-lex-
master/fe-train/train_multi_df.csv
Loaded trial_val_single_df from /content/drive/MyDrive/266-final/data/266-comp-
lex-master/fe-trial-val/trial_val_single_df.csv
Loaded trial_val_multi_df from /content/drive/MyDrive/266-final/data/266-comp-
lex-master/fe-trial-val/trial_val_multi_df.csv
Loaded test_single_df from /content/drive/MyDrive/266-final/data/266-comp-lex-
master/fe-test-labels/test_single_df.csv
```

```
Loaded test_multi_df from /content/drive/MyDrive/266-final/data/266-comp-lex-
master/fe-test-labels/test_multi_df.csv
train_single_df loaded into global namespace.
train_multi_df loaded into global namespace.
trial_val_single_df loaded into global namespace.
trial_val_multi_df loaded into global namespace.
test_single_df loaded into global namespace.
test_multi_df loaded into global namespace.
```

- Functional tests pass, we can proceed with Baseline Modeling

[15]:

```
#@title Experiment 1: Baseline Modeling
```

0.1.1 Reminders:

- Precision

$$\text{Precision} = \frac{TP}{TP + FP}$$

- Recall

$$\text{Recall} = \frac{TP}{TP + FN}$$

- Accuracy

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- F1 Score

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- Cosine Similarity

$$\text{Cosine Similarity} = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \, \|\mathbf{B}\|}$$

- Jaccard Similarity

$$\text{Jaccard Similarity} = \frac{|A \cap B|}{|A \cup B|}$$

- Overlap Similarity (Overlap Coefficient)

$$\text{Overlap Similarity} = \frac{|A \cap B|}{\min(|A|, |B|)}$$

- Dice Coefficient

$$\text{Dice Coefficient} = \frac{2\,|A \cap B|}{|A| + |B|}$$

0.2 Naive Bayes

0.2.1 $\mathbf{X}$ = Sentence: contractions and no contractions

- sentence no contractions

[16]:

```
train_df = train_single_df
val_df = trial_val_single_df

vectorizer = TfidfVectorizer()  # just on 'sentence_no_contractions'
X_train = vectorizer.fit_transform(train_df['sentence_no_contractions'])
y_train = train_df['binary_complexity']

X_val = vectorizer.transform(val_df['sentence_no_contractions'])
y_val = val_df['binary_complexity']

clf = MultinomialNB()
clf.fit(X_train, y_train)
preds = clf.predict(X_val)
print(classification_report(y_val, preds))
```

```
              precision    recall  f1-score   support

           0       0.57      0.74      0.64       518
           1       0.59      0.40      0.47       482

    accuracy                           0.57      1000
   macro avg       0.58      0.57      0.56      1000
weighted avg       0.58      0.57      0.56      1000
```

- sentence with contractions

[17]:

```
train_df = train_single_df
val_df = trial_val_single_df

vectorizer = TfidfVectorizer()  # just on 'sentence'
X_train = vectorizer.fit_transform(train_df['sentence'])
y_train = train_df['binary_complexity']

X_val = vectorizer.transform(val_df['sentence'])
y_val = val_df['binary_complexity']

clf = MultinomialNB()
clf.fit(X_train, y_train)
preds = clf.predict(X_val)
print(classification_report(y_val, preds))
```

```
              precision    recall  f1-score   support

           0       0.57      0.74      0.64       518
           1       0.59      0.40      0.48       482

    accuracy                           0.57      1000
   macro avg       0.58      0.57      0.56      1000
```

```
weighted avg       0.58      0.57      0.56      1000
```

- sentence no contractions

[18]:

```
train_df = train_multi_df
val_df = trial_val_multi_df

vectorizer = TfidfVectorizer()  # just on 'sentence_no_contractions'
X_train = vectorizer.fit_transform(train_df['sentence_no_contractions'])
y_train = train_df['binary_complexity']

X_val = vectorizer.transform(val_df['sentence_no_contractions'])
y_val = val_df['binary_complexity']

clf = MultinomialNB()
clf.fit(X_train, y_train)
preds = clf.predict(X_val)
print(classification_report(y_val, preds))
```

```
              precision    recall  f1-score   support

           0       0.66      0.75      0.70       142
           1       0.60      0.48      0.53       108

    accuracy                           0.64       250
   macro avg       0.63      0.62      0.62       250
weighted avg       0.63      0.64      0.63       250
```

- sentence with contractions

[19]:

```
train_df = train_multi_df
val_df = trial_val_multi_df

vectorizer = TfidfVectorizer()  # just on 'sentence'
X_train = vectorizer.fit_transform(train_df['sentence'])
y_train = train_df['binary_complexity']

X_val = vectorizer.transform(val_df['sentence'])
y_val = val_df['binary_complexity']

clf = MultinomialNB()
clf.fit(X_train, y_train)
preds = clf.predict(X_val)
print(classification_report(y_val, preds))
```

```
              precision    recall  f1-score   support

           0       0.66      0.75      0.70       142
           1       0.60      0.49      0.54       108

    accuracy                           0.64       250
   macro avg       0.63      0.62      0.62       250
weighted avg       0.64      0.64      0.63       250
```

- **Score is higher than expected for a Naive Bayes model**
- **There is no difference in performance when using the input sequence of the sentence with and without contractions**

0.2.2 X = pos_sequence: Part-of-Speech Tags

- POS Tags: Extracts the part-of-speech (POS) tags for each token (e.g., "DET", "NOUN", "VERB").

[20]:

```
train_df = train_single_df
val_df = trial_val_single_df

vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(train_df['pos_sequence'])
y_train = train_df['binary_complexity']

X_val = vectorizer.transform(val_df['pos_sequence'])
y_val = val_df['binary_complexity']

clf = MultinomialNB()
clf.fit(X_train, y_train)
preds = clf.predict(X_val)
print(classification_report(y_val, preds))
```

```
              precision    recall  f1-score   support

           0       0.54      0.69      0.61       518
           1       0.53      0.38      0.44       482

    accuracy                           0.54      1000
   macro avg       0.54      0.53      0.52      1000
weighted avg       0.54      0.54      0.53      1000
```

[21]:

```
train_df = train_multi_df
val_df = trial_val_multi_df

vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(train_df['pos_sequence'])
y_train = train_df['binary_complexity']

X_val = vectorizer.transform(val_df['pos_sequence'])
```

```
y_val = val_df['binary_complexity']

clf = MultinomialNB()
clf.fit(X_train, y_train)
preds = clf.predict(X_val)
print(classification_report(y_val, preds))
```

```
              precision    recall  f1-score   support

           0       0.62      0.73      0.67       142
           1       0.53      0.40      0.46       108

    accuracy                           0.59       250
   macro avg       0.57      0.57      0.56       250
weighted avg       0.58      0.59      0.58       250
```

- Part of Speech tags outperform raw input sequence

0.2.3 X = dep_sequence: Dependency Tags

- Dependency Tags: Extracts the syntactic dependency labels for each token (e.g., “det”, “nsubj”, “ROOT”).

[22]:

```
train_df = train_single_df
val_df = trial_val_single_df

vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(train_df['dep_sequence'])
y_train = train_df['binary_complexity']

X_val = vectorizer.transform(val_df['dep_sequence'])
y_val = val_df['binary_complexity']

clf = MultinomialNB()
clf.fit(X_train, y_train)
preds = clf.predict(X_val)
print(classification_report(y_val, preds))
```

```
              precision    recall  f1-score   support

           0       0.55      0.62      0.58       518
           1       0.53      0.46      0.49       482

    accuracy                           0.54      1000
   macro avg       0.54      0.54      0.54      1000
weighted avg       0.54      0.54      0.54      1000
```

[23]:

```
train_df = train_multi_df
val_df = trial_val_multi_df

vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(train_df['dep_sequence'])
y_train = train_df['binary_complexity']

X_val = vectorizer.transform(val_df['dep_sequence'])
y_val = val_df['binary_complexity']

clf = MultinomialNB()
clf.fit(X_train, y_train)
preds = clf.predict(X_val)
print(classification_report(y_val, preds))
```

```
              precision    recall  f1-score   support

           0       0.60      0.68      0.64       142
           1       0.49      0.42      0.45       108

    accuracy                           0.56       250
   macro avg       0.55      0.55      0.55       250
weighted avg       0.56      0.56      0.56       250
```

0.2.4 X = morph_sequence: Morphological Features

- For each token, the morphological attributes have been retrieved for each token

[24]:

```
train_df = train_single_df
val_df = trial_val_single_df

vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(train_df['morph_sequence'])
y_train = train_df['binary_complexity']

X_val = vectorizer.transform(val_df['morph_sequence'])
y_val = val_df['binary_complexity']

clf = MultinomialNB()
clf.fit(X_train, y_train)
preds = clf.predict(X_val)
print(classification_report(y_val, preds))
```

```
              precision    recall  f1-score   support

           0       0.56      0.59      0.57       518
           1       0.53      0.50      0.51       482
```

```
    accuracy                           0.55      1000
   macro avg       0.54      0.54      0.54      1000
weighted avg       0.54      0.55      0.54      1000
```

[25]:

```
train_df = train_multi_df
val_df = trial_val_multi_df

vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(train_df['morph_sequence'])
y_train = train_df['binary_complexity']

X_val = vectorizer.transform(val_df['morph_sequence'])
y_val = val_df['binary_complexity']

clf = MultinomialNB()
clf.fit(X_train, y_train)
preds = clf.predict(X_val)
print(classification_report(y_val, preds))
```

```
              precision    recall  f1-score   support

           0       0.66      0.56      0.61       142
           1       0.52      0.63      0.57       108

    accuracy                           0.59       250
   macro avg       0.59      0.59      0.59       250
weighted avg       0.60      0.59      0.59       250
```

0.2.5 Baseline Experiment Results

Evaluation

- Naive Bayes baseline demonstrates improved performance on the re-balanced dataset, compared with the original balance, including on pure engineered features.

0.3 Update: Concatenated and Interleaved Features

0.3.1 Single

[26]:

```
train_df = train_single_df
val_df = trial_val_single_df

vectorizer = TfidfVectorizer()  # just on 'sentence_no_contractions'
X_train = vectorizer.fit_transform(train_df['snc_pos_seq'])
y_train = train_df['binary_complexity']

X_val = vectorizer.transform(val_df['snc_pos_seq'])
```

```
y_val = val_df['binary_complexity']

clf = MultinomialNB()
clf.fit(X_train, y_train)
preds = clf.predict(X_val)
print(classification_report(y_val, preds))
```

```
              precision    recall  f1-score   support

           0       0.56      0.74      0.64       518
           1       0.57      0.37      0.45       482

    accuracy                           0.56      1000
   macro avg       0.56      0.55      0.54      1000
weighted avg       0.56      0.56      0.55      1000
```

[27]:

```
train_df = train_single_df
val_df = trial_val_single_df

vectorizer = TfidfVectorizer()  # just on 'sentence_no_contractions'
X_train = vectorizer.fit_transform(train_df['snc_pos_alt'])
y_train = train_df['binary_complexity']

X_val = vectorizer.transform(val_df['snc_pos_alt'])
y_val = val_df['binary_complexity']

clf = MultinomialNB()
clf.fit(X_train, y_train)
preds = clf.predict(X_val)
print(classification_report(y_val, preds))
```

```
              precision    recall  f1-score   support

           0       0.56      0.74      0.64       518
           1       0.57      0.37      0.45       482

    accuracy                           0.56      1000
   macro avg       0.56      0.56      0.54      1000
weighted avg       0.56      0.56      0.55      1000
```

[28]:

```
train_df = train_single_df
val_df = trial_val_single_df

vectorizer = TfidfVectorizer()  # just on 'sentence_no_contractions'
X_train = vectorizer.fit_transform(train_df['snc_morph_seq'])
y_train = train_df['binary_complexity']
```

```
X_val = vectorizer.transform(val_df['snc_morph_seq'])
y_val = val_df['binary_complexity']

clf = MultinomialNB()
clf.fit(X_train, y_train)
preds = clf.predict(X_val)
print(classification_report(y_val, preds))
```

```
              precision    recall  f1-score   support

           0       0.56      0.76      0.65       518
           1       0.59      0.37      0.45       482

    accuracy                           0.57      1000
   macro avg       0.58      0.56      0.55      1000
weighted avg       0.58      0.57      0.55      1000
```

[29]:

```
train_df = train_single_df
val_df = trial_val_single_df

vectorizer = TfidfVectorizer()  # just on 'sentence_no_contractions'
X_train = vectorizer.fit_transform(train_df['snc_morph_alt'])
y_train = train_df['binary_complexity']

X_val = vectorizer.transform(val_df['snc_morph_alt'])
y_val = val_df['binary_complexity']

clf = MultinomialNB()
clf.fit(X_train, y_train)
preds = clf.predict(X_val)
print(classification_report(y_val, preds))
```

```
              precision    recall  f1-score   support

           0       0.56      0.77      0.65       518
           1       0.59      0.36      0.45       482

    accuracy                           0.57      1000
   macro avg       0.58      0.56      0.55      1000
weighted avg       0.58      0.57      0.55      1000
```

[30]:

```
train_df = train_single_df
val_df = trial_val_single_df

vectorizer = TfidfVectorizer()  # just on 'sentence_no_contractions'
```

```
X_train = vectorizer.fit_transform(train_df['snc_dep_seq'])
y_train = train_df['binary_complexity']

X_val = vectorizer.transform(val_df['snc_dep_seq'])
y_val = val_df['binary_complexity']

clf = MultinomialNB()
clf.fit(X_train, y_train)
preds = clf.predict(X_val)
print(classification_report(y_val, preds))
```

```
              precision    recall  f1-score   support

           0       0.56      0.73      0.63       518
           1       0.57      0.37      0.45       482

    accuracy                           0.56      1000
   macro avg       0.56      0.55      0.54      1000
weighted avg       0.56      0.56      0.54      1000
```

[31]:

```
train_df = train_single_df
val_df = trial_val_single_df

vectorizer = TfidfVectorizer()  # just on 'sentence_no_contractions'
X_train = vectorizer.fit_transform(train_df['snc_dep_alt'])
y_train = train_df['binary_complexity']

X_val = vectorizer.transform(val_df['snc_dep_alt'])
y_val = val_df['binary_complexity']

clf = MultinomialNB()
clf.fit(X_train, y_train)
preds = clf.predict(X_val)
print(classification_report(y_val, preds))
```

```
              precision    recall  f1-score   support

           0       0.56      0.75      0.64       518
           1       0.58      0.37      0.45       482

    accuracy                           0.57      1000
   macro avg       0.57      0.56      0.55      1000
weighted avg       0.57      0.57      0.55      1000
```

[31]:

```

```

[31]:

```
```

0.3.2 Multi

[32]:

```
train_df = train_multi_df
val_df = trial_val_multi_df

vectorizer = TfidfVectorizer()  # just on 'sentence_no_contractions'
X_train = vectorizer.fit_transform(train_df['snc_pos_seq'])
y_train = train_df['binary_complexity']

X_val = vectorizer.transform(val_df['snc_pos_seq'])
y_val = val_df['binary_complexity']

clf = MultinomialNB()
clf.fit(X_train, y_train)
preds = clf.predict(X_val)
print(classification_report(y_val, preds))
```

```
              precision    recall  f1-score   support

           0       0.66      0.77      0.71       142
           1       0.62      0.49      0.55       108

    accuracy                           0.65       250
   macro avg       0.64      0.63      0.63       250
weighted avg       0.64      0.65      0.64       250
```

[33]:

```
train_df = train_multi_df
val_df = trial_val_multi_df

vectorizer = TfidfVectorizer()  # just on 'sentence_no_contractions'
X_train = vectorizer.fit_transform(train_df['snc_pos_alt'])
y_train = train_df['binary_complexity']

X_val = vectorizer.transform(val_df['snc_pos_alt'])
y_val = val_df['binary_complexity']

clf = MultinomialNB()
clf.fit(X_train, y_train)
preds = clf.predict(X_val)
print(classification_report(y_val, preds))
```

```
              precision    recall  f1-score   support

           0       0.65      0.77      0.70       142
           1       0.60      0.45      0.52       108
```

```
    accuracy                           0.63       250
   macro avg       0.62      0.61      0.61       250
weighted avg       0.63      0.63      0.62       250
```

[34]:

```
train_df = train_multi_df
val_df = trial_val_multi_df

vectorizer = TfidfVectorizer()  # just on 'sentence_no_contractions'
X_train = vectorizer.fit_transform(train_df['snc_morph_seq'])
y_train = train_df['binary_complexity']

X_val = vectorizer.transform(val_df['snc_morph_seq'])
y_val = val_df['binary_complexity']

clf = MultinomialNB()
clf.fit(X_train, y_train)
preds = clf.predict(X_val)
print(classification_report(y_val, preds))
```

```
              precision    recall  f1-score   support

           0       0.67      0.77      0.72       142
           1       0.63      0.51      0.56       108

    accuracy                           0.66       250
   macro avg       0.65      0.64      0.64       250
weighted avg       0.66      0.66      0.65       250
```

[35]:

```
train_df = train_multi_df
val_df = trial_val_multi_df

vectorizer = TfidfVectorizer()  # just on 'sentence_no_contractions'
X_train = vectorizer.fit_transform(train_df['snc_morph_alt'])
y_train = train_df['binary_complexity']

X_val = vectorizer.transform(val_df['snc_morph_alt'])
y_val = val_df['binary_complexity']

clf = MultinomialNB()
clf.fit(X_train, y_train)
preds = clf.predict(X_val)
print(classification_report(y_val, preds))
```

```
              precision    recall  f1-score   support

           0       0.66      0.78      0.72       142
           1       0.62      0.47      0.54       108

    accuracy                           0.65       250
   macro avg       0.64      0.63      0.63       250
weighted avg       0.64      0.65      0.64       250
```

[36]:

```
train_df = train_multi_df
val_df = trial_val_multi_df

vectorizer = TfidfVectorizer()  # just on 'sentence_no_contractions'
X_train = vectorizer.fit_transform(train_df['snc_dep_seq'])
y_train = train_df['binary_complexity']

X_val = vectorizer.transform(val_df['snc_dep_seq'])
y_val = val_df['binary_complexity']

clf = MultinomialNB()
clf.fit(X_train, y_train)
preds = clf.predict(X_val)
print(classification_report(y_val, preds))
```

```
              precision    recall  f1-score   support

           0       0.65      0.77      0.71       142
           1       0.60      0.46      0.52       108

    accuracy                           0.64       250
   macro avg       0.63      0.62      0.61       250
weighted avg       0.63      0.64      0.63       250
```

[37]:

```
train_df = train_multi_df
val_df = trial_val_multi_df

vectorizer = TfidfVectorizer()  # just on 'sentence_no_contractions'
X_train = vectorizer.fit_transform(train_df['snc_dep_alt'])
y_train = train_df['binary_complexity']

X_val = vectorizer.transform(val_df['snc_dep_alt'])
y_val = val_df['binary_complexity']

clf = MultinomialNB()
clf.fit(X_train, y_train)
preds = clf.predict(X_val)
print(classification_report(y_val, preds))
```

```
              precision    recall  f1-score   support
```

```
           0       0.64      0.77      0.70       142
           1       0.59      0.44      0.51       108

    accuracy                           0.63       250
   macro avg       0.62      0.61      0.60       250
weighted avg       0.62      0.63      0.62       250
```

[38]:

```
train_df = train_multi_df
val_df = trial_val_multi_df

vectorizer = TfidfVectorizer()  # just on 'sentence_no_contractions'
X_train = vectorizer.fit_transform(train_df['snc_morph_complexity_value'])
y_train = train_df['binary_complexity']

X_val = vectorizer.transform(val_df['snc_morph_complexity_value'])
y_val = val_df['binary_complexity']

clf = MultinomialNB()
clf.fit(X_train, y_train)
preds = clf.predict(X_val)
print(classification_report(y_val, preds))
```

```
              precision    recall  f1-score   support

           0       0.65      0.75      0.70       142
           1       0.59      0.47      0.53       108

    accuracy                           0.63       250
   macro avg       0.62      0.61      0.61       250
weighted avg       0.63      0.63      0.62       250
```

Evaluation

- Naive Bayes baseline demonstrates considerably improved performance on the re-balanced dataset, compared with the original balance, including on pure engineered features and enriched features (concatenated or interleaved).

[ ]:

```

```