3_0_Lexical_Complexity_Binary_Classification_Prediction_Baseline_Mode

April 7, 2025

[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
32.9 MB/s eta 0:00:00
                         116.3/116.3 kB
12.2 MB/s eta 0:00:00
                         183.9/183.9 kB
19.7 MB/s eta 0:00:00
                         143.5/143.5 kB
15.0 MB/s eta 0:00:00
                         194.8/194.8 kB
19.6 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
```

```
8.5 MB/s eta 0:00:00
                         289.9/289.9 kB
26.6 MB/s eta 0:00:00
                         118.3/118.3 kB
12.6 MB/s eta 0:00:00
```

[3]:
```
!sudo apt-get update
! sudo apt-get install tree
```

```
Get:1 https://cloud.r-project.org/bin/linux/ubuntu jammy-cran40/ InRelease
[3,632 B]
Get:2 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_64
InRelease [1,581 B]
Get:3 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_64
Packages [1,383 kB]
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]
Hit:7 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy InRelease
Get:8 https://r2u.stat.illinois.edu/ubuntu jammy InRelease [6,555 B]
Hit:9 https://ppa.launchpadcontent.net/graphics-drivers/ppa/ubuntu jammy
InRelease
Get:10 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages
[2,783 kB]
Hit:11 https://ppa.launchpadcontent.net/ubuntugis/ppa/ubuntu jammy InRelease
Get:12 https://r2u.stat.illinois.edu/ubuntu jammy/main all Packages [8,804 kB]
Get:13 http://archive.ubuntu.com/ubuntu jammy-backports InRelease [127 kB]
Get:14 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 Packages
[1,243 kB]
Get:15 http://security.ubuntu.com/ubuntu jammy-security/restricted amd64
Packages [3,994 kB]
Get:16 http://archive.ubuntu.com/ubuntu jammy-updates/restricted amd64 Packages
[4,154 kB]
Get:17 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [3,097
kB]
Get:18 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages
[1,540 kB]
Get:19 https://r2u.stat.illinois.edu/ubuntu jammy/main amd64 Packages [2,683 kB]
Fetched 30.1 MB in 4s (6,871 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 37 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 1s (55.0 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 … 126213 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]:

```
#@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/'
```

[33]:
```
wandbai_api_key = "5236444b7e96f5cf74038116d8c1efba161a4310"
```

[8]:
```
!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
```

[9]:
```
!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
```

[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]:

```
#@title Import Data
```

[12]:

```
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

[13]:
```
#@title Experiment 1: Baseline Modeling
```

0.0.1 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

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

0.1 Naive Bayes

0.1.1 X = Sentence: contractions and no contractions

- sentence no contractions

[14]:
```
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       0.58      0.74      0.65       229
           1       0.55      0.38      0.44       192

    accuracy                           0.57       421
   macro avg       0.57      0.56      0.55       421
weighted avg       0.57      0.57      0.56       421
```

- sentence with contractions

[15]:
```
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       0.58      0.74      0.65       229
              1       0.55      0.38      0.44       192

       accuracy                           0.57       421
      macro avg       0.57      0.56      0.55       421
   weighted avg       0.57      0.57      0.56       421
```

- sentence no contractions

[16]:
```
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       0.52      0.67      0.58        48
              1       0.57      0.41      0.48        51

       accuracy                           0.54        99
      macro avg       0.54      0.54      0.53        99
   weighted avg       0.54      0.54      0.53        99
```

- sentence with contractions

[17]:
```
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       0.52      0.67      0.58        48
           1       0.57      0.41      0.48        51

    accuracy                           0.54        99
   macro avg       0.54      0.54      0.53        99
weighted avg       0.54      0.54      0.53        99
```

- **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**

0.1.2 X = pos_sequence: Part-of-Speech Tags

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

[18]:
```
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       0.60      0.67      0.63       229
           1       0.54      0.46      0.50       192

    accuracy                           0.57       421
   macro avg       0.57      0.57      0.56       421
weighted avg       0.57      0.57      0.57       421
```

[19]:
```
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       0.58      0.54      0.56        48
           1       0.59      0.63      0.61        51

    accuracy                           0.59        99
   macro avg       0.59      0.58      0.58        99
weighted avg       0.59      0.59      0.59        99
```

- Part of Speech tags outperform raw input sequence

0.1.3 X = dep_sequence: Dependency Tags

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

[20]:
```
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       0.61      0.60      0.60       229
           1       0.53      0.54      0.54       192
```

```
    accuracy                           0.57       421
   macro avg       0.57      0.57      0.57       421
weighted avg       0.57      0.57      0.57       421
```

[21]:
```
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       0.51      0.46      0.48        48
           1       0.54      0.59      0.56        51

    accuracy                           0.53        99
   macro avg       0.52      0.52      0.52        99
weighted avg       0.52      0.53      0.52        99
```

0.1.4 X = morph_sequence: Morphological Features

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

[22]:
```
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       0.62      0.59      0.60       229
           1       0.53      0.57      0.55       192

    accuracy                           0.58       421
   macro avg       0.58      0.58      0.58       421
weighted avg       0.58      0.58      0.58       421
```

[23]:
```
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       0.62      0.52      0.57        48
           1       0.61      0.71      0.65        51

    accuracy                           0.62        99
   macro avg       0.62      0.61      0.61        99
weighted avg       0.62      0.62      0.61        99
```

0.1.5 Baseline Experiment Results

The table below summarizes the evaluation metrics for our Naive Bayes experiments. We report results for both sentence inputs (with and without contractions) as well as for the linguistic feature representations: Part-of-Speech tags (POS), Dependency tags, and Morphological features. Results are provided separately for the *Single* and *Multi* datasets. **Our Preferred Evaluation Metric of Interest is F1 Score**.

Input Type	Dataset	Accuracy	Precision	Recall	F1 Score
Sentence (with contractions)	Single	57%	57%	57%	57%

Input Type	Dataset	Accuracy	Precision	Recall	F1 Score
Sentence (without contractions)	Single	57%	57%	57%	57%
Sentence (with contractions)	Multi	54%	54%	54%	54%
Sentence (without contractions)	Multi	54%	54%	54%	54%
POS Tags (pos_sequence)	Single	57%	57%	57%	57%
POS Tags (pos_sequence)	Multi	59%	59%	59%	59%
Dependency Tags (dep_sequence)	Single	57%	57%	57%	57%
Dependency Tags (dep_sequence)	Multi	52%	52%	52%	52%
Morphological Features (morph_sequence)	Single	58%	58%	58%	58%
Morphological Features (morph_sequence)	Multi	62%	62%	62%	62%

Note: The metrics shown above are the weighted averages derived from Trial_Val.

Evaluation

- **Raw Sentence Input:** Both with and without contractions, the single-dataset experiment shows a macro F1-score of 0.57, while the multi-dataset experiment yields a lower F1-score (0.54). This suggests that for raw text, model performance degrades on the multi-label version. **While there is no contextual difference between in the contexts between the single and multi versions, the binary_complexity is different, as the complexity scores derived from the 'complex unigram and bigram tokens' in both the single and multi splits of the datasets achieved different scores, and thus different medians (from which we derived our binarized value).**
- **POS Tags:** Using part-of-speech tag sequences produces results similar to raw text on the single dataset (F1 = 0.57) and even slightly better performance on the multi dataset (F1 = 0.59).
- **Dependency Tags:** Dependency label sequences perform on par with the other features in the single-dataset setting (F1 = 0.57) but drop to an F1-score of 0.52 on the multi dataset, indicating less robustness for this representation in that setting.
- **Morphological Features:** On the single dataset, morphological features give a modest improvement (F1 = 0.58) over raw text. Notably, on the multi dataset, they yield the highest performance (F1 = 0.62), suggesting that despite there being no contextual difference between the two, Naive Bayes' capacity to split the complexity of the input sequence is more aligned with the median threshold of the multi-version split of the data. However, it should be noted that the multi-split for trial_val is literally only 99 records, so I expect that these performance metrics will drop substantially on the test set
- **Hyperparameter Tuning:** Naive Bayes was used in a fairly vanilla manner, not reflected in this notebook were some experiments done with varying alphas (i.e. Laplace Smoothing Values)—these led to effectively no difference in average F1 Score results.

Overall, these results indicate that while raw text and simple POS tags are competitive, the morphological feature representation provides an edge—especially in the multi dataset scenario. **This indicates keeping these additional features on-hand for transformers-based ablations may be a good call.**

0.2 Experiments with Transformers Models

[24]:
```
!ls /content/drive/MyDrive/266-final/models/
```

```
nltk_data
```

[25]:
```
# !mkdir /content/drive/MyDrive/266-final/results/bert-base-uncased/
!ls -R /content/drive/MyDrive/266-final/results/
```

```
/content/drive/MyDrive/266-final/results/:
bert-base-uncased                          sentence_span_analysis.csv
sentence_no_contraction_span_analysis.csv
sentence_span_analysis_no_contractions.csv

/content/drive/MyDrive/266-final/results/bert-base-uncased:
morph_single

/content/drive/MyDrive/266-final/results/bert-base-uncased/morph_single:
```

Helper Functions and Experiment Parameters

[26]:
```
# Convert Pandas DataFrame to a HF Dataset
def dataframe_to_hf_dataset(df, text_col, label_col="binary_complexity"):
    """
    Converts a Pandas DataFrame to a Hugging Face Dataset, keeping only:
      - text_col (e.g. 'sentence_no_contractions')
      - label_col (e.g. 'binary_complexity')
    Returns a HF Dataset with columns ["text", "label"].
    """
    subset_df = df[[text_col, label_col]].copy()
    subset_df.columns = ["text", "label"]
    hf_dataset = Dataset.from_pandas(subset_df)
    return hf_dataset

def prepare_dataset_for_trainer(df, text_col, tokenizer,␣
 ↪label_col="binary_complexity", max_length=128):
    """
    1) Converts the DataFrame to a HF Dataset (only text_col, label_col).
    2) Tokenizes using tokenizer (truncation, padding).
    3) Casts the dataset to PyTorch with columns ["input_ids",␣
 ↪"attention_mask", "label"].
    """
    hf_dataset = dataframe_to_hf_dataset(df, text_col, label_col=label_col)
```

```
    def tokenize_batch(examples):
        return tokenizer(
            examples["text"],
            truncation=True,
            padding="max_length",
            max_length=max_length
        )
    hf_dataset = hf_dataset.map(tokenize_batch, batched=True)
    hf_dataset.set_format(type="torch", columns=["input_ids", "attention_mask",␣
 ↪"label"])
    return hf_dataset
```

[27]:
```
# Compute Binary Classification Metrics
def compute_metrics(eval_pred):
    """
    Returns dict with accuracy, precision, recall, f1
    for binary classification.
    """
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)
    precision, recall, f1, _ = precision_recall_fscore_support(
        labels, predictions, average="binary", zero_division=0
    )
    acc = accuracy_score(labels, predictions)
    return {
        "accuracy": acc,
        "precision": precision,
        "recall": recall,
        "f1": f1
    }
```

[28]:
```
# Build Output Directory
def build_output_dir(base_dir, model_name, lr, epochs, batch_size, max_len,␣
 ↪x_task, x_col):
    """
    Creates a folder name that encodes the experimental setup, e.g.:
      /content/drive/MyDrive/266-final/results/
        [model_name]_[x_task]_[x_col]_lr1e-3_ep3_bs4_len348
    """
    # Clean up model_name for folder naming (remove any slashes)
    model_dir_name = model_name.replace("/", "-")

    folder_name =␣
 ↪f"{model_dir_name}_{x_task}_{x_col}_lr{lr}_ep{epochs}_bs{batch_size}_len{max_len}"
    out_path = os.path.join(base_dir, folder_name)
    return out_path
```

[29]:
```
# Experiment Runner
def run_transformer_experiment(
    train_df,
    val_df,
    test_df,
    text_col,
    model_name,
    lr,
    epochs,
    batch_size,
    max_len,
    y_col="binary_complexity",
    base_out_dir="/content/drive/MyDrive/266-final/results/"
):
    # Build output directory from hyperparams
    output_dir = build_output_dir(
        base_out_dir, model_name, lr, epochs, batch_size, max_len, x_task,␣
↪text_col
    )

    print(f"[INFO] Using output_dir: {output_dir}")

    # 1. Load tokenizer
    tokenizer = AutoTokenizer.from_pretrained(model_name)

    # 2. Prepare Datasets
    train_dataset = prepare_dataset_for_trainer(train_df, text_col, tokenizer,␣
↪label_col=y_col, max_length=max_len)
    val_dataset   = prepare_dataset_for_trainer(val_df,   text_col, tokenizer,␣
↪label_col=y_col, max_length=max_len)
    test_dataset  = prepare_dataset_for_trainer(test_df,  text_col, tokenizer,␣
↪label_col=y_col, max_length=max_len)

    # 3. Load model
    model = AutoModelForSequenceClassification.from_pretrained(model_name,␣
↪num_labels=2)

    # 4. TrainingArguments
    training_args = TrainingArguments(
        output_dir = output_dir,
        evaluation_strategy = "epoch",
        save_strategy = "epoch",
        learning_rate = lr,
        per_device_train_batch_size = batch_size,
        per_device_eval_batch_size = batch_size,
        num_train_epochs = epochs,
        weight_decay = 0,
```

```
        logging_dir = "./my_bert_logs",
        logging_steps = 500
    )

    # 5. Trainer
    trainer = Trainer(
        model = model,
        args = training_args,
        train_dataset = train_dataset,
        eval_dataset = val_dataset,
        tokenizer = tokenizer,
        compute_metrics = compute_metrics
    )

    # 6. Train
    trainer.train()

    # 7. Evaluate
    metrics_val = trainer.evaluate(eval_dataset=val_dataset)
    metrics_test = trainer.evaluate(eval_dataset=test_dataset)

    print(f"\n===== RESULTS for text_col='{text_col}' =====")
    print("[Validation]:", metrics_val)
    print("[Test]:", metrics_test)

    return trainer, (metrics_val, metrics_test)
```

0.2.1 Experiment Configuration

[41]:

```
# Define Experiment Parameters

named_model = "bert-base-cased"
# named_model = "roberta-base"
# named_model = "bert-large"
# named_model = "roberta-large"
# named_model = ""

# learning_rate = 1e-3
learning_rate = 1e-4
# learning_rate = 1e-5
# learning_rate = 5e-6

# num_epochs = 3
num_epochs = 5
# num_epochs = 10
# num_epochs = 15
# num_epochs = 20
```

```python
length_max = 348
# length_max = 512

# size_batch = 1
# size_batch = 4
# size_batch = 8
# size_batch = 16
# size_batch = 24
size_batch = 32

y_col = "binary_complexity"
# y_col = "complexity"

x_task = "single"
# x_task = "multi"

# x_col = "sentence"
# x_col = "sentence_no_contractions"
# x_col = "pos_sequence"
# x_col = "dep_sequence"
x_col = "morph_sequence"

if x_task == "single":
    df_train = train_single_df
    df_val   = trial_val_single_df
    df_test  = test_single_df
else:
    df_train = train_multi_df
    df_val   = trial_val_multi_df
    df_test  = test_multi_df
```

[38]:

```python
trainer_obj, (val_metrics, test_metrics) = run_transformer_experiment(
    train_df = df_train,
    val_df = df_val,
    test_df = df_test,
    text_col = x_col,
    model_name = named_model,
    lr = learning_rate,
    epochs = num_epochs,
    batch_size = size_batch,
    max_len = length_max,
    y_col = y_col,
    base_out_dir = "/content/drive/MyDrive/266-final/results/"
)

print("\nFinal Validation Metrics:", val_metrics)
```

```python
print("Final Test Metrics:", test_metrics)
```

```
[INFO] Using output_dir: /content/drive/MyDrive/266-final/results/bert-base-
uncased_single_morph_sequence_lr0.0001_ep5_bs32_len348

Map:   0%|          | 0/7662 [00:00<?, ? examples/s]

Map:   0%|          | 0/421 [00:00<?, ? examples/s]

Map:   0%|          | 0/917 [00:00<?, ? examples/s]

Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at bert-base-uncased and are newly initialized:
['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
/usr/local/lib/python3.11/dist-packages/transformers/training_args.py:1611:
FutureWarning: `evaluation_strategy` is deprecated and will be removed in
version 4.46 of   Transformers. Use `eval_strategy` instead
  warnings.warn(
<ipython-input-29-fc1f37fce11c>:48: FutureWarning: `tokenizer` is deprecated and
will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class`
instead.
  trainer = Trainer(

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>


===== RESULTS for text_col='morph_sequence' =====
[Validation]: {'eval_loss': 0.6926975846290588, 'eval_accuracy':
0.5439429928741093, 'eval_precision': 0.0, 'eval_recall': 0.0, 'eval_f1': 0.0,
'eval_runtime': 1.7336, 'eval_samples_per_second': 242.853,
'eval_steps_per_second': 8.076, 'epoch': 5.0}
[Test]: {'eval_loss': 0.692959725856781, 'eval_accuracy': 0.5190839694656488,
'eval_precision': 0.0, 'eval_recall': 0.0, 'eval_f1': 0.0, 'eval_runtime':
3.6839, 'eval_samples_per_second': 248.919, 'eval_steps_per_second': 7.872,
'epoch': 5.0}

Final Validation Metrics: {'eval_loss': 0.6926975846290588, 'eval_accuracy':
0.5439429928741093, 'eval_precision': 0.0, 'eval_recall': 0.0, 'eval_f1': 0.0,
'eval_runtime': 1.7336, 'eval_samples_per_second': 242.853,
'eval_steps_per_second': 8.076, 'epoch': 5.0}
Final Test Metrics: {'eval_loss': 0.692959725856781, 'eval_accuracy':
0.5190839694656488, 'eval_precision': 0.0, 'eval_recall': 0.0, 'eval_f1': 0.0,
'eval_runtime': 3.6839, 'eval_samples_per_second': 248.919,
'eval_steps_per_second': 7.872, 'epoch': 5.0}
```

[32]:
```
!tree /content/drive/MyDrive/266-final/results/
```

```
/content/drive/MyDrive/266-final/results/
  bert-base-uncased
     morph_single
  bert-base-
uncased_multi_sentence_no_contractions_lr0.001_ep3_bs4_len348
     checkpoint-1140
        config.json
        model.safetensors
        optimizer.pt
        rng_state.pth
        scheduler.pt
        special_tokens_map.json
        tokenizer_config.json
        tokenizer.json
        trainer_state.json
        training_args.bin
        vocab.txt
     checkpoint-380
        config.json
        model.safetensors
        optimizer.pt
        rng_state.pth
        scheduler.pt
        special_tokens_map.json
        tokenizer_config.json
        tokenizer.json
        trainer_state.json
        training_args.bin
        vocab.txt
     checkpoint-760
        config.json
        model.safetensors
        optimizer.pt
        rng_state.pth
        scheduler.pt
        special_tokens_map.json
        tokenizer_config.json
        tokenizer.json
        trainer_state.json
        training_args.bin
        vocab.txt
  sentence_no_contraction_span_analysis.csv
  sentence_span_analysis.csv
  sentence_span_analysis_no_contractions.csv

6 directories, 36 files
```

0.2.2 BERT base uncased

[34]:
```
# # 1. DataFrame to HF Dataset Conversion
# def dataframe_to_hf_dataset(df, text_col, label_col="binary_complexity"):
#     """
#     Converts a Pandas DataFrame to a Hugging Face Dataset, keeping only:
#       - text_col: Name of column containing input text (e.g. 'sentence')
#       - label_col: Name of column containing the label (default␣
↪'binary_complexity')
#     The resulting columns in the HF Dataset will be ["text", "label"].
#     """
#     # Subset the DF
#     subset_df = df[[text_col, label_col]].copy()
#     # Rename columns for clarity
#     subset_df.columns = ["text", "label"]
#     # Build and return HF Dataset
#     hf_dataset = Dataset.from_pandas(subset_df)
#     return hf_dataset

# def prepare_dataset_for_trainer(df, text_col, tokenizer,␣
↪label_col="binary_complexity", max_length=128):
#     """
#     1. Converts the DataFrame to a HF Dataset (only text_col and label_col).
#     2. Tokenizes using the provided tokenizer (truncates/pads to max_length).
#     3. Casts the dataset to PyTorch format, ensuring columns are␣
↪["input_ids", "attention_mask", "label"].
#     4. Returns the prepared dataset.
#     """
#     # Create initial dataset
#     hf_dataset = dataframe_to_hf_dataset(df, text_col, label_col=label_col)

#     # Define our tokenize function for the dataset.map
#     def tokenize_batch(examples):
#         return tokenizer(
#             examples["text"],
#             truncation=True,
#             padding="max_length",
#             max_length=max_length
#         )

#     # Tokenize entire dataset (batched for speed)
#     hf_dataset = hf_dataset.map(tokenize_batch, batched=True)

#     # Set the dataset format so PyTorch can read "input_ids",␣
↪"attention_mask", "label"
#     hf_dataset.set_format(type="torch", columns=["input_ids",␣
↪"attention_mask", "label"])
```

```
#     return hf_dataset

# # 2. Compute Metrics
# def compute_metrics(eval_pred):
#     logits, labels = eval_pred
#     predictions = np.argmax(logits, axis=-1)
#     precision, recall, f1, _ = precision_recall_fscore_support(
#         labels, predictions, average="binary", zero_division=0
#     )
#     acc = accuracy_score(labels, predictions)
#     return {
#         "accuracy": acc,
#         "precision": precision,
#         "recall": recall,
#         "f1": f1
#     }

# # 3. Training
# def run_bert_experiment(
#     train_df,
#     val_df,
#     test_df,
#     text_col,
#     model_name="bert-base-uncased",
#     output_dir="/content/drive/MyDrive/266-final/results/bert-base-uncased/",
#     epochs=num_epochs,
#     lr=learning_rate,
#     batch_size=size_batch,
#     max_length=length_max
# ):
#     """
#     A compartmentalized function to:
#       1. Prepare DataFrames (train_df, val_df, test_df) for BERT-based␣
↪classification
#          (binary_complexity as label, text_col as input text).
#       2. Instantiate a Transformer model with `num_labels=2`.
#       3. Configure and run a Hugging Face Trainer.
#       4. Evaluate on both the validation and test sets.
#     Returns:
#       trainer (Trainer) object,
#       (metrics_val, metrics_test) tuple containing evaluation metrics on val/
↪test.
#     """
#     # 1. Load tokenizer
#     tokenizer = AutoTokenizer.from_pretrained(model_name)

#     # 2. Convert DataFrames -> tokenized HF Datasets
```

```
#     train_dataset = prepare_dataset_for_trainer(train_df, text_col,␣
↪tokenizer, max_length=max_length)
#     val_dataset   = prepare_dataset_for_trainer(val_df,   text_col,␣
↪tokenizer, max_length=max_length)
#     test_dataset  = prepare_dataset_for_trainer(test_df,  text_col,␣
↪tokenizer, max_length=max_length)

#     # 3. Load model (AutoModelForSequenceClassification) with binary␣
↪classification
#     model = AutoModelForSequenceClassification.from_pretrained(model_name,␣
↪num_labels=2)

#     # 4. Define training arguments
#     training_args = TrainingArguments(
#         output_dir=output_dir,
#         evaluation_strategy="epoch",
#         save_strategy="epoch",
#         learning_rate=lr,
#         per_device_train_batch_size=batch_size,
#         per_device_eval_batch_size=batch_size,
#         num_train_epochs=epochs,
#         weight_decay=0,
#         logging_dir="./my_bert_logs",  # local logs
#         logging_steps=500
#     )

#     # 5. Build the Trainer
#     trainer = Trainer(
#         model=model,
#         args=training_args,
#         train_dataset=train_dataset,
#         eval_dataset=val_dataset,
#         tokenizer=tokenizer,
#         compute_metrics=compute_metrics  # optional
#     )

#     # 6. Train
#     trainer.train()

#     # 7. Evaluate on val & test
#     metrics_val = trainer.evaluate(eval_dataset=val_dataset)
#     metrics_test = trainer.evaluate(eval_dataset=test_dataset)

#     print(f"\n----- RESULTS for text_col='{text_col}' -----")
#     print("Validation:", metrics_val)
#     print("Test:", metrics_test)
```

```
#     return trainer, (metrics_val, metrics_test)
```

[35]:
```
# trainer_single_morph, (val_metrics_single_morph, test_metrics_single_morph) =
↪run_bert_experiment(
#     train_df=train_single_df,
#     val_df=trial_val_single_df,
#     test_df=test_single_df,
#     text_col="sentence_no_contractions", # "pos_sequence", "dep_sequence",
↪"morph_sequence",
#     model_name="bert-base-uncased",
#     output_dir="/content/drive/MyDrive/266-final/results/bert-base-uncased/
↪morph_single",
#     epochs=3,
#     lr=1e-4, #5e-6 1e-5 1e-4 1e-3
#     batch_size=8,
#     max_length=348
# )
```

```
Map:   0%|          | 0/7662 [00:00<?, ? examples/s]

Map:   0%|          | 0/421 [00:00<?, ? examples/s]

Map:   0%|          | 0/917 [00:00<?, ? examples/s]

Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at bert-base-uncased and are newly initialized:
['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
/usr/local/lib/python3.11/dist-packages/transformers/training_args.py:1611:
FutureWarning: `evaluation_strategy` is deprecated and will be removed in
version 4.46 of  Transformers. Use `eval_strategy` instead
  warnings.warn(
<ipython-input-34-81f1dfcc869e>:108: FutureWarning: `tokenizer` is deprecated
and will be removed in version 5.0.0 for `Trainer.__init__`. Use
`processing_class` instead.
  trainer = Trainer(

<IPython.core.display.HTML object>
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-35-5c5394ef8c5c> in <cell line: 0>()
----> 1 trainer_single_morph, (val_metrics_single_morph,
↪test_metrics_single_morph) = run_bert_experiment(
      2     train_df=train_single_df,
      3     val_df=trial_val_single_df,
      4     test_df=test_single_df,
```

```
      5     text_col="sentence_no_contractions", # "pos_sequence",␣
 ↪"dep_sequence", "morph_sequence",

<ipython-input-34-81f1dfcc869e> in run_bert_experiment(train_df, val_df,␣
 ↪test_df, text_col, model_name, output_dir, epochs, lr, batch_size, max_length
    116
    117     # 6. Train
--> 118     trainer.train()
    119
    120     # 7. Evaluate on val & test

/usr/local/lib/python3.11/dist-packages/transformers/trainer.py in train(self,␣
 ↪resume_from_checkpoint, trial, ignore_keys_for_eval, **kwargs)
   2243                 hf_hub_utils.enable_progress_bars()
   2244         else:
-> 2245             return inner_training_loop(

   2246                 args=args,
   2247                 resume_from_checkpoint=resume_from_checkpoint,

/usr/local/lib/python3.11/dist-packages/transformers/trainer.py in␣
 ↪_inner_training_loop(self, batch_size, args, resume_from_checkpoint, trial,␣
 ↪ignore_keys_for_eval)
   2559                         args.logging_nan_inf_filter
   2560                         and not is_torch_xla_available()
-> 2561                         and (torch.isnan(tr_loss_step) or torch.
 ↪isinf(tr_loss_step))
   2562                     ):
   2563                         # if loss is nan or inf simply add the average␣
 ↪of previous logged losses

KeyboardInterrupt:
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

0.3 BERT

[ ]:
```

```