Install and import necessary libraries

```
!pip install numpy pandas gensim scikit-learn torch torchvision
!pip install contractions
!pip install ipython-autotime
!pip install fastparquet
!pip install bs4
```

```
Requirement already satisfied: numpy in
/usr/local/lib/python3.11/dist-packages (1.26.4)
Requirement already satisfied: pandas in
/usr/local/lib/python3.11/dist-packages (2.2.2)
Requirement already satisfied: gensim in
/usr/local/lib/python3.11/dist-packages (4.3.3)
Requirement already satisfied: scikit-learn in
/usr/local/lib/python3.11/dist-packages (1.6.1)
Requirement already satisfied: torch in
/usr/local/lib/python3.11/dist-packages (2.5.1+cu124)
Requirement already satisfied: torchvision in
/usr/local/lib/python3.11/dist-packages (0.20.1+cu124)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.11/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.11/dist-packages (from pandas) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.11/dist-packages (from pandas) (2025.1)
Requirement already satisfied: scipy<1.14.0,>=1.7.0 in
/usr/local/lib/python3.11/dist-packages (from gensim) (1.13.1)
Requirement already satisfied: smart-open>=1.8.1 in
/usr/local/lib/python3.11/dist-packages (from gensim) (7.1.0)
Requirement already satisfied: joblib>=1.2.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: filelock in
/usr/local/lib/python3.11/dist-packages (from torch) (3.17.0)
Requirement already satisfied: typing-extensions>=4.8.0 in
/usr/local/lib/python3.11/dist-packages (from torch) (4.12.2)
Requirement already satisfied: networkx in
/usr/local/lib/python3.11/dist-packages (from torch) (3.4.2)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.11/dist-packages (from torch) (3.1.5)
Requirement already satisfied: fsspec in
/usr/local/lib/python3.11/dist-packages (from torch) (2024.10.0)
Collecting nvidia-cuda-nvrtc-cu12==12.4.127 (from torch)
  Downloading nvidia_cuda_nvrtc_cu12-12.4.127-py3-none-
manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cuda-runtime-cu12==12.4.127 (from torch)
  Downloading nvidia_cuda_runtime_cu12-12.4.127-py3-none-
```

```
manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cuda-cupti-cu12==12.4.127 (from torch)
  Downloading nvidia_cuda_cupti_cu12-12.4.127-py3-none-
manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cudnn-cu12==9.1.0.70 (from torch)
  Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-
manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cublas-cu12==12.4.5.8 (from torch)
  Downloading nvidia_cublas_cu12-12.4.5.8-py3-none-
manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cufft-cu12==11.2.1.3 (from torch)
  Downloading nvidia_cufft_cu12-11.2.1.3-py3-none-
manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-curand-cu12==10.3.5.147 (from torch)
  Downloading nvidia_curand_cu12-10.3.5.147-py3-none-
manylinux2014_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cusolver-cu12==11.6.1.9 (from torch)
  Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-
manylinux2014_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cusparse-cu12==12.3.1.170 (from torch)
  Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-
manylinux2014_x86_64.whl.metadata (1.6 kB)
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in
/usr/local/lib/python3.11/dist-packages (from torch) (2.21.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in
/usr/local/lib/python3.11/dist-packages (from torch) (12.4.127)
Collecting nvidia-nvjitlink-cu12==12.4.127 (from torch)
  Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-
manylinux2014_x86_64.whl.metadata (1.5 kB)
Requirement already satisfied: triton==3.1.0 in
/usr/local/lib/python3.11/dist-packages (from torch) (3.1.0)
Requirement already satisfied: sympy==1.13.1 in
/usr/local/lib/python3.11/dist-packages (from torch) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/usr/local/lib/python3.11/dist-packages (from sympy==1.13.1->torch)
(1.3.0)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in
/usr/local/lib/python3.11/dist-packages (from torchvision) (11.1.0)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2-
>pandas) (1.17.0)
Requirement already satisfied: wrapt in
/usr/local/lib/python3.11/dist-packages (from smart-open>=1.8.1-
>gensim) (1.17.2)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.11/dist-packages (from jinja2->torch) (3.0.2)
Downloading nvidia_cublas_cu12-12.4.5.8-py3-none-
manylinux2014_x86_64.whl (363.4 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.4/363.4 MB 2.9 MB/s eta
```

```
0:00:00
anylinux2014_x86_64.whl (13.8 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.8/13.8 MB 110.9 MB/s eta
0:00:00
anylinux2014_x86_64.whl (24.6 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 88.1 MB/s eta
0:00:00
e_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (883 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 53.4 MB/s eta
0:00:00
anylinux2014_x86_64.whl (664.8 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 1.6 MB/s eta
0:00:00
anylinux2014_x86_64.whl (211.5 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 10.8 MB/s eta
0:00:00
anylinux2014_x86_64.whl (56.3 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 41.2 MB/s eta
0:00:00
anylinux2014_x86_64.whl (127.9 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 19.2 MB/s eta
0:00:00
anylinux2014_x86_64.whl (207.5 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 4.2 MB/s eta
0:00:00
anylinux2014_x86_64.whl (21.1 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 82.8 MB/s eta
0:00:00
e-cu12, nvidia-cuda-nvrtc-cu12, nvidia-cuda-cupti-cu12, nvidia-cublas-
cu12, nvidia-cusparse-cu12, nvidia-cudnn-cu12, nvidia-cusolver-cu12
  Attempting uninstall: nvidia-nvjitlink-cu12
    Found existing installation: nvidia-nvjitlink-cu12 12.5.82
    Uninstalling nvidia-nvjitlink-cu12-12.5.82:
      Successfully uninstalled nvidia-nvjitlink-cu12-12.5.82
  Attempting uninstall: nvidia-curand-cu12
    Found existing installation: nvidia-curand-cu12 10.3.6.82
    Uninstalling nvidia-curand-cu12-10.3.6.82:
      Successfully uninstalled nvidia-curand-cu12-10.3.6.82
  Attempting uninstall: nvidia-cufft-cu12
    Found existing installation: nvidia-cufft-cu12 11.2.3.61
    Uninstalling nvidia-cufft-cu12-11.2.3.61:
      Successfully uninstalled nvidia-cufft-cu12-11.2.3.61
  Attempting uninstall: nvidia-cuda-runtime-cu12
    Found existing installation: nvidia-cuda-runtime-cu12 12.5.82
    Uninstalling nvidia-cuda-runtime-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-runtime-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-nvrtc-cu12
    Found existing installation: nvidia-cuda-nvrtc-cu12 12.5.82
    Uninstalling nvidia-cuda-nvrtc-cu12-12.5.82:
```

```
      Successfully uninstalled nvidia-cuda-nvrtc-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-cupti-cu12
    Found existing installation: nvidia-cuda-cupti-cu12 12.5.82
    Uninstalling nvidia-cuda-cupti-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-cupti-cu12-12.5.82
  Attempting uninstall: nvidia-cublas-cu12
    Found existing installation: nvidia-cublas-cu12 12.5.3.2
    Uninstalling nvidia-cublas-cu12-12.5.3.2:
      Successfully uninstalled nvidia-cublas-cu12-12.5.3.2
  Attempting uninstall: nvidia-cusparse-cu12
    Found existing installation: nvidia-cusparse-cu12 12.5.1.3
    Uninstalling nvidia-cusparse-cu12-12.5.1.3:
      Successfully uninstalled nvidia-cusparse-cu12-12.5.1.3
  Attempting uninstall: nvidia-cudnn-cu12
    Found existing installation: nvidia-cudnn-cu12 9.3.0.75
    Uninstalling nvidia-cudnn-cu12-9.3.0.75:
      Successfully uninstalled nvidia-cudnn-cu12-9.3.0.75
  Attempting uninstall: nvidia-cusolver-cu12
    Found existing installation: nvidia-cusolver-cu12 11.6.3.83
    Uninstalling nvidia-cusolver-cu12-11.6.3.83:
      Successfully uninstalled nvidia-cusolver-cu12-11.6.3.83
Successfully installed nvidia-cublas-cu12-12.4.5.8 nvidia-cuda-cupti-
cu12-12.4.127 nvidia-cuda-nvrtc-cu12-12.4.127 nvidia-cuda-runtime-
cu12-12.4.127 nvidia-cudnn-cu12-9.1.0.70 nvidia-cufft-cu12-11.2.1.3
nvidia-curand-cu12-10.3.5.147 nvidia-cusolver-cu12-11.6.1.9 nvidia-
cusparse-cu12-12.3.1.170 nvidia-nvjitlink-cu12-12.4.127
Collecting contractions
  Downloading contractions-0.1.73-py2.py3-none-any.whl.metadata (1.2
kB)
Collecting textsearch>=0.0.21 (from contractions)
  Downloading textsearch-0.0.24-py2.py3-none-any.whl.metadata (1.2 kB)
Collecting anyascii (from textsearch>=0.0.21->contractions)
  Downloading anyascii-0.3.2-py3-none-any.whl.metadata (1.5 kB)
Collecting pyahocorasick (from textsearch>=0.0.21->contractions)
  Downloading pyahocorasick-2.1.0-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (13 kB)
Downloading contractions-0.1.73-py2.py3-none-any.whl (8.7 kB)
Downloading textsearch-0.0.24-py2.py3-none-any.whl (7.6 kB)
Downloading anyascii-0.3.2-py3-none-any.whl (289 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 289.9/289.9 kB 5.0 MB/s eta
0:00:00
anylinux_2_17_x86_64.manylinux2014_x86_64.whl (118 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 118.3/118.3 kB 13.4 MB/s eta
0:00:00
e
  Downloading ipython_autotime-0.3.2-py2.py3-none-any.whl.metadata
(1.4 kB)
Requirement already satisfied: ipython in
/usr/local/lib/python3.11/dist-packages (from ipython-autotime)
```

```
(7.34.0)
Requirement already satisfied: setuptools>=18.5 in
/usr/local/lib/python3.11/dist-packages (from ipython->ipython-
autotime) (75.1.0)
Collecting jedi>=0.16 (from ipython->ipython-autotime)
  Downloading jedi-0.19.2-py2.py3-none-any.whl.metadata (22 kB)
Requirement already satisfied: decorator in
/usr/local/lib/python3.11/dist-packages (from ipython->ipython-
autotime) (4.4.2)
Requirement already satisfied: pickleshare in
/usr/local/lib/python3.11/dist-packages (from ipython->ipython-
autotime) (0.7.5)
Requirement already satisfied: traitlets>=4.2 in
/usr/local/lib/python3.11/dist-packages (from ipython->ipython-
autotime) (5.7.1)
Requirement already satisfied: prompt-toolkit!=3.0.0,!
=3.0.1,<3.1.0,>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from
ipython->ipython-autotime) (3.0.50)
Requirement already satisfied: pygments in
/usr/local/lib/python3.11/dist-packages (from ipython->ipython-
autotime) (2.18.0)
Requirement already satisfied: backcall in
/usr/local/lib/python3.11/dist-packages (from ipython->ipython-
autotime) (0.2.0)
Requirement already satisfied: matplotlib-inline in
/usr/local/lib/python3.11/dist-packages (from ipython->ipython-
autotime) (0.1.7)
Requirement already satisfied: pexpect>4.3 in
/usr/local/lib/python3.11/dist-packages (from ipython->ipython-
autotime) (4.9.0)
Requirement already satisfied: parso<0.9.0,>=0.8.4 in
/usr/local/lib/python3.11/dist-packages (from jedi>=0.16->ipython-
>ipython-autotime) (0.8.4)
Requirement already satisfied: ptyprocess>=0.5 in
/usr/local/lib/python3.11/dist-packages (from pexpect>4.3->ipython-
>ipython-autotime) (0.7.0)
Requirement already satisfied: wcwidth in
/usr/local/lib/python3.11/dist-packages (from prompt-toolkit!=3.0.0,!
=3.0.1,<3.1.0,>=2.0.0->ipython->ipython-autotime) (0.2.13)
Downloading ipython_autotime-0.3.2-py2.py3-none-any.whl (7.0 kB)
Downloading jedi-0.19.2-py2.py3-none-any.whl (1.6 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 21.6 MB/s eta
0:00:00
e
Successfully installed ipython-autotime-0.3.2 jedi-0.19.2
Collecting fastparquet
  Downloading fastparquet-2024.11.0-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (4.2 kB)
Requirement already satisfied: pandas>=1.5.0 in
```

```
/usr/local/lib/python3.11/dist-packages (from fastparquet) (2.2.2)
Requirement already satisfied: numpy in
/usr/local/lib/python3.11/dist-packages (from fastparquet) (1.26.4)
Requirement already satisfied: cramjam>=2.3 in
/usr/local/lib/python3.11/dist-packages (from fastparquet) (2.9.1)
Requirement already satisfied: fsspec in
/usr/local/lib/python3.11/dist-packages (from fastparquet) (2024.10.0)
Requirement already satisfied: packaging in
/usr/local/lib/python3.11/dist-packages (from fastparquet) (24.2)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.11/dist-packages (from pandas>=1.5.0-
>fastparquet) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.11/dist-packages (from pandas>=1.5.0-
>fastparquet) (2025.1)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.11/dist-packages (from pandas>=1.5.0-
>fastparquet) (2025.1)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2-
>pandas>=1.5.0->fastparquet) (1.17.0)
Downloading fastparquet-2024.11.0-cp311-cp311-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.8 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 21.4 MB/s eta
0:00:00
etadata (411 bytes)
Requirement already satisfied: beautifulsoup4 in
/usr/local/lib/python3.11/dist-packages (from bs4) (4.13.3)
Requirement already satisfied: soupsieve>1.2 in
/usr/local/lib/python3.11/dist-packages (from beautifulsoup4->bs4)
(2.6)
Requirement already satisfied: typing-extensions>=4.0.0 in
/usr/local/lib/python3.11/dist-packages (from beautifulsoup4->bs4)
(4.12.2)
Downloading bs4-0.0.2-py2.py3-none-any.whl (1.2 kB)
Installing collected packages: bs4
Successfully installed bs4-0.0.2
```

```python
import numpy as np
import pandas as pd
import gensim
import torch
import torch.nn as nn
import torch.optim as optim
import requests
import os
import re
import shutil
import urllib.request
import unicodedata
```

```
import multiprocessing
import warnings
import nltk
import contractions
import gensim.downloader as api
warnings.filterwarnings("ignore")

from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.linear_model import Perceptron
from gensim.models import Word2Vec, KeyedVectors
from nltk.corpus import stopwords, wordnet
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from gensim.models import Word2Vec
from sklearn.metrics import precision_score, recall_score, f1_score,
accuracy_score
from sklearn.svm import LinearSVC
from torch.utils.data.sampler import RandomSampler, BatchSampler
from torch.utils.data import Dataset, DataLoader
from tqdm.notebook import tqdm
from bs4 import BeautifulSoup

nltk.download('punkt', quiet=True)
nltk.download('wordnet', quiet=True)
nltk.download('stopwords', quiet=True)
nltk.download('averaged_perceptron_tagger', quiet=True)
nltk.download('punkt_tab')

%load_ext autotime
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.

time: 278 µs (started: 2025-02-14 00:15:17 +00:00)
```

Set config Values

```
CURRENT_DIR = os.getcwd()

class ConfigValues:
    RANDOM_STATE_VALUE = 42
    MAX_TFIDF_FEATURES = 45000
    TEST_SPLIT = 0.2
    N_SAMPLES_EACH_CLASS = 50000
    DATA_PATH = os.path.join(
        CURRENT_DIR, "amazon_reviews_us_Office_Products_v1_00.tsv.gz"
    )
    PARQUET_PATH = os.path.join(CURRENT_DIR,
"amazon_reviews_og.parquet")
```

```
    URL =
"https://web.archive.org/web/20201127142707if_/https://s3.amazonaws.co
m/amazon-reviews-pds/tsv/
amazon_reviews_us_Office_Products_v1_00.tsv.gz"

class W2VConfigValues:
    GOOGLE_PRETRAINED_MODEL = "word2vec-google-news-300"
    # TO DO: change variables
    PRETRAINED_OG_PATH = os.path.join(
        gensim.downloader.BASE_DIR, GOOGLE_PRETRAINED_MODEL,
f"{GOOGLE_PRETRAINED_MODEL}.gz"
    )
    PRETRAINED_SAVED_PATH = os.path.join(
        CURRENT_DIR, GOOGLE_PRETRAINED_MODEL,
f"{GOOGLE_PRETRAINED_MODEL}.gz"
    )
    WINDOW_SIZE = 11
    MAX_LENGTH = 300
    EMBEDDING_SIZE = 300
    MIN_WORD_COUNT = 10
    CUSTOM_MODEL_PATH = os.path.join(CURRENT_DIR, "word2vec-
custom.model")

time: 708 µs (started: 2025-02-14 00:15:17 +00:00)
```

Load Google Pretrained Model for future use

```
def load_model_pretrained():
    if not os.path.exists(W2VConfigValues.PRETRAINED_SAVED_PATH):
        os.makedirs(W2VConfigValues.GOOGLE_PRETRAINED_MODEL,
exist_ok=True)
        pretrained_model =
api.load(W2VConfigValues.GOOGLE_PRETRAINED_MODEL)
        shutil.copyfile(
            W2VConfigValues.PRETRAINED_OG_PATH,
W2VConfigValues.PRETRAINED_SAVED_PATH
        )
    else:
        pretrained_model =
gensim.models.keyedvectors.KeyedVectors.load_word2vec_format(
            W2VConfigValues.PRETRAINED_SAVED_PATH, binary=True
        )
    return pretrained_model


# Load the pretrained model
google_pretrained_model = load_model_pretrained()

time: 43.4 s (started: 2025-02-13 23:49:05 +00:00)
```

Load and Preprocess dataset

Reference :

- https://stackoverflow.com/questions/16694907/download-large-file-in-python-with-requests
- https://inside-machinelearning.com/en/open-parquet-python/

```
# url =
"https://web.archive.org/web/20201127142707if_/https://s3.amazonaws.co
m/amazon-reviews-pds/tsv/
amazon_reviews_us_Office_Products_v1_00.tsv.gz"
# data = pd.read_csv(url, sep='\t', compression='gzip',
on_bad_lines='skip')

os.makedirs(os.path.dirname(ConfigValues.DATA_PATH), exist_ok=True)
if not os.path.exists(ConfigValues.DATA_PATH):
    url = ConfigValues.URL
    file_name = ConfigValues.DATA_PATH

    # Stream and download heavy file in chunks
    with requests.get(url, stream=True) as response:
        if response.status_code == 200:
            with open(file_name, "wb") as file:
                for chunk in response.iter_content(chunk_size=8192):
                    file.write(chunk)
            print(f"Dataset downloaded successfully.")
        else:
            print(f"Failed to download the file. HTTP Status:
{response.status_code}")

else:
    print(f"File '{ConfigValues.DATA_PATH}' already exists.")

# Load dataset
if os.path.exists(ConfigValues.PARQUET_PATH):
    print("Loading dataset from Parquet")
    data = pd.read_parquet(ConfigValues.PARQUET_PATH)
else:
    print("Loading dataset from TSV")
    data = pd.read_csv(ConfigValues.DATA_PATH, sep='\t',
compression='gzip', on_bad_lines='skip')

data = data[['review_body', 'star_rating']]
data['star_rating'] = pd.to_numeric(data['star_rating'],
errors='coerce')
data = data.dropna(subset=['star_rating'])
data['star_rating'] = data['star_rating'].astype(int)
data = data.dropna(subset=['review_body'])
```

```
# Save to Parquet for faster future loads
print("Saving dataset to Parquet format for faster access next time")
data.to_parquet(ConfigValues.PARQUET_PATH, engine='fastparquet')
# data.head(10)
```

```
Dataset downloaded successfully.
Loading dataset from TSV
Saving dataset to Parquet format for faster access next time
time: 1min 37s (started: 2025-02-13 22:50:36 +00:00)
```

Balance the dataset (50k samples per rating) and Assign Ternary Labels

```
balanced_data = data.groupby("star_rating").apply(lambda x:
x.sample(ConfigValues.N_SAMPLES_EACH_CLASS,
replace=True)).reset_index(drop=True)

def get_label(rating):
       return 1 if rating > 3 else 2 if rating < 3 else 3

balanced_data["label"] = balanced_data["star_rating"].apply(get_label)

num_rows = balanced_data.shape[0]
print("Number of rows:", num_rows)
# check if labelled corectly
sampled_data = balanced_data.groupby("label").apply(lambda x:
x.sample(1)).reset_index(drop=True)
print(sampled_data)
```

```
Number of rows: 250000
                                                review_body  star_rating
label
0  I previously thought that vTech was the best p...            5
1
1  I am on my second Onetouch 8650.  I had to shi...            1
2
2  Very good quality case, horrible keyboard. It'...            3
3
time: 494 ms (started: 2025-02-13 22:52:21 +00:00)
```

Clean & Process Data

From Assigment 1:

Using regex expressions to match and replace the below items with empty strings:

- change all to lower case
- URLs

- emails
- HTML tags
- punctuations
- extra spaces
- special / non-alphabetical characters

```
# clean & preprocess balanced_data
balanced_data.dropna(inplace=True)
balanced_data["review_body"] =
balanced_data["review_body"].astype(str)

# Remove URLs first
def remove_html_urls(text):
    text = re.sub(r'https?://\S+|www\.\S+', '', text)
    text = BeautifulSoup(text, "html.parser").get_text()
    return text

# REmove spaces & spl chars
def remove_space_characters(text):
  text = re.sub(r'\s+', ' ', text)
  text = re.sub(r'[^a-zA-Z\s]', '', text)
  text = re.sub(r'[a-zA-Z0-9_\-\.]+@[a-zA-Z0-9_\-\.]+\.[a-zA-Z]{2,5}',
' ', text)
  return text

# Stop Words
stop_owrds = set(stopwords.words('english'))
negitive_words = ['nor', 'no', 'not', 'none', 'nowhere' 'never',
'neither', 'nobody']
refined_stopwords = [word for word in stop_owrds if word not in
negitive_words]
def remove_stop_words(text):
  words = word_tokenize(text)
  filtered_words = [word for word in words if word.lower() not in
refined_stopwords]
  return ' '.join(filtered_words)

# Lemmatization
lemmatizer = WordNetLemmatizer()
def lemmatize_text(text):
  words = word_tokenize(text)
  lemmatized_words = [lemmatizer.lemmatize(word) for word in words]
  return ' '.join(lemmatized_words)

balanced_data["review_body"] =
balanced_data["review_body"].apply(remove_html_urls)
```

```
balanced_data["review_body"] =
balanced_data["review_body"].apply(remove_space_characters)
balanced_data["review_body"] =
balanced_data["review_body"].apply(remove_stop_words)
balanced_data["review_body"] =
balanced_data["review_body"].apply(lemmatize_text)
balanced_data['review_body'] =
balanced_data['review_body'].apply(lambda x: contractions.fix(x))

# Drop reviews that are empty
balanced_data =
balanced_data.loc[balanced_data["review_body"].str.strip() != ""]

# Tokenize Reviews
balanced_data["review_body"] =
balanced_data["review_body"].apply(word_tokenize)
balanced_data.head(10)
```

{"type":"dataframe","variable_name":"balanced_data"}

```
time: 3min 18s (started: 2025-02-13 23:11:40 +00:00)
```

```
# also extract word embeddings
def extract_embeddings(text, w2v_model, topn=None):
    word_embeddings = [w2v_model[word] for word in text if word in
w2v_model]
    if topn is not None:
      # For top10 concat, used further
        if len(word_embeddings) < topn:
            padding = [np.zeros(w2v_model.vector_size) for _ in
range(topn - len(word_embeddings))]
            word_embeddings.extend(padding)
        elif len(word_embeddings) > topn:
            word_embeddings = word_embeddings[:topn]
        word_embeddings = np.concatenate(word_embeddings, axis=0)
    else:
        if len(word_embeddings) == 0:
            word_embeddings = np.zeros(w2v_model.vector_size)
        else:
            word_embeddings = np.mean(word_embeddings, axis=0)

    return word_embeddings


# Preprocess data and generate word2vec embeddings Avg and top 10
balanced_data["embeddings"] =
balanced_data["review_body"].apply(lambda text:
extract_embeddings(text, google_pretrained_model, topn=None))

# Drop rows with NaN embeddings
balanced_data.dropna(subset=["embeddings"], inplace=True)
```

```
balanced_data["embeddings_top_10"] =
balanced_data["review_body"].apply(lambda text:
extract_embeddings(text, google_pretrained_model, topn=10))
```

```
time: 31 s (started: 2025-02-13 23:14:59 +00:00)
```

Save Ternary class model

Reference:

- https://stackoverflow.com/questions/41066582/python-save-pandas-data-frame-to-parquet-file

```
# Change the data type of the 'embeddings' column to float32
balanced_data["embeddings"] = balanced_data["embeddings"].apply(lambda
x: x.astype(np.float32) if isinstance(x, np.ndarray) else x)
balanced_data["embeddings_top_10"] =
balanced_data["embeddings_top_10"].apply(lambda x:
x.astype(np.float32) if isinstance(x, np.ndarray) else x)

balanced_data.to_parquet("amazon_reviews_balanced_ternary.parquet",
engine="pyarrow", index=False)
print("Dataset saved as 'amazon_reviews_balanced_ternary.parquet'")

# # TO LOAD PARQUET DATA:
# parquet_balanced_data =
pd.read_parquet("amazon_reviews_balanced.parquet", engine="pyarrow")
# print(parquet_balanced_data.head())
# print(f"Total rows in dataset: {len(parquet_balanced_data)}")
```

```
Dataset saved as 'amazon_reviews_balanced_ternary.parquet'
time: 42 s (started: 2025-02-13 23:15:30 +00:00)
```

Split dataset

```
# Split data
balanced_data = balanced_data.dropna(subset=['review_body', 'label'])
train_df, test_df = train_test_split(balanced_data,
test_size=ConfigValues.TEST_SPLIT,
random_state=ConfigValues.RANDOM_STATE_VALUE,
stratify=balanced_data["label"])
```

```
time: 251 ms (started: 2025-02-13 23:16:12 +00:00)
```

Word Embeddings:

Reference:

- https://radimrehurek.com/gensim/auto_examples/tutorials/run_word2vec.html
- https://www.kaggle.com/code/bavalpreet26/word2vec-pretrained
- https://stackabuse.com/implementing-word2vec-with-gensim-library-in-python/

(a) Load Pretrained Word2Vec

```
# verify pretrained model examples
print("google model: ", google_pretrained_model)
print(google_pretrained_model.most_similar(positive=["king", "woman"],
negative=["man"])[0])
print("Similarity between 'excellent' and 'outstanding':",
google_pretrained_model.similarity('excellent', 'outstanding'))
```

```
google model:  KeyedVectors<vector_size=300, 3000000 keys>
('queen', 0.7118193507194519)
Similarity between 'excellent' and 'outstanding': 0.55674857
time: 219 ms (started: 2025-02-13 23:16:12 +00:00)
```

Additional tests on pretrained model

```
# Check for more examples
print(google_pretrained_model.most_similar(positive=["paris",
"berlin"], negative=["france"])[0])
print("Similarity between 'excellent' and 'outstanding':",
google_pretrained_model.similarity('student', 'university'))
```

```
('kunst', 0.417977780103683 47)
Similarity between 'excellent' and 'outstanding': 0.60054
time: 103 ms (started: 2025-02-13 23:16:12 +00:00)
```

```
# delete pretrained model - handle ram, for now, DONT DELETE
# del google_pretrained_model
```

```
time: 421 µs (started: 2025-02-13 07:40:39 +00:00)
```

(b) Custom Word2Vec Model

```
sentences_w2v = train_df['review_body'].tolist()
print("Sample tokenized sentences:", sentences_w2v[:5])

# Word2Vec model training
custom_w2v_model = Word2Vec(sentences_w2v,
vector_size=W2VConfigValues.EMBEDDING_SIZE,
window=W2VConfigValues.WINDOW_SIZE,
min_count=W2VConfigValues.MIN_WORD_COUNT, workers=4)
custom_w2v_model.save("custom_word2vec.model")

try:
    print("King : Man :: Woman : ",
custom_w2v_model.wv.most_similar(positive=['king', 'woman'],
```

```
negative=['man'])[0])
    print("Similarity between 'excellent' and 'outstanding': ",
custom_w2v_model.wv.similarity('excellent', 'outstanding'))
except KeyError as e:
    print(f"Word not in vocabulary: {e}")
```

```
Sample tokenized sentences: [['excellent', 'product', 'Inkoneram',
'company', 'use'], ['Came', 'quickly', 'good', 'leaving', 'note',
'pinning', 'reminder', 'Kids', 'love', 'jot', 'note', 'dry', 'erase',
'part'], ['fountain', 'pen', 'buyer', 'like', 'not', 'buy', 'pen',
'without', 'knowing', 'nib', 'size', 'Every', 'supplier', 'Amazon',
'one', 'found', 'true', 'every', 'Germansounding', 'cigarshaped',
'pen', 'they', 'are', 'currently', 'selling', 'advice', 'sell',
'Lamywidth', 'F', 'fine', 'nib', 'desirable', 'size', 'limited',
'competition'], ['Ordered', 'keep', 'Visor', 'belt', 'clip', 'hold',
'belt', 'nicely', 'bend', 'squat', 'Visor', 'fall', 'belt', 'clipI',
'since', 'bought', 'small', 'leather', 'pouch', 'hold', 'Visor',
'belt', 'work', 'great'], ['printing', 'job', 'find', 'ECO', 'Ink',
'good', 'original', 'Dell', 'Series', 'ink', 'problem', 'printer',
'constantly', 'telling', 'ink', 'low', 'run', 'fact', 'installed',
'new', 'cartridge', 'use', 'printer', 'personal', 'use']]
King : Man :: Woman : ('lady' , 0.6125918626785279)
Similarity between 'excellent' and 'outstanding': 0.37514123
time: 30.6 s (started: 2025-02-13 23:16:12 +00:00)
```

```
# Check for more examples
try:
  print("Custom W2V Model ===> boat : car :: wheel :
",custom_w2v_model.wv.most_similar(positive=["boat", "car"],
negative=["wheel"])[0])
  print("Custom W2V Model ===> Similarity between 'student' and
'university':", custom_w2v_model.wv.similarity('student',
'university'))
except KeyError as e:
    print(f"Word not in vocabulary: {e}")
```

```
Custom W2V Model ===> boat : car :: wheel : ('road' ,
06838219704654968)
Custom W2V Model ===> Similarity between 'student' and
'university':0.69874583
time: 535 µs (started: 2025-02-13 23:16:43 +00:00)
```

Conclusion

1. **Pretrained "word2vec-google-news-300" Model:**
   - this model has been pretrained on a vast corpus, which shows strong generalization and the ability to capture a wide range of semantic relationships across different domains.
   - Does not capture domain-specific relationships as effectively as models trained on specialized data.

2. **Custom-trained Word2Vec Model:**
   - My custom model excels at capturing domain-specific relationships when trained on specialized data.
   - But it struggles with tasks outside its domain, and embedding quality depends on the size and representativeness of the training dataset.

- The semantic similarity score is higher for the pretrained model compared to the custom model. This indicates that the pretrained model is better at encoding semantic similarities between words.

- The custom Word2Vec model, which was trained on the provided dataset, may not have had access to as diverse and extensive a corpus as the pretrained model. This can lead to limitations in its ability to generalize and capture nuanced semantic relationships.

SIMPLE MODELS

Calculate Avg W2V Features

```
def get_avg_w2v(tokens, model,
embedding_size=W2VConfigValues.EMBEDDING_SIZE):
    vectors = []
    for token in tokens:
        if token in model:
            vectors.append(model[token])
    if len(vectors) == 0:
        return np.zeros(embedding_size)
    return np.mean(vectors, axis=0)

# For Google's model
balanced_data["w2v_google"] = balanced_data["review_body"].apply(
    lambda x: get_avg_w2v(x, google_pretrained_model)
)

# For custom model
balanced_data["w2v_custom"] = balanced_data["review_body"].apply(
    lambda x: get_avg_w2v(x, custom_w2v_model.wv)
)
```

```
time: 38 s (started: 2025-02-13 23:16:43 +00:00)
```

```
# For Word2Vec features - pretrained and custom models
X_google = np.stack(balanced_data["w2v_google"].values)
X_custom = np.stack(balanced_data["w2v_custom"].values)
y = balanced_data["label"].values
```

```
time: 746 ms (started: 2025-02-13 23:17:23 +00:00)
```

TF-IDF Features Vectorization

```
# tf-idf comparison from assignment 1

from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(max_features=ConfigValues.MAX_TFIDF_FEATURES)
tfidf_features =
tfidf.fit_transform(balanced_data["review_body"].apply(" ".join))
```

```
time: 6.18 s (started: 2025-02-13 23:17:24 +00:00)
```

Train-test split for all features

```
# Split all feature sets for both pretrained &custom models

X_google_train, X_google_test, y_train, y_test = train_test_split(
    X_google, y, test_size=ConfigValues.TEST_SPLIT,
random_state=ConfigValues.RANDOM_STATE_VALUE
)
X_custom_train, X_custom_test, _, _ = train_test_split(
    X_custom, y, test_size=ConfigValues.TEST_SPLIT,
random_state=ConfigValues.RANDOM_STATE_VALUE
)
X_tfidf_train, X_tfidf_test, _, _ = train_test_split(
    tfidf_features, y, test_size=ConfigValues.TEST_SPLIT,
random_state=ConfigValues.RANDOM_STATE_VALUE
)
```

```
time: 346 ms (started: 2025-02-13 23:17:30 +00:00)
```

Train models & evaluate

```
#helper fun to train both perceptron + svm
def train_evaluate_model(model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    return model.score(X_test, y_test)

perceptron = Perceptron()
svm = LinearSVC()
```

```
time: 460 µs (started: 2025-02-13 23:17:30 +00:00)
```

1) Perceptron

```
# Google News W2V
p_acc_google = train_evaluate_model(perceptron, X_google_train,
X_google_test, y_train, y_test)

# Custom W2V
```

```
p_acc_custom = train_evaluate_model(perceptron, X_custom_train,
X_custom_test, y_train, y_test)

# TF-IDF (from HW1)
p_acc_tfidf = train_evaluate_model(perceptron, X_tfidf_train,
X_tfidf_test, y_train, y_test)

time: 11.3 s (started: 2025-02-13 23:17:30 +00:00)
```

2) SVM

```
# Google News W2V
s_acc_google = train_evaluate_model(svm, X_google_train,
X_google_test, y_train, y_test)

# Custom W2V
s_acc_custom = train_evaluate_model(svm, X_custom_train,
X_custom_test, y_train, y_test)

# TF-IDF (from HW1)
s_acc_tfidf = train_evaluate_model(svm, X_tfidf_train, X_tfidf_test,
y_train, y_test)

time: 10min 8s (started: 2025-02-13 23:17:41 +00:00)

# Results for comparison

results = pd.DataFrame({
    "Feature Type": ["Google W2V", "Custom W2V", "TF-IDF"],
    "Perceptron": [p_acc_google, p_acc_custom, p_acc_tfidf],
    "SVM": [s_acc_google, s_acc_custom, s_acc_tfidf]
})

print(results)

  Feature Type  Perceptron       SVM
0   Google W2V    0.593336  0.664719
1   Custom W2V    0.660576  0.692075
2       TF-IDF    0.669322  0.726016
time: 5.64 ms (started: 2025-02-13 23:27:49 +00:00)
```

Analysis and Conclusions

- **Pretrained W2V Performance:** Gives deceent baseline results due to rich semantic information from massive training data.

- **Custom W2V Performance:** May underperform compared to pretrained due to smaller domain-specific data, but could capture niche patterns.

- **TF-IDF Performance:** Likely highest accuracy but lacks semantic understanding. Might outperform custom W2V if domain is very different from pretrained data.

Key Findings:

- **Pretrained embeddings** generally outperform others when semantic relationships are crucial
- **Custom embeddings** need sufficient domain-specific data to be effective
- **TF-IDF** remains competitive for simple classification tasks

Pytorch & FNN

Reference:

- http://pytorch.org/tutorials/beginner/basics/tensorqs_tutorial.html
- https://machinelearningmastery.com/building-multilayer-perceptron-models-in-pytorch/
- https://discuss.pytorch.org/t/how-to-create-mlp-model-with-arbitrary-number-of-hidden-layers/13124/6
- https://www.tutorialspoint.com/how-to-compute-the-cross-entropy-loss-between-input-and-target-tensors-in-pytorch

```
# refernece from prev embeddings calc
def get_top10_concat(tokens, model,
embedding_size=W2VConfigValues.EMBEDDING_SIZE, topn=10):
    vectors = []
    for token in tokens:
        if token in model:
            vectors.append(model[token])
        if len(vectors) == topn:
            break
    # If fewer than topn vectors are found, pad with zeros
    if len(vectors) < topn:
        pad_vector = np.zeros(embedding_size)
        vectors.extend([pad_vector] * (topn - len(vectors)))
    return np.concatenate(vectors)
```

```
time: 541 µs (started: 2025-02-13 23:27:51 +00:00)
```

```
# pretrained model top 10 concat features
balanced_data["w2v_google_top10"] =
balanced_data["review_body"].apply(
    lambda tokens: get_top10_concat(tokens, google_pretrained_model,
embedding_size=W2VConfigValues.EMBEDDING_SIZE, topn=10)
)

# custom model top 10 concat features
balanced_data["w2v_custom_top10"] =
balanced_data["review_body"].apply(
    lambda tokens: get_top10_concat(tokens, custom_w2v_model.wv,
embedding_size=W2VConfigValues.EMBEDDING_SIZE, topn=10)
)
```

```
time: 12 s (started: 2025-02-13 23:27:51 +00:00)
```

```
balanced_data.head(10)
```

{"type":"dataframe","variable_name":"balanced_data"}

```
time: 49.6 ms (started: 2025-02-13 23:28:03 +00:00)
```

Helper Functions

```
import random

# Set random seeds for reproducibility
seed = 42
torch.manual_seed(seed)
np.random.seed(seed)
random.seed(seed)

# Heavey processing
# Device configuration
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Create dataLoader using features and labels.
def create_dataloader(X, y, batch_size=64, shuffle=True):
    X_tensor = torch.tensor(np.stack(X.values), dtype=torch.float32)
    y_tensor = torch.tensor(y.values, dtype=torch.long)
    dataset = TensorDataset(X_tensor, y_tensor)
    return DataLoader(dataset, batch_size=batch_size, shuffle=shuffle)


# 1 epoch pass
def train_epoch(model, dataloader, criterion, optimizer,
device=device):
    model.train()
    running_loss = 0.0
    for inputs, labels in dataloader:
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item() * inputs.size(0)
    epoch_loss = running_loss / len(dataloader.dataset)
    return epoch_loss


def evaluate_model(model, dataloader, device=device):
    model.eval()
    correct, total = 0, 0
```

```
    with torch.no_grad():
        for inputs, labels in dataloader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    return correct / total
```

```
time: 70.3 ms (started: 2025-02-13 21:13:00 +00:00)
```

Define the MLP Model

```
class MLP(nn.Module):
    def __init__(self, input_dim, hidden1, hidden2, output_dim):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden1)
        self.fc2 = nn.Linear(hidden1, hidden2)
        self.fc3 = nn.Linear(hidden2, output_dim)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.3)

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

```
time: 744 µs (started: 2025-02-13 20:33:53 +00:00)
```

```
# model training & eval
def run_testcase(input_dim, hidden1, hidden2, output_dim,
train_loader, test_loader, epochs=10, lr=0.001):
    model = MLP(input_dim, hidden1, hidden2, output_dim).to(device)
    optimizer = optim.Adam(model.parameters(), lr=lr)
    criterion = nn.CrossEntropyLoss()

    for epoch in range(epochs):
        loss = train_epoch(model, train_loader, criterion, optimizer,
device)
        print(f"Epoch {epoch+1}/{epochs}, Loss: {loss:.4f}")

    acc = evaluate_model(model, test_loader, device)
    return model, acc
```

```
time: 931 µs (started: 2025-02-13 20:33:55 +00:00)
```

```
# DISCARDING BELOW CODE TO CREATE REUSABLE FUNCTIONS
```

```
# # Binary Class
# binary_data = balanced_data[balanced_data['label'].isin([1,
2])].copy()
# # Split into train and test
# X_bin_avg = binary_data['embeddings']  # For part (a) Avg w2v
features
# y_bin = binary_data['label'].replace({1: 0, 2: 1})  # Convert to 0/1
labels
# X_train_bin_avg, X_test_bin_avg, y_train_bin, y_test_bin =
train_test_split(
#     X_bin_avg, y_bin, test_size=ConfigValues.TEST_SPLIT,
random_state=seed, stratify=y_bin
# )
```

```
time: 192 ms (started: 2025-02-13 08:27:05 +00:00)
```

```
# DISCARDING BELOW CODE TO CREATE REUSABLE FUNCTIONS
# # Ternary Classes
# ternary_data = balanced_data.copy()
# ternary_data['label'] = ternary_data['label'] - 1  # Map 1/2/3 →
0/1/2
# # Split into train and test
# X_tern_avg = ternary_data['embeddings']  # For part (a) Avg w2V
features
# y_tern = ternary_data['label']
# X_train_tern_avg, X_test_tern_avg, y_train_tern, y_test_tern =
train_test_split(
#     X_tern_avg, y_tern, test_size=0.2, random_state=seed,
stratify=y_tern
# )
```

```
time: 203 ms (started: 2025-02-13 08:27:08 +00:00)
```

Define PyTorch datasets and DataLoaders

```
# DISCARDING BELOW CODE TO CREATE REUSABLE FUNCTIONS
# # Create DataLoaders for average features
# batch_size = 64
# train_loader_bin_avg = create_dataloader(X_train_bin_avg,
y_train_bin, batch_size=batch_size, shuffle=True)
# test_loader_bin_avg  = create_dataloader(X_test_bin_avg, y_test_bin,
batch_size=batch_size, shuffle=False)

# train_loader_tern_avg = create_dataloader(X_train_tern_avg,
y_train_tern, batch_size=batch_size, shuffle=True)
# test_loader_tern_avg  = create_dataloader(X_test_tern_avg,
y_test_tern, batch_size=batch_size, shuffle=False)

# DISCARDING BELOW CODE TO CREATE REUSABLE FUNCTIONS
# # For concatenated top 10 Word2Vec features
```

```
# X_bin_concat = binary_data['embeddings_top_10']
# X_tern_concat = ternary_data['embeddings_top_10']

# X_train_bin_concat, X_test_bin_concat, y_train_bin_concat,
y_test_bin_concat = train_test_split(
#     X_bin_concat, y_bin, test_size=0.2, random_state=seed,
stratify=y_bin
# )

# X_train_tern_concat, X_test_tern_concat, y_train_tern_concat,
y_test_tern_concat = train_test_split(
#     X_tern_concat, y_tern, test_size=0.2, random_state=seed,
stratify=y_tern
# )

# DISCARDING BELOW CODE TO CREATE REUSABLE FUNCTIONS
# train_loader_bin_concat = create_dataloader(X_train_bin_concat,
y_train_bin_concat, batch_size=batch_size, shuffle=True)
# test_loader_bin_concat  = create_dataloader(X_test_bin_concat,
y_test_bin_concat, batch_size=batch_size, shuffle=False)

# train_loader_tern_concat = create_dataloader(X_train_tern_concat,
y_train_tern_concat, batch_size=batch_size, shuffle=True)
# test_loader_tern_concat  = create_dataloader(X_test_tern_concat,
y_test_tern_concat, batch_size=batch_size, shuffle=False)

experiments = {
    # Part A - avg W2V
    'Avg W2V Pretrained - Binary': {
        'feature_col': 'w2v_google', 'task': 'binary', 'input_dim':
300},
    'Avg W2V Pretrained - Ternary': {
        'feature_col': 'w2v_google', 'task': 'ternary', 'input_dim':
300},
    'Avg W2V Custom - Binary': {
        'feature_col': 'w2v_custom', 'task': 'binary', 'input_dim':
300},
    'Avg W2V Custom - Ternary': {
        'feature_col': 'w2v_custom', 'task': 'ternary', 'input_dim':
300},
    # PArt B - top 10 concat W2V
    'Top10 W2V Pretrained - Binary': {
        'feature_col': 'w2v_google_top10', 'task': 'binary',
'input_dim': 3000},
    'Top10 W2V Pretrained - Ternary': {
        'feature_col': 'w2v_google_top10', 'task': 'ternary',
'input_dim': 3000},
    'Top10 W2V Custom - Binary': {
        'feature_col': 'w2v_custom_top10', 'task': 'binary',
'input_dim': 3000},
```

```
    'Top10 W2V Custom - Ternary': {
        'feature_col': 'w2v_custom_top10', 'task': 'ternary',
'input_dim': 3000},
}
```

```
time: 498 µs (started: 2025-02-13 20:33:58 +00:00)
```

```
from torch.utils.data import Dataset, DataLoader, TensorDataset
results = {}
batch_size = 64
epochs = 10

# Loop over experiments
for exp_name, config in experiments.items():
    print("\n--- Running Experiment:", exp_name, "---")
    feature_col = config['feature_col']
    task = config['task']
    input_dim = config['input_dim']

    if task == 'binary':
        data_subset = balanced_data[balanced_data['label'].isin([1,
2])].copy()
        data_subset['binary_label'] = data_subset['label'].replace({1:
0, 2: 1})
        X = data_subset[feature_col]
        y = data_subset['binary_label']
        output_dim = 2
    else:
        data_subset = balanced_data.copy()
        # 1,2,3 becomes 0,1,2
        data_subset['ternary_label'] = data_subset['label'] - 1
        X = data_subset[feature_col]
        y = data_subset['ternary_label']
        output_dim = 3

    # data split
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=ConfigValues.TEST_SPLIT, random_state=seed,
stratify=y)
    train_loader = create_dataloader(X_train, y_train,
batch_size=batch_size, shuffle=True)
    test_loader  = create_dataloader(X_test, y_test,
batch_size=batch_size, shuffle=False)

    # run every exp case
    _, accuracy = run_testcase(input_dim=input_dim, hidden1=50,
hidden2=10,
                                 output_dim=output_dim,
train_loader=train_loader,
                                 test_loader=test_loader,
```

```
epochs=epochs, lr=0.001)
    results[exp_name] = accuracy
    print(f"{exp_name} Accuracy: {accuracy:.4f}")
```

```
--- Running Experiment: Avg W2V Pretrained - Binary ---
Epoch 1/10, Loss: 0.4043
Epoch 2/10, Loss: 0.3725
Epoch 3/10, Loss: 0.3623
Epoch 4/10, Loss: 0.3552
Epoch 5/10, Loss: 0.3480
Epoch 6/10, Loss: 0.3431
Epoch 7/10, Loss: 0.3374
Epoch 8/10, Loss: 0.3346
Epoch 9/10, Loss: 0.3328
Epoch 10/10, Loss: 0.3289
Avg W2V Pretrained - Binary Accuracy: 0.8593

--- Running Experiment: Avg W2V Pretrained - Ternary ---
Epoch 1/10, Loss: 0.7935
Epoch 2/10, Loss: 0.7568
Epoch 3/10, Loss: 0.7460
Epoch 4/10, Loss: 0.7381
Epoch 5/10, Loss: 0.7318
Epoch 6/10, Loss: 0.7272
Epoch 7/10, Loss: 0.7236
Epoch 8/10, Loss: 0.7208
Epoch 9/10, Loss: 0.7183
Epoch 10/10, Loss: 0.7160
Avg W2V Pretrained - Ternary Accuracy: 0.6966

--- Running Experiment: Avg W2V Custom - Binary ---
Epoch 1/10, Loss: 0.3416
Epoch 2/10, Loss: 0.3209
Epoch 3/10, Loss: 0.3134
Epoch 4/10, Loss: 0.3072
Epoch 5/10, Loss: 0.3036
Epoch 6/10, Loss: 0.3010
Epoch 7/10, Loss: 0.2980
Epoch 8/10, Loss: 0.2966
Epoch 9/10, Loss: 0.2945
Epoch 10/10, Loss: 0.2931
Avg W2V Custom - Binary Accuracy: 0.8789

--- Running Experiment: Avg W2V Custom - Ternary ---
Epoch 1/10, Loss: 0.7311
Epoch 2/10, Loss: 0.7065
Epoch 3/10, Loss: 0.6990
Epoch 4/10, Loss: 0.6941
Epoch 5/10, Loss: 0.6904
```

```
Epoch 6/10, Loss: 0.6869
Epoch 7/10, Loss: 0.6852
Epoch 8/10, Loss: 0.6826
Epoch 9/10, Loss: 0.6810
Epoch 10/10, Loss: 0.6794
Avg W2V Custom - Ternary Accuracy: 0.7149

--- Running Experiment: Top10 W2V Pretrained - Binary ---
Epoch 1/10, Loss: 0.4717
Epoch 2/10, Loss: 0.4250
Epoch 3/10, Loss: 0.3953
Epoch 4/10, Loss: 0.3673
Epoch 5/10, Loss: 0.3445
Epoch 6/10, Loss: 0.3240
Epoch 7/10, Loss: 0.3069
Epoch 8/10, Loss: 0.2926
Epoch 9/10, Loss: 0.2781
Epoch 10/10, Loss: 0.2682
Top10 W2V Pretrained - Binary Accuracy: 0.8116

--- Running Experiment: Top10 W2V Pretrained - Ternary ---
Epoch 1/10, Loss: 0.8528
Epoch 2/10, Loss: 0.8030
Epoch 3/10, Loss: 0.7738
Epoch 4/10, Loss: 0.7481
Epoch 5/10, Loss: 0.7247
Epoch 6/10, Loss: 0.7072
Epoch 7/10, Loss: 0.6893
Epoch 8/10, Loss: 0.6721
Epoch 9/10, Loss: 0.6584
Epoch 10/10, Loss: 0.6461
Top10 W2V Pretrained - Ternary Accuracy: 0.6524

--- Running Experiment: Top10 W2V Custom - Binary ---
Epoch 1/10, Loss: 0.4428
Epoch 2/10, Loss: 0.4083
Epoch 3/10, Loss: 0.3912
Epoch 4/10, Loss: 0.3750
Epoch 5/10, Loss: 0.3613
Epoch 6/10, Loss: 0.3487
Epoch 7/10, Loss: 0.3363
Epoch 8/10, Loss: 0.3265
Epoch 9/10, Loss: 0.3170
Epoch 10/10, Loss: 0.3075
Top10 W2V Custom - Binary Accuracy: 0.8197

--- Running Experiment: Top10 W2V Custom - Ternary ---
Epoch 1/10, Loss: 0.8278
Epoch 2/10, Loss: 0.7904
Epoch 3/10, Loss: 0.7731
```

```
Epoch 4/10, Loss: 0.7579
Epoch 5/10, Loss: 0.7450
Epoch 6/10, Loss: 0.7331
Epoch 7/10, Loss: 0.7231
Epoch 8/10, Loss: 0.7127
Epoch 9/10, Loss: 0.7044
Epoch 10/10, Loss: 0.6971
Top10 W2V Custom - Ternary Accuracy: 0.6634
time: 9min 4s (started: 2025-02-13 20:34:40 +00:00)
```

```
results_df = pd.DataFrame(list(results.items()),
columns=['Experiment', 'Accuracy'])
print("\nFinal Results:\n", results_df)
```

```

Final Results:
                         Experiment  Accuracy
0     Avg W2V Pretrained - Binary  0.859334
1    Avg W2V Pretrained - Ternary  0.696600
2         Avg W2V Custom - Binary  0.878921
3        Avg W2V Custom - Ternary  0.714872
4   Top10 W2V Pretrained - Binary  0.811603
5  Top10 W2V Pretrained - Ternary  0.652431
6       Top10 W2V Custom - Binary  0.819683
7      Top10 W2V Custom - Ternary  0.663398
time: 2.78 ms (started: 2025-02-13 20:43:58 +00:00)
```

CNN

Reference:

- https://www.digitalocean.com/community/tutorials/writing-cnns-from-scratch-in-pytorch

```
torch.cuda.empty_cache()
```

```
time: 301 µs (started: 2025-02-13 23:35:53 +00:00)
```

Helper Functions

```
# ref from prev impl
def get_sequence_embeddings(tokens, model,
max_length=W2VConfigValues.MAX_LENGTH,
embedding_size=W2VConfigValues.EMBEDDING_SIZE):
    embeddings = []
    for token in tokens[:max_length]:
        if token in model.key_to_index:
```

```
            embeddings.append(model[token])
        else:
            embeddings.append(np.zeros(embedding_size))
    if len(embeddings) < max_length:
        pad = [np.zeros(embedding_size)] * (max_length -
len(embeddings))
        embeddings.extend(pad)
    return np.array(embeddings)


def prepare_loaders(data, text_column='sequence_embeddings',
label_column='label', test_size=0.2, batch_size=16):
    # stack embeddings
    X = np.stack(data[text_column].values)
    y = data[label_column].values

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=test_size, stratify=y, random_state=42
    )

    # Convert arrays to tensors
    X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
    X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
    y_train_tensor = torch.tensor(y_train, dtype=torch.long)
    y_test_tensor = torch.tensor(y_test, dtype=torch.long)

    # Create TensorDatasets and DataLoaders
    train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
    test_dataset = TensorDataset(X_test_tensor, y_test_tensor)
    train_loader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True)
    test_loader = DataLoader(test_dataset, batch_size=batch_size)

    return train_loader, test_loader
```

```
time: 1.01 ms (started: 2025-02-13 23:38:40 +00:00)
```

```
# cnn model def
class CNN(nn.Module):
    def __init__(self, embedding_dim=W2VConfigValues.EMBEDDING_SIZE,
num_classes=2):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv1d(in_channels=embedding_dim,
out_channels=50, kernel_size=3, padding=1)
        self.relu1 = nn.ReLU()
        self.conv2 = nn.Conv1d(in_channels=50, out_channels=10,
kernel_size=3, padding=1)
        self.relu2 = nn.ReLU()
        self.global_pool = nn.AdaptiveMaxPool1d(1)
        self.fc = nn.Linear(10, num_classes)
```

```
    def forward(self, x):
        x = x.permute(0, 2, 1)
        x = self.relu1(self.conv1(x))
        x = self.relu2(self.conv2(x))
        x = self.global_pool(x).squeeze(-1)
        return self.fc(x)
```

```
time: 873 µs (started: 2025-02-13 23:39:35 +00:00)
```

```
def train_model(model, train_loader, optimizer, criterion, epochs,
device):
    model.to(device)
    for epoch in range(epochs):
        model.train()
        total_loss = 0.0
        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            total_loss += loss.item()
        avg_loss = total_loss / len(train_loader)
        print(f"Epoch {epoch+1}/{epochs}, Loss: {avg_loss:.4f}")

def evaluate_model(model, test_loader, device):
    model.to(device)
    model.eval()
    correct, total = 0, 0
    with torch.no_grad():
        for inputs, labels in test_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            _, predicted = torch.max(outputs, dim=1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    accuracy = correct / total
    print(f"Test Accuracy: {accuracy:.4f}")
    return accuracy
```

```
time: 1.03 ms (started: 2025-02-13 23:41:02 +00:00)
```

```
# for pretrained
balanced_data['sequence_embeddings_google'] =
balanced_data['review_body'].apply(
    lambda x: get_sequence_embeddings(x, google_pretrained_model,
max_length=50)
```

```
)

# for custom model
balanced_data['sequence_embeddings_custom'] =
balanced_data['review_body'].apply(
    lambda x: get_sequence_embeddings(x, custom_w2v_model.wv,
max_length=50)
)
```

```
time: 32.6 s (started: 2025-02-13 23:50:05 +00:00)
```

```
# create datasets for pretrained embeddings
# bin classification
binary_data_google = balanced_data[balanced_data['label'].isin([1,
2])].copy()
binary_data_google['label'] = binary_data_google['label'].replace({1:
0, 2: 1})
binary_data_google['sequence_embeddings'] =
binary_data_google['sequence_embeddings_google']

# ter classification
ternary_data_google = balanced_data.copy()
ternary_data_google['label'] = ternary_data_google['label'] - 1
ternary_data_google['sequence_embeddings'] =
ternary_data_google['sequence_embeddings_google']

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# train & eval
# bin classf
train_loader_bin_google, test_loader_bin_google = prepare_loaders(
    binary_data_google, text_column='sequence_embeddings',
label_column='label', test_size=ConfigValues.TEST_SPLIT, batch_size=16
)
model_bin_google = CNN(num_classes=2)
criterion_bin_google = nn.CrossEntropyLoss()
optimizer_bin_google = torch.optim.Adam(model_bin_google.parameters(),
lr=0.001)
train_model(model_bin_google, train_loader_bin_google,
optimizer_bin_google, criterion_bin_google, epochs=10, device=device)
accuracy_binary_pretrained = evaluate_model(model_bin_google,
test_loader_bin_google, device=device)

# tern classf
train_loader_tern_google, test_loader_tern_google = prepare_loaders(
    ternary_data_google, text_column='sequence_embeddings',
label_column='label', test_size=ConfigValues.TEST_SPLIT, batch_size=16
)
model_tern_google = CNN(num_classes=3)
criterion_tern_google = nn.CrossEntropyLoss()
```

```
optimizer_tern_google =
torch.optim.Adam(model_tern_google.parameters(), lr=0.001)
train_model(model_tern_google, train_loader_tern_google,
optimizer_tern_google, criterion_tern_google, epochs=10,
device=device)
accuracy_ternary_pretrained = evaluate_model(model_tern_google,
test_loader_tern_google, device=device)



import gc
gc.collect()
torch.cuda.empty_cache()
```

```
time: 778 ms (started: 2025-02-13 23:52:03 +00:00)
```

```
del google_pretrained_model
```

```
time: 311 µs (started: 2025-02-13 23:52:25 +00:00)
```

```
# create datasets for custom embeddings
# bin classification
binary_data_custom = balanced_data[balanced_data['label'].isin([1,
2])].copy()
binary_data_custom['label'] = binary_data_custom['label'].replace({1:
0, 2: 1})
binary_data_custom['sequence_embeddings'] =
binary_data_custom['sequence_embeddings_custom']

# ter classification
ternary_data_custom = balanced_data.copy()
ternary_data_custom['label'] = ternary_data_custom['label'] - 1
binary_data_custom['sequence_embeddings'] =
binary_data_custom['sequence_embeddings_custom']
ternary_data_custom['sequence_embeddings'] =
ternary_data_custom['sequence_embeddings_custom']

#t train & eval
train_loader_bin_custom, test_loader_bin_custom = prepare_loaders(
    binary_data_custom, text_column='sequence_embeddings',
label_column='label', test_size=ConfigValues.TEST_SPLIT, batch_size=16
)
model_bin_custom = CNN(num_classes=2)
criterion_bin_custom = nn.CrossEntropyLoss()
optimizer_bin_custom = torch.optim.Adam(model_bin_custom.parameters(),
lr=0.001)
train_model(model_bin_custom, train_loader_bin_custom,
optimizer_bin_custom, criterion_bin_custom, epochs=10, device=device)
accuracy_binary_custom = evaluate_model(model_bin_custom,
test_loader_bin_custom, device=device)
```

```
# tern classf
train_loader_tern_custom, test_loader_tern_custom = prepare_loaders(
    ternary_data_custom, text_column='sequence_embeddings',
label_column='label', test_size=ConfigValues.TEST_SPLIT, batch_size=16
)
model_tern_custom = CNN(num_classes=3)
criterion_tern_custom = nn.CrossEntropyLoss()
optimizer_tern_custom =
torch.optim.Adam(model_tern_custom.parameters(), lr=0.001)
train_model(model_tern_custom, train_loader_tern_custom,
optimizer_tern_custom, criterion_tern_custom, epochs=10,
device=device)
accuracy_ternary_custom = evaluate_model(model_tern_custom,
test_loader_tern_custom, device=device)

print(f"Binary Pretrained - Accuracy:
{accuracy_binary_pretrained:.4f}")
print(f"Binary Custom Model - Accuracy: {accuracy_binary_custom:.4f}")
print(f"Ternary Pretrained - Accuracy:
{accuracy_ternary_pretrained:.4f}")
print(f"Ternary Custom Model - Accuracy:
{accuracy_ternary_custom:.4f}")
```

```
Binary Pretrained - Accuracy: 0.8575
Binary Custom Model - Accuracy: 0.8978
Ternary Pretrained - Accuracy: 0.7739
Ternary Custom Model - Accuracy: 0.8132
time: 545 µs (started: 2025-02-13 23:50:38 +00:00)
```

Baed on the above computations, consolidating the values:

Step 2 Word Embeddings:

- Part a: Accuracy for Pre-trained model similarities :

  ('queen', 0.7118193507194519)

  Similarity between 'excellent' and 'outstanding': 0.55674857

- Part b: Accuracy Custom model similarities:

  King : Man :: Woman : ('lady' , 0.6125918626785279)

  Similarity between 'excellent' and 'outstanding': 0.37514123

Step 3 Simple models:

TF-IDF Features:

- Perceptron Accuracy: 0.669322
- SVM Accuracy: 0.726016

Pre-trained Word2Vec Features:

- Perceptron Accuracy: 0.593336
- SVM Accuracy: 0.664719

Custom Word2Vec Features:

- Perceptron Accuracy: 0.660576
- SVM Accuracy: 0.692075

Step 4 Feedforward Neural Networks:

Part (a) - Average Word2Vec Results:

- Pretrained-Binary-Avg: 0.859334
- Pretrained-Ternary-Avg: 0.696600
- Custom-Binary-Avg: 0.878921
- Custom-Ternary-Avg: 0.714872

Part (b) - Concatenated Word2Vec Results:

- Pretrained-Binary-Concat: 0.811603
- Pretrained-Ternary-Concat: 0.652431
- Custom-Binary-Concat: 0.819683
- Custom-Ternary-Concat: 0.663398

Step 5 Convolutional Neural Networks:

- Pretrained-Binary: 0.8575
- Custom-Binary: 0.8978
- Pretrained-Ternary: 0.7739
- Custom-Ternary: 0.8132

Conclusion

Feature Representations

- TF-IDF outperforms Word2Vec, achieving the highest accuracy in traditional models. Custom Word2Vec embeddings perform better than pretrained ones, especially in neural models.

Model Comparisons

- SVM consistently outperforms Perceptron, with TF-IDF giving the best results. MLP and CNN models surpass traditional models, with CNN achieving the highest accuracy.

CNN Models

- CNN's strong performance suggests deep learning models are more effective in sentiment analysis.

Overall Performance

CNN with Custom Word2Vec achieves the highest accuracy (89.78%). TF-IDF is best for traditional models, while Custom Word2Vec is ideal for deep learning approaches.

THE END