In [1]:
```
import numpy as np
import pandas as pd
# from datasets import load_dataset, Dataset, DatasetDict
from nltk.tokenize import sent_tokenize, word_tokenize
from gensim.models import Word2Vec
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.model_selection import KFold
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix
from sklearn import svm
from tqdm import tqdm
from keras.preprocessing import sequence
from sklearn.decomposition import PCA
from nltk.probability import FreqDist
```

In [2]:
```
"""
Utility functions.
"""
def f1_score(tp, fp, fn):
    return (2 * tp) / (2 * tp + fp + fn)

def precision_score(tp, fp):
    return tp / (tp + fp)

def accuracy_score(tp, fp, tn, fn):
    return (tp + tn) / (tp + fp + tn + fn)

def recall_score(tp, fn):
    return tp / (tp + fn)

def flatten(matrix):
    flat_list = []
    for row in matrix:
        flat_list += row
    return flat_list
```

In [3]:
```
"""
Download dataset SubtaskA.jsonl from
https://github.com/mbzuai-nlp/M4GT-Bench.
"""
DATA_PATH = "C:/Users/Admin/Desktop/cse842_hw3/SubtaskA.jsonl"

# initialize dataset
df = pd.read_json(DATA_PATH, lines=True)
df = df[['text', 'label', 'model']]
print(df)
```

	text	label	model
0	We consider a system of many polymers in solut...	1	cohere
1	We present a catalog of 66 YSOs in the Serpens...	1	cohere
2	Spectroscopic Observations of the Intermediate...	1	cohere
3	We present a new class of stochastic Lie group...	1	cohere
4	ALMA as the ideal probe of the solar chromosph...	1	cohere
...	...	...	...
152804	The main results presented in this dissertati...	0	human
152805	Fine-grained sketch-based image retrieval (FG...	0	human
152806	We present the derivation of the NNLO two-par...	0	human
152807	The principle of optimism in the face of unce...	0	human
152808	We consider the setting of prediction with ex...	0	human

[152809 rows x 3 columns]

In [14]:
```
"""
Evaluate model using count/TFIDF vectorization.
"""
def run_cv(model, X, y, count_vectorizer, tfidf_transformer=None):
    results = []
    k_fold = KFold(n_splits=K_FOLDS, shuffle=True, random_state=777)
    for train, test in tqdm(k_fold.split(X, y)):
        # split fold into training & testing sets
        X_train, y_train, X_test, y_test = X[train], y[train], X[test], y[test]

        # fit & transform data sets
        print("Count vectorizing...")
        X_train = count_vectorizer.fit_transform(X_train)
        X_test = count_vectorizer.transform(X_test)

        if tfidf_transformer:
            print("TFIDF transforming...")
            X_train = tfidf_transformer.fit_transform(X_train)
            X_test = tfidf_transformer.transform(X_test)

        # train the model
        print("Fitting the model...")
        model.fit(X_train, y_train)

        # test the model
        print("Predicting the model...")
        y_hat = model.predict(X_test)

        # evaluate the model
        tn, fp, fn, tp = confusion_matrix(y_test, y_hat).ravel()
        results.append({
            'accuracy': accuracy_score(tp=tp, fp=fp, tn=tn, fn=fn),
            'recall': recall_score(tp=tp, fn=fn),
            'precision': precision_score(tp=tp, fp=fp),
            'f1': f1_score(tp=tp, fp=fp, fn=fn),
        })

    # analyze the run results
    results_df = pd.DataFrame.from_records(results).mean()

    return results_df
```

In [15]:
```
"""
Train and evaluate SVM classifier model using count/TFIDF vectorization.
"""
# consts
MAX_FEATURES = 3000
K_FOLDS = 3
MIN_DF = 2
MAX_DF = 0.7
NGRAM_RANGE = (1, 1)
ANALYZER = 'word'

# init model
model = svm.SVC(
```

```
    verbose=True,
    max_iter=-1,
    kernel='linear',
)

# load the data set
X = np.array(df.text)
y = np.array(df.label)

# init vectorizer and transformer
count_vectorizer = CountVectorizer(
    min_df=MIN_DF,
    max_df=MAX_DF,
    max_features=MAX_FEATURES,
    tokenizer=word_tokenize,
    token_pattern=None,
    ngram_range=NGRAM_RANGE,
    # strip_accents=STRIP_ACCENTS,
    # stop_words=STOP_WORDS,
)
tfidf_transformer = TfidfTransformer()

# run cross validation
results = run_cv(model, X, y, count_vectorizer, tfidf_transformer)
print(f"# model={model}, k_folds={K_FOLDS}, max_features={MAX_FEATURES}, min_df={MI
          f"ngram_range={NGRAM_RANGE}\n{results}")
```

```
0it [00:00, ?it/s]
Count vectorizing...
TFIDF transforming...
Fitting the model...
[LibSV
M].....................................................*.........................
.............*.....................*.*
optimization finished, #iter = 116277
obj = -27890.634461, rho = 2.379204
nSV = 31663, nBSV = 29212
Total nSV = 31663
Predicting the model...
1it [1:22:04, 4924.60s/it]
Count vectorizing...
TFIDF transforming...
Fitting the model...
[LibSV
M].....................................................*.........................
...............*.....................*.*
optimization finished, #iter = 122330
obj = -28099.448361, rho = 2.310642
nSV = 31863, nBSV = 29406
Total nSV = 31863
Predicting the model...
2it [2:44:41, 4943.49s/it]
```

```
Count vectorizing...
TFIDF transforming...
Fitting the model...
[LibSV
M].....................................................*.........................
.............*.....................*.*
optimization finished, #iter = 115238
obj = -27973.743292, rho = 2.387817
nSV = 31719, nBSV = 29263
Total nSV = 31719
Predicting the model...
3it [4:06:20, 4926.93s/it]
# model=SVC(kernel='linear', verbose=True), k_folds=3, max_features=3000, min_df=2,
max_df=0.7, ngram_range=(1, 1)
accuracy     0.887768
recall       0.904635
precision    0.900156
f1           0.902390
dtype: float64
```

In [4]:

```
"""
Train Word2Vec model & create embeddings.
"""
def tokenize(x):
    return word_tokenize(x.lower())

tokenized_text = df['text'].apply(tokenize)

# train Word2Vec model
w2v_model = Word2Vec(
    sentences=tokenized_text,
    vector_size=100,
    window=5,
    min_count=1,
    workers=4
)

# access embeddings
word_embeddings = w2v_model.wv
# print(word_embeddings['natural'])
print(word_embeddings)
```

```
KeyedVectors<vector_size=100, 803002 keys>
```

In [5]:

```
"""
Sequencer to convert texts to word embedding sequences.
"""
class Sequencer():
    def __init__(self, all_words, max_words, seq_len, embedding_matrix):
        self.seq_len = seq_len
        self.embed_matrix = embedding_matrix

        # build vocab
        self.vocab = list(set(all_words))
        print(f"Vocab size: {len(self.vocab)}")
        word_fdist = FreqDist(self.vocab)
```

```
        self.vocab = [word for (word, freq) in word_fdist.most_common(n=max_words)]
        print(f"Vocab size: {len(self.vocab)}")

    def text_to_vector(self, text):
        """
        https://www.kaggle.com/code/mehmetlaudatekman/tutorial-word-embeddings-with
        """
        tokens = text.split()
        len_v = len(tokens)-1 if len(tokens) < self.seq_len else self.seq_len-1
        vec = []
        for tok in tokens[:len_v]:
            try:
                vec.append(self.embed_matrix[tok])
            except Exception as E:
                pass

        last_pieces = self.seq_len - len(vec)
        for i in range(last_pieces):
            vec.append(np.zeros(100,))

        return np.asarray(vec).flatten()
```

In [6]:

```
"""
Evaluate model using Word2Vec embeddings.
"""
def run_w2v(model, X, y):
    # run k-folds
    results = []
    k_fold = KFold(n_splits=K_FOLDS, shuffle=True, random_state=777)
    for train, test in tqdm(k_fold.split(X, y)):
        # split fold into training & testing sets
        train = train[:N_SAMPLES]
        test = test[:N_SAMPLES]
        X_train, y_train, X_test, y_test = X[train], y[train], X[test], y[test]

        # train the model
        print("Fitting the model...")
        model.fit(X_train, y_train)

        # test the model
        print("Predicting the model...")
        y_hat = model.predict(X_test)

        # evaluate the model
        tn, fp, fn, tp = confusion_matrix(y_test, y_hat).ravel()
        results.append({
            'accuracy': accuracy_score(tp=tp, fp=fp, tn=tn, fn=fn),
            'recall': recall_score(tp=tp, fn=fn),
            'precision': precision_score(tp=tp, fp=fp),
            'f1': f1_score(tp=tp, fp=fp, fn=fn),
        })

    # analyze the run results
    results_df = pd.DataFrame.from_records(results).mean()

    return results_df
```

```
def sequence_embeddings(X, max_words, seq_len, word_embeddings):
    # list of all sequence tokens
    all_tokens = flatten(X)
    print(len(all_tokens))

    # init sequencer
    print("Initializing sequencer...")
    sequencer = Sequencer(
        all_words=all_tokens,
        max_words=max_words,
        seq_len=seq_len,
        embedding_matrix=word_embeddings,
    )

    # vectorize token sequences
    print("Vectorizing token sequences...")
    X = np.asarray([sequencer.text_to_vector(" ".join(seq)) for seq in X])
    print(X.shape)

    # PCA dimensionality reduction
    print("Reducing sequence dimensions...")
    pca_model = PCA(n_components=0.99)
    pca_model.fit(X)
    print("Sum of variance ratios: ",sum(pca_model.explained_variance_ratio_))
    X = pca_model.transform(X)
    print(X.shape)

    return X
```

In [29]:

```
"""
Train and evaluate SVM classifier model using Word2Vec embeddings.
"""
# consts
K_FOLDS = 3
N_SAMPLES = 10_000
LEN_SEQUENCE = 15
MAX_WORDS = 1200

# init model
model = svm.SVC(
    verbose=True,
    max_iter=-1,
    kernel='linear',
)

# load the data set
print("Loading datasets...")
y = np.array(df.label)
X = sequence_embeddings(
    X=np.array(tokenized_text),
    max_words=MAX_WORDS,
    seq_len=LEN_SEQUENCE,
    word_embeddings=word_embeddings,
)
```

```
# run
print("Running w2v model...")
results = run_w2v(model, X, y)
print(f"# model={model}, k_folds={K_FOLDS}, n_samples={N_SAMPLES}, max_words={MAX_W
```

```
Loading datasets...
73389956
Initializing sequencer...
Vocab size: 803002
Vocab size: 1200
Vectorizing token sequences...
(152809, 1500)
Reducing sequence dimensions...
Sum of variance ratios:  0.9900348149783275
(152809, 1288)
Running w2v model...
0it [00:00, ?it/s]
```

```
Fitting the model...
[LibSV
M]...................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
```

```
.....................................................................................
.....................................................................................
.....................................................................................
.........................................*.............................................
.....................................................................................
.....................................................................................
.....................................................................................
..................*
optimization finished, #iter = 372033905
obj = -4133.966266, rho = 0.362423
nSV = 4806, nBSV = 3491
Total nSV = 4806
Predicting the model...
3it [2:55:40, 3513.45s/it]
# model=SVC(kernel='linear', verbose=True), k_folds=3, n_samples=10000, max_words=12
00, len_seq=15,
accuracy     0.610867
recall       0.659757
precision    0.768751
f1           0.710068
dtype: float64
```

In [9]:

```
"""
Evaluate model using count/TFIDF vectorization + Word2Vec embeddings.
"""
def run_cvw2v(model, X, y, count_vectorizer, tfidf_transformer, embedding_seqs):
    results = []
    k_fold = KFold(n_splits=K_FOLDS, shuffle=True, random_state=777)
    for train, test in tqdm(k_fold.split(X, y)):
        # split fold into training & testing sets
        train = train[:N_SAMPLES]
        test = test[:N_SAMPLES // 3]
        X_train, y_train, X_test, y_test = X[train], y[train], X[test], y[test]
        embedding_train = np.array(embedding_seqs[train])
        embedding_test = np.array(embedding_seqs[test])

        # fit & transform data sets
        print("Count vectorizing...")
        X_train = count_vectorizer.fit_transform(X_train)
        X_test = count_vectorizer.transform(X_test)
        print("TFIDF transforming...")
        X_train = tfidf_transformer.fit_transform(X_train)
        X_test = tfidf_transformer.transform(X_test)

        # add embeddings
        print("Adding sequence embeddings...")
        X_train = np.hstack([X_train.toarray(), embedding_train])
        X_test = np.hstack([X_test.toarray(), embedding_test])
        print(X_train.shape)
        print(X_test.shape)

        # train the model
        print("Fitting the model...")
        model.fit(X_train, y_train)
```

```
        # test the model
        print("Predicting the model...")
        y_hat = model.predict(X_test)

        # evaluate the model
        tn, fp, fn, tp = confusion_matrix(y_test, y_hat).ravel()
        results.append({
            'accuracy': accuracy_score(tp=tp, fp=fp, tn=tn, fn=fn),
            'recall': recall_score(tp=tp, fn=fn),
            'precision': precision_score(tp=tp, fp=fp),
            'f1': f1_score(tp=tp, fp=fp, fn=fn),
        })

    # analyze the run results
    results_df = pd.DataFrame.from_records(results).mean()

    return results_df
```

In [10]:

```
"""
Train and evaluate SVM classifier model using count/TFIDF vectorization + Word2Vec
"""
# consts
K_FOLDS = 3
N_SAMPLES = 25_000
LEN_SEQUENCE = 15
MAX_WORDS = 3000
MIN_DF = 2
MAX_DF = 0.7
NGRAM_RANGE = (1, 1)
ANALYZER = 'word'

# init model
model = svm.SVC(
    verbose=True,
    # max_iter=150_000,
    max_iter=-1,
    kernel='rbf',
)

# load the data set
print("Loading datasets...")
X = np.array(df.text)
y = np.array(df.label)

# create sequence embeddings
embedding_seqs = sequence_embeddings(
    X=np.array(tokenized_text),
    max_words=MAX_WORDS,
    seq_len=LEN_SEQUENCE,
    word_embeddings=word_embeddings,
)

# init vectorizer and transformer
count_vectorizer = CountVectorizer(
    min_df=MIN_DF,
    max_df=MAX_DF,
```

```
    max_features=MAX_WORDS,
    tokenizer=word_tokenize,
    token_pattern=None,
    ngram_range=NGRAM_RANGE,
)
tfidf_transformer = TfidfTransformer()

# run
print("Running cvw2v model...")
results = run_cvw2v(
    model,
    X, y,
    count_vectorizer,
    tfidf_transformer,
    embedding_seqs
)
print(f"# model={model}, k_folds={K_FOLDS}, n_samples={N_SAMPLES}, max_words={MAX_W
```

```
Loading datasets...
73389956
Initializing sequencer...
Vocab size: 803002
Vocab size: 3000
Vectorizing token sequences...
(152809, 1500)
Reducing sequence dimensions...
Sum of variance ratios:  0.9900116167961788
(152809, 1288)
Running cvw2v model...
0it [00:00, ?it/s]
Count vectorizing...
TFIDF transforming...
Adding sequence embeddings...
(25000, 4288)
(8333, 4288)
Fitting the model...
[LibSVM]Predicting the model...
1it [1:10:20, 4220.41s/it]
Count vectorizing...
TFIDF transforming...
Adding sequence embeddings...
(25000, 4288)
(8333, 4288)
Fitting the model...
[LibSVM]Predicting the model...
2it [2:21:54, 4263.58s/it]
Count vectorizing...
TFIDF transforming...
Adding sequence embeddings...
(25000, 4288)
(8333, 4288)
Fitting the model...
[LibSVM]Predicting the model...
3it [3:35:21, 4307.18s/it]
```

```
# model=SVC(verbose=True), k_folds=3, n_samples=25000, max_words=3000, len_seq=15, m
in_df=2, max_df=0.7, ngram_range=(1, 1)
accuracy     0.726229
recall       0.986138
precision    0.713071
f1           0.827657
dtype: float64
```

In [ ]: