notebook

April 29, 2024

1 Lab 02: Feature extraction and embeddings

[1]:
```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV, cross_val_score
```

1.0.1 A. Data Preparation

[2]:
```python
data = pd.read_csv('out.csv')
data.drop(['Unnamed: 0'],axis=1,inplace=True)
data
```

[2]:
```
            id                                               text author
0      id26305  this proces however afforded me no means of as…    EAP
1      id17569  it never once occurred to me that the fumbling…    HPL
2      id11008  in his left hand was a gold snuff box from whi…    EAP
3      id27763  how lovely is spring as we looked from windsor…    MWS
4      id12958  finding nothing else not even gold the superin…    HPL
…          …                                                …     …
19574  id17718  i could have fancied while i looked at it that…    EAP
19575  id08973  the lids clenched themselves together as if in…    EAP
19576  id05267  mais il faut agir that is to say a frenchman n…    EAP
19577  id17513  for an item of news like this it strikes us it…    EAP
19578  id00393  he laid a gnarled claw on my shoulder and it s…    HPL

[19579 rows x 3 columns]
```

1.0.2 B. Encoding of the Target Variable

[3]:
```
le = LabelEncoder()
data['author_encoded'] = le.fit_transform(data['author'])
data
```

[3]:
```
              id                                               text author  \
0      id26305  this proces however afforded me no means of as…    EAP
1      id17569  it never once occurred to me that the fumbling…    HPL
2      id11008  in his left hand was a gold snuff box from whi…    EAP
3      id27763  how lovely is spring as we looked from windsor…    MWS
4      id12958  finding nothing else not even gold the superin…    HPL
…          …                                                …      …
19574  id17718  i could have fancied while i looked at it that…    EAP
19575  id08973  the lids clenched themselves together as if in…    EAP
19576  id05267  mais il faut agir that is to say a frenchman n…    EAP
19577  id17513  for an item of news like this it strikes us it…    EAP
19578  id00393  he laid a gnarled claw on my shoulder and it s…    HPL

       author_encoded
0                   0
1                   1
2                   0
3                   2
4                   1
…                 …
19574               0
19575               0
19576               0
19577               0
19578               1

[19579 rows x 4 columns]
```

1.0.3 C. Construction of Training and Testing Sets

[4]:
```
X_train, X_test, y_train, y_test = train_test_split(data['text'].values,
                                                    data['author_encoded'].
 ↪values,
                                                    test_size=0.2,
                                                    random_state=33,
                                                    stratify =␣
 ↪data['author_encoded'].values)
```

[5]:
```
print(100*y_train.tolist().count(0)/(len(y_train)))
print(100*y_test.tolist().count(0)/(len(y_test)))
```

```
40.349869118304284
40.34729315628192
```

1.0.4 D. Vectorization Methods

1. Use the lexical frequency method and one-hot encoding to vectorize the training and testing datasets.

[6]:
```
from sklearn.feature_extraction.text import CountVectorizer

# création d'un objet CountVectorizer pour effectuer la fréquence lexicale et␣
↪l'encodage one-hot
# (
#     si binary=False => fréquence lexicale
#     si binary=True => one-hot encoding
#)
vectorizer = CountVectorizer(binary=False,analyzer= 'word',␣
↪stop_words='english')

# ajustement du vectorizer sur le texte d'entraînement
vectorizer.fit(X_train)

# transformation du texte d'entraînement et de test en vecteurs one-hot
train_cv = vectorizer.transform(X_train)
test_cv = vectorizer.transform(X_test)
```

[7]:
```
count_array_cv = train_cv.toarray()
df = pd.DataFrame(data=count_array_cv,columns = vectorizer.
↪get_feature_names_out())
df
```

[7]:
```
       ab  aback  abandon  abandoned  abandoning  abandonment  abaout  \
0       0      0        0          0           0            0       0
1       0      0        0          0           0            0       0
2       0      0        0          0           0            0       0
3       0      0        0          0           0            0       0
4       0      0        0          0           0            0       0
…      ..    …      …       …         …          …        …
15658   0      0        0          0           0            0       0
15659   0      0        0          0           0            0       0
15660   0      0        0          0           0            0       0
15661   0      0        0          0           0            0       0
15662   0      0        0          0           0            0       0

       abasement  abashed  abashment  …  zobnarian  zodiac  zodiacal  \
0              0        0          0  …          0       0         0
1              0        0          0  …          0       0         0
2              0        0          0  …          0       0         0
```

```
3                0        0          0  …           0       0         0
4                0        0          0  …           0       0         0
…            …      …        …  …        …     …       …
15658            0        0          0  …           0       0         0
15659            0        0          0  …           0       0         0
15660            0        0          0  …           0       0         0
15661            0        0          0  …           0       0         0
15662            0        0          0  …           0       0         0

       zoilus  zokar  zone  zones  zory  zubmizion  zuro
0           0      0     0      0     0          0     0
1           0      0     0      0     0          0     0
2           0      0     0      0     0          0     0
3           0      0     0      0     0          0     0
4           0      0     0      0     0          0     0
…         …    …   …    …   …       …   …
15658       0      0     0      0     0          0     0
15659       0      0     0      0     0          0     0
15660       0      0     0      0     0          0     0
15661       0      0     0      0     0          0     0
15662       0      0     0      0     0          0     0

[15663 rows x 24463 columns]
```

2 & 3. Train a TF-IDF vectorization model on the training part and vectorize it and vectorize the testing part

[8]:
```python
from sklearn.feature_extraction.text import TfidfVectorizer

# création d'un objet TfidfVectorizer pour effectuer la vectorisation TF-IDF
tfidf_vectorizer = TfidfVectorizer(analyzer= 'word', stop_words='english')

# ajustement du vectorizer sur le texte d'entraînement
tfidf_vectorizer.fit(X_train)

# transformation du texte d'entraînement et de test en vecteurs TF-IDF
train_tfidf = tfidf_vectorizer.transform(X_train)
test_tfidf = tfidf_vectorizer.transform(X_test)
```

[9]:
```python
count_array_tfidf = train_tfidf.toarray()
df = pd.DataFrame(data=count_array_tfidf,columns = tfidf_vectorizer.
 ↪get_feature_names_out())
df
```

[9]:
```
        ab  aback  abandon  abandoned  abandoning  abandonment  abaout  \
0      0.0    0.0      0.0        0.0         0.0          0.0     0.0
1      0.0    0.0      0.0        0.0         0.0          0.0     0.0
```

```
2      0.0    0.0      0.0        0.0         0.0           0.0      0.0
3      0.0    0.0      0.0        0.0         0.0           0.0      0.0
4      0.0    0.0      0.0        0.0         0.0           0.0      0.0
…     …     …      …        …         …           …      …
15658  0.0    0.0      0.0        0.0         0.0           0.0      0.0
15659  0.0    0.0      0.0        0.0         0.0           0.0      0.0
15660  0.0    0.0      0.0        0.0         0.0           0.0      0.0
15661  0.0    0.0      0.0        0.0         0.0           0.0      0.0
15662  0.0    0.0      0.0        0.0         0.0           0.0      0.0

       abasement  abashed  abashment  …  zobnarian  zodiac  zodiacal  \
0            0.0      0.0        0.0  …        0.0     0.0       0.0
1            0.0      0.0        0.0  …        0.0     0.0       0.0
2            0.0      0.0        0.0  …        0.0     0.0       0.0
3            0.0      0.0        0.0  …        0.0     0.0       0.0
4            0.0      0.0        0.0  …        0.0     0.0       0.0
…             …        …          …  …          …       …         …
15658        0.0      0.0        0.0  …        0.0     0.0       0.0
15659        0.0      0.0        0.0  …        0.0     0.0       0.0
15660        0.0      0.0        0.0  …        0.0     0.0       0.0
15661        0.0      0.0        0.0  …        0.0     0.0       0.0
15662        0.0      0.0        0.0  …        0.0     0.0       0.0

       zoilus  zokar  zone  zones  zory  zubmizion  zuro
0         0.0    0.0   0.0    0.0   0.0        0.0   0.0
1         0.0    0.0   0.0    0.0   0.0        0.0   0.0
2         0.0    0.0   0.0    0.0   0.0        0.0   0.0
3         0.0    0.0   0.0    0.0   0.0        0.0   0.0
4         0.0    0.0   0.0    0.0   0.0        0.0   0.0
…         …     …     …     …    …          …    …
15658     0.0    0.0   0.0    0.0   0.0        0.0   0.0
15659     0.0    0.0   0.0    0.0   0.0        0.0   0.0
15660     0.0    0.0   0.0    0.0   0.0        0.0   0.0
15661     0.0    0.0   0.0    0.0   0.0        0.0   0.0
15662     0.0    0.0   0.0    0.0   0.0        0.0   0.0

[15663 rows x 24463 columns]
```

1.0.5 E. Training

1. Create three models of the MLPClassifier type. (You can change the learning algorithm: use other scikit-learn algorithms)

[10]:
```python
from sklearn.neural_network import MLPClassifier

# Modèle 1 : 1 couche cachée avec 100 neurones
model1 = MLPClassifier(hidden_layer_sizes=(100,), max_iter=200, solver='adam',␣
 ↪random_state=1)
```

```
# Modèle 2 : 2 couches cachées avec 50 et 25 neurones, respectivement
model2 = MLPClassifier(hidden_layer_sizes=(50, 25), max_iter=200,␣
 ↪solver='adam', random_state=1)

# Modèle 3 : 3 couches cachées avec 100, 50 et 25 neurones, respectivement
model3 = MLPClassifier(hidden_layer_sizes=(100, 50, 25), max_iter=200,␣
 ↪solver='adam', random_state=1)
```

2,3 & 4.

- Train these three models on the three vector representations.
- Predict the classes by applying the three models to the three training representations.
- Display the classification report using performance measures (accuracy, precision, recall…).

[12]:
```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
import pickle



# création des modèles avec différentes architectures de couches cachées
models = [
    model1,
    model2,
    model3
]

# entrainement des modèles sur les différentes représentations
representations = [(train_cv, test_cv),(train_tfidf, test_tfidf)]
for i, model in enumerate(models):
    print(f"Model {i+1}")
    for j, rep in enumerate(representations):

        # entraînement du modèle
        model.fit(rep[0], y_train)

        # prédiction sur le jeu de train
        y_pred = model.predict(rep[0])

        # évaluation de la performance du modèle

        print(f"Representation {j+1}:")
        print(classification_report(y_train, y_pred))
```

```
        # save the model to disk
        filename = f'models/finalized_model{i+1}.{j+1}.sav'
        pickle.dump(model, open(filename, 'wb'))
```

```
Model 1
Representation 1:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      6320
           1       1.00      1.00      1.00      4508
           2       1.00      1.00      1.00      4835

    accuracy                           1.00     15663
   macro avg       1.00      1.00      1.00     15663
weighted avg       1.00      1.00      1.00     15663

Representation 2:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      6320
           1       1.00      1.00      1.00      4508
           2       1.00      1.00      1.00      4835

    accuracy                           1.00     15663
   macro avg       1.00      1.00      1.00     15663
weighted avg       1.00      1.00      1.00     15663

Model 2
Representation 1:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      6320
           1       1.00      1.00      1.00      4508
           2       1.00      1.00      1.00      4835

    accuracy                           1.00     15663
   macro avg       1.00      1.00      1.00     15663
weighted avg       1.00      1.00      1.00     15663

Representation 2:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      6320
           1       1.00      1.00      1.00      4508
           2       1.00      1.00      1.00      4835

    accuracy                           1.00     15663
```

```
   macro avg       1.00      1.00      1.00     15663
weighted avg       1.00      1.00      1.00     15663

Model 3
Representation 1:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      6320
           1       1.00      1.00      1.00      4508
           2       1.00      1.00      1.00      4835

    accuracy                           1.00     15663
   macro avg       1.00      1.00      1.00     15663
weighted avg       1.00      1.00      1.00     15663

Representation 2:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      6320
           1       1.00      1.00      1.00      4508
           2       1.00      1.00      1.00      4835

    accuracy                           1.00     15663
   macro avg       1.00      1.00      1.00     15663
weighted avg       1.00      1.00      1.00     15663
```

[15]:

```
import pickle
from sklearn.metrics import accuracy_score

#load the model from disk
filename = "models/finalized_model1.2.sav"
loaded_model = pickle.load(open(filename, 'rb'))
y_pred = loaded_model.predict(train_cv)

result = accuracy_score(y_pred, y_train)
print(result)
```

```
0.9983400370299432
```

1.0.6 F. Testing

[13]:

```
import time
import pickle
from sklearn.metrics import classification_report

for i in range(3):
    print('\n\n************************************************************')
```

```python
    print(f"Pour Model N°{i+1} et La Representation 1 (CountVect ) et 2␣
↪(TF-IDF)")
    representations = [test_cv, test_tfidf]

    for j, rep in enumerate(representations):
        print(f"Representation {j+1}:")

        # load the model from disk
        filename = f"models/finalized_model{i+1}.{j+1}.sav"
        loaded_model = pickle.load(open(filename, 'rb'))

        # Model prediction time
        start_time = time.time()
        y_pred = loaded_model.predict(rep)
        end_time = time.time()

        result = classification_report(y_pred, y_test)
        print(result)
        print(f"Model {i+1} prediction time: {(end_time - start_time)} seconds")
```

```
************************************************************
Pour Model N°1 et La Representation 1 (CountVect ) et 2 (TF-IDF)
Representation 1:
              precision    recall  f1-score   support

           0       0.72      0.76      0.74      1511
           1       0.70      0.73      0.72      1084
           2       0.77      0.71      0.74      1321

    accuracy                           0.73      3916
   macro avg       0.73      0.73      0.73      3916
weighted avg       0.73      0.73      0.73      3916

Model 1 prediction time: 0.0 seconds
Representation 2:
              precision    recall  f1-score   support

           0       0.76      0.78      0.77      1532
           1       0.73      0.78      0.76      1054
           2       0.80      0.73      0.77      1330

    accuracy                           0.77      3916
   macro avg       0.77      0.77      0.76      3916
weighted avg       0.77      0.77      0.77      3916
```

```
Model 1 prediction time: 0.005999088287353516 seconds


************************************************************
Pour Model N°2 et La Representation 1 (CountVect ) et 2 (TF-IDF)
Representation 1:
              precision    recall  f1-score   support

           0       0.74      0.75      0.75      1549
           1       0.72      0.73      0.73      1104
           2       0.76      0.73      0.74      1263

    accuracy                           0.74      3916
   macro avg       0.74      0.74      0.74      3916
weighted avg       0.74      0.74      0.74      3916

Model 2 prediction time: 0.004000425338745117 seconds
Representation 2:
              precision    recall  f1-score   support

           0       0.75      0.79      0.77      1501
           1       0.76      0.76      0.76      1129
           2       0.80      0.75      0.77      1286

    accuracy                           0.77      3916
   macro avg       0.77      0.77      0.77      3916
weighted avg       0.77      0.77      0.77      3916

Model 2 prediction time: 0.0038728713989257812 seconds


************************************************************
Pour Model N°3 et La Representation 1 (CountVect ) et 2 (TF-IDF)
Representation 1:
              precision    recall  f1-score   support

           0       0.78      0.78      0.78      1595
           1       0.75      0.77      0.76      1092
           2       0.77      0.76      0.77      1229

    accuracy                           0.77      3916
   macro avg       0.77      0.77      0.77      3916
weighted avg       0.77      0.77      0.77      3916

Model 3 prediction time: 0.019878625869750977 seconds
Representation 2:
              precision    recall  f1-score   support
```

```
           0       0.78      0.77      0.78      1602
           1       0.71      0.81      0.76       993
           2       0.80      0.73      0.77      1321

    accuracy                           0.77      3916
   macro avg       0.76      0.77      0.77      3916
weighted avg       0.77      0.77      0.77      3916

Model 3 prediction time: 0.013283014297485352 seconds
```

1.0.7 G. Vectorizations based on word embeddings

1.0.8 Word2Vec

[16]:
```python
from gensim.models.word2vec import Word2Vec
from gensim.models import FastText, KeyedVectors

X_train_tokens = [text.split() for text in X_train]
X_test_tokens = [text.split() for text in X_test]



# Skip-Gram
model_w2v_sg = Word2Vec(X_train_tokens, vector_size=200, window=5, min_count=1,␣
↪workers=4, sg=1)
# CBOW
model_w2v_cbow = Word2Vec(X_train_tokens, vector_size=200, window=5,␣
↪min_count=1, workers=4, sg=0)
```

1.0.9 Glove model

[28]:
```python
#must run this command on glove.6B.100d.txt file
#download glove.6B.100d.txt from kaggle and place it in lab2 folder
!python -m  gensim.scripts.glove2word2vec -i glove.6B.100d.txt -o glove.6B.100d.
↪word2vec.txt
```

```
2024-04-29 12:25:13,444 - glove2word2vec - INFO - running c:\Python311\Lib\site-
packages\gensim\scripts\glove2word2vec.py -i glove.6B.100d.txt -o
glove.6B.100d.word2vec.txt
c:\Python311\Lib\site-packages\gensim\scripts\glove2word2vec.py:125:
DeprecationWarning: Call to deprecated `glove2word2vec`
(KeyedVectors.load_word2vec_format(.., binary=False, no_header=True) loads GLoVE
text vectors.).
  num_lines, num_dims = glove2word2vec(args.input, args.output)
2024-04-29 12:25:13,444 - keyedvectors - INFO - loading projection weights from
glove.6B.100d.txt
2024-04-29 12:25:45,878 - utils - INFO - KeyedVectors lifecycle event {'msg':
'loaded (400000, 100) matrix of type float32 from glove.6B.100d.txt', 'binary':
False, 'encoding': 'utf8', 'datetime': '2024-04-29T12:25:45.858322', 'gensim':
```

```
'4.3.2', 'python': '3.11.4 (tags/v3.11.4:d2340ef, Jun  7 2023, 05:45:37) [MSC
v.1934 64 bit (AMD64)]', 'platform': 'Windows-10-10.0.22631-SP0', 'event':
'load_word2vec_format'}
2024-04-29 12:25:45,878 - glove2word2vec - INFO - converting 400000 vectors from
glove.6B.100d.txt to glove.6B.100d.word2vec.txt
2024-04-29 12:25:46,123 - keyedvectors - INFO - storing 400000x100 projection
weights into glove.6B.100d.word2vec.txt
2024-04-29 12:26:19,349 - glove2word2vec - INFO - Converted model with 400000
vectors and 100 dimensions
```

[26]:
```
model_glove = KeyedVectors.load_word2vec_format('glove.6B.100d.word2vec.txt',␣
↪binary=False)
```

1.0.10 FastText

[29]:
```
model_fasttext = FastText(X_train_tokens, vector_size=200, window=5,␣
↪min_count=1, workers=4)
```

1.1 H. Training / Testing

1.1.1 Visualization

[30]:
```
from sklearn.decomposition import PCA

#pass the embeddings to PCA
pca = PCA(n_components=2)
result = pca.fit_transform(model_w2v_sg.wv.vectors)

#create df from the pca results
pca_df = pd.DataFrame(result, columns = ['x','y'])

#add the words for the hover effect
pca_df['word'] = model_w2v_sg.wv.index_to_key
pca_df.head()
```

[30]:
```
          x         y word
0  0.991814 -0.135516  the
1  0.979402  0.037267   of
2  0.883758 -0.061537  and
3  1.301585  0.877599   to
4  1.027505  0.035662    a
```

[31]:
```
import plotly.graph_objs as go

N = 1000000
words = list(model_w2v_sg.wv.index_to_key)
fig = go.Figure(data=go.Scattergl(
```

```
    x = pca_df['x'],
    y = pca_df['y'],
    mode='markers',
    marker=dict(
        color=np.random.randn(N),
        colorscale='Viridis',
        line_width=1
    ),
    text=pca_df['word'],
    textposition="bottom center"
))

fig.show()
```

1.1.2 Vectorizing our sentence (Sentence --> Embedding Vector) by calculating the MEAN

[32]:
```
def get_mean_vector(w2v_vectors, words):
    words = [word for word in words if word in w2v_vectors]
    if words:
        avg_vector = np.mean(w2v_vectors[words], axis=0)
    else:
        avg_vector = np.zeros_like(w2v_vectors['hi'])
    return avg_vector
```

Skip-Gram

[33]:
```
import numpy as np
import pandas as pd

df_data = []
for token in X_train_tokens:
    df_data.append(get_mean_vector(model_w2v_sg.wv,token))

train_embeddings = pd.DataFrame(data={'Skip-Gram':df_data})

df_data = []
for token in X_test_tokens:
    df_data.append(get_mean_vector(model_w2v_sg.wv,token))

test_embeddings = pd.DataFrame(data={'Skip-Gram':df_data})
```

Cbow

[34]:
```
df_data = []
for token in X_train_tokens:
    df_data.append(get_mean_vector(model_w2v_cbow.wv,token))
```

```python
train_embeddings['Cbow'] = df_data

df_data = []
for token in X_test_tokens:
    df_data.append(get_mean_vector(model_w2v_cbow.wv,token))

test_embeddings['Cbow'] = df_data
```

FastText

[35]:
```python
df_data = []
for token in X_train_tokens:
    df_data.append(get_mean_vector(model_fasttext.wv,token))

train_embeddings['FastText'] = df_data

df_data = []
for token in X_test_tokens:
    df_data.append(get_mean_vector(model_fasttext.wv,token))

test_embeddings['FastText'] = df_data
```

[36]:
```python
train_embeddings['author'] = y_train
test_embeddings['author'] = y_test
```

[37]:
```python
train_embeddings
```

[37]:
```
                                               Skip-Gram  \
0      [-0.09245351, -0.13474467, -0.19422822, 0.0571…
1      [-0.0138421515, -0.06436534, -0.1446154, 0.008…
2      [-0.016322251, -0.05566424, -0.13252683, 0.013…
3      [0.042666577, -0.06912962, -0.14757802, 0.0211…
4      [0.030173779, -0.048087917, -0.13290967, 0.019…
…                                                   …
15658  [0.037498757, -0.076692045, -0.16398796, 0.024…
15659  [0.008315314, -0.055191915, -0.119723015, 0.03…
15660  [-0.003307657, -0.08177994, -0.12526338, 0.078…
15661  [-0.016894873, -0.02929884, -0.09387635, 0.052…
15662  [-0.016567785, -0.052925322, -0.13258567, 0.05…

                                                    Cbow  \
0      [0.076555416, -0.3873256, -0.21496993, 0.16724…
1      [0.04013775, -0.14664431, -0.24667382, 0.26135…
2      [0.047090776, -0.123674214, -0.24026245, 0.220…
3      [0.200636, -0.39847913, -0.19032557, 0.0736103…
4      [0.13835351, -0.24336997, -0.32883242, 0.23244…
…                                                   …
```

```
15658  [0.14120299, -0.2590958, -0.24012192, 0.126241…
15659  [0.11323017, -0.2953621, -0.25744775, 0.208803…
15660  [0.074559696, -0.2186441, -0.22983482, 0.24306…
15661  [0.044977337, -0.16403587, -0.26551414, 0.2932…
15662  [0.069187514, -0.12838382, -0.2805442, 0.25940…

                                                FastText  author
0      [-0.64794165, -0.2699343, 0.68553156, -0.05720…       0
1      [-0.450563, -0.1549467, 0.7243632, 0.004474306…       2
2      [-0.4520924, -0.12596405, 0.6855266, -0.043536…       2
3      [-0.8359748, -0.5553028, 0.88246626, -0.073657…       2
4      [-0.51334476, -0.28475353, 0.9356317, -0.01246…       2
…                                                    …     …
15658  [-0.55205256, -0.29024366, 0.69331557, -0.0324…       2
15659  [-0.60343003, -0.33598742, 0.8039362, -0.03032…       2
15660  [-0.5472723, -0.20986466, 0.68265975, -0.04442…       0
15661  [-0.44843885, -0.1873457, 0.7674887, -0.005266…       1
15662  [-0.5150041, -0.18791518, 0.8278089, -0.038176…       0

[15663 rows x 4 columns]
```

[41]:
```
test_embeddings
```

[41]:
```
                                              Skip-Gram  \
0     [-0.02533995, -0.057708997, -0.1383868, 0.0334…
1     [-0.013364219, -0.04435305, -0.115443155, 0.04…
2     [-0.038911216, -0.04846803, -0.10835905, 0.019…
3     [0.02638327, -0.042123396, -0.106831744, 0.055…
4     [0.0032943068, -0.035050638, -0.14190407, 0.04…
…                                                  …
3911  [0.011293744, -0.033249523, -0.13720067, 0.045…
3912  [-0.0038440789, -0.035394795, -0.12100706, 0.0…
3913  [0.02209774, -0.057479277, -0.11369388, -0.006…
3914  [-0.0031537581, -0.040496238, -0.09115279, 0.0…
3915  [0.010578151, -0.072131395, -0.16562384, 0.015…

                                                   Cbow  \
0     [0.08728299, -0.23488268, -0.24730948, 0.19595…
1     [0.1310399, -0.28475758, -0.23302014, 0.156321…
2     [0.041968066, -0.27902886, -0.19055352, 0.2374…
3     [0.13256206, -0.25670433, -0.25646424, 0.21515…
4     [0.09842654, -0.1773736, -0.29563868, 0.256905…
…                                                  …
3911  [0.082615644, -0.3048804, -0.23068282, 0.23892…
3912  [0.072965354, -0.24091473, -0.29348248, 0.3058…
3913  [0.10891265, -0.24729455, -0.28157568, 0.22512…
3914  [0.09743194, -0.21748036, -0.22617775, 0.15707…
```

```
3915  [0.1377265, -0.28232977, -0.23600832, 0.139908…

                                                FastText  author
0     [-0.5960142, -0.24000153, 0.7607671, -0.057615…       0
1     [-0.76362294, -0.40808642, 0.93084824, -0.0655…       0
2     [-0.4527249, -0.21626551, 0.5690407, 0.0437042…       1
3     [-0.5001691, -0.21661125, 0.6957603, -0.003744…       0
4     [-0.6131109, -0.24261205, 0.8562926, -0.080922…       0
…                                                  …     …
3911  [-0.6011191, -0.30326766, 0.7502945, -0.024315…       0
3912  [-0.5458445, -0.24774297, 0.8056308, -0.017394…       0
3913  [-0.49014255, -0.24898033, 0.73027086, -0.0010…       1
3914  [-0.529733, -0.24831183, 0.7210892, -0.0501421…       1
3915  [-0.6428726, -0.32382765, 0.76352257, -0.05812…       2

[3916 rows x 4 columns]
```

1.1.3 Train and Test

Skip-Gram with MLP's

[52]:
```
#Lets try to train our data using Skip-Gram
X_train =  pd.DataFrame(list(train_embeddings['Skip-Gram'])).
 ↪reset_index(drop=True)
y_train = pd.DataFrame(train_embeddings['author'])
```

[53]:
```
import tensorflow as tf
from tensorflow import keras
from keras.optimizers import Adam
from keras.losses import SparseCategoricalCrossentropy


mlp_model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=(200,)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(3, activation='softmax')
])

adam = Adam(learning_rate=0.01)
mlp_model.compile(loss=SparseCategoricalCrossentropy(), optimizer=adam,␣
 ↪metrics=['accuracy'])

#adam_history = model_ADAM.fit(X_train, y_train, epochs=50,␣
 ↪validation_data=(X_val, y_val))
skip_history = mlp_model.fit(X_train, y_train, epochs=50)
```

```
Epoch 1/50
490/490 [==============================] - 2s 2ms/step - loss: 0.8887 -
accuracy: 0.5877
Epoch 2/50
490/490 [==============================] - 1s 1ms/step - loss: 0.8300 -
accuracy: 0.6326
Epoch 3/50
490/490 [==============================] - 1s 2ms/step - loss: 0.8128 -
accuracy: 0.6427
Epoch 4/50
490/490 [==============================] - 1s 1ms/step - loss: 0.8029 -
accuracy: 0.6450
Epoch 5/50
490/490 [==============================] - 1s 1ms/step - loss: 0.7897 -
accuracy: 0.6550
Epoch 6/50
490/490 [==============================] - 1s 1ms/step - loss: 0.7900 -
accuracy: 0.6543
Epoch 7/50
490/490 [==============================] - 1s 1ms/step - loss: 0.7818 -
accuracy: 0.6656
Epoch 8/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7699 -
accuracy: 0.6665
Epoch 9/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7688 -
accuracy: 0.6665
Epoch 10/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7625 -
accuracy: 0.6686
Epoch 11/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7558 -
accuracy: 0.6729
Epoch 12/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7605 -
accuracy: 0.6710
Epoch 13/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7480 -
accuracy: 0.6749
Epoch 14/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7517 -
accuracy: 0.6761
Epoch 15/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7482 -
accuracy: 0.6780
Epoch 16/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7409 -
accuracy: 0.6828
```

```
Epoch 17/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7418 -
accuracy: 0.6782
Epoch 18/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7402 -
accuracy: 0.6774
Epoch 19/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7391 -
accuracy: 0.6794
Epoch 20/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7339 -
accuracy: 0.6881
Epoch 21/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7368 -
accuracy: 0.6813
Epoch 22/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7317 -
accuracy: 0.6861
Epoch 23/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7286 -
accuracy: 0.6853
Epoch 24/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7276 -
accuracy: 0.6875
Epoch 25/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7333 -
accuracy: 0.6875
Epoch 26/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7270 -
accuracy: 0.6875
Epoch 27/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7250 -
accuracy: 0.6884
Epoch 28/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7266 -
accuracy: 0.6915
Epoch 29/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7236 -
accuracy: 0.6907
Epoch 30/50
490/490 [==============================] - 1s 3ms/step - loss: 0.7251 -
accuracy: 0.6886
Epoch 31/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7233 -
accuracy: 0.6922
Epoch 32/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7246 -
accuracy: 0.6897
```

```
Epoch 33/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7196 -
accuracy: 0.6953
Epoch 34/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7195 -
accuracy: 0.6907
Epoch 35/50
490/490 [==============================] - 1s 3ms/step - loss: 0.7186 -
accuracy: 0.6934
Epoch 36/50
490/490 [==============================] - 1s 3ms/step - loss: 0.7167 -
accuracy: 0.6899
Epoch 37/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7176 -
accuracy: 0.6947
Epoch 38/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7195 -
accuracy: 0.6909
Epoch 39/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7177 -
accuracy: 0.6935
Epoch 40/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7120 -
accuracy: 0.6957
Epoch 41/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7117 -
accuracy: 0.6965
Epoch 42/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7187 -
accuracy: 0.6930
Epoch 43/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7090 -
accuracy: 0.6990
Epoch 44/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7129 -
accuracy: 0.6967
Epoch 45/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7097 -
accuracy: 0.6985
Epoch 46/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7098 -
accuracy: 0.6975
Epoch 47/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7096 -
accuracy: 0.6977
Epoch 48/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7061 -
accuracy: 0.6992
```

```
Epoch 49/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7053 -
accuracy: 0.6995
Epoch 50/50
490/490 [==============================] - 1s 2ms/step - loss: 0.7079 -
accuracy: 0.6999
```

Cbow with MLP's

[44]:
```
#Lets try to train our data using Skip-Gram
X_train =  pd.DataFrame(list(train_embeddings['Cbow'])).reset_index(drop=True)
y_train = pd.DataFrame(train_embeddings['author'])
```

[45]:
```
import tensorflow as tf
from tensorflow import keras
from keras.optimizers import Adam
from keras.losses import SparseCategoricalCrossentropy


mlp_model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=(200,)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(3, activation='softmax')
])

adam = Adam(learning_rate=0.01)
mlp_model.compile(loss=SparseCategoricalCrossentropy(), optimizer=adam,
 ↪metrics=['accuracy'])

#adam_history = model_ADAM.fit(X_train, y_train, epochs=50,
 ↪validation_data=(X_val, y_val))
cbow_history = mlp_model.fit(X_train, y_train, epochs=50)
```

```
Epoch 1/50
490/490 [==============================] - 2s 2ms/step - loss: 1.0407 -
accuracy: 0.4566
Epoch 2/50
490/490 [==============================] - 1s 1ms/step - loss: 1.0166 -
accuracy: 0.4750
Epoch 3/50
490/490 [==============================] - 1s 2ms/step - loss: 1.0069 -
accuracy: 0.4877
Epoch 4/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9998 -
accuracy: 0.4979
Epoch 5/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9910 -
```

```
accuracy: 0.5090
Epoch 6/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9920 -
accuracy: 0.5055
Epoch 7/50
490/490 [==============================] - 1s 3ms/step - loss: 0.9846 -
accuracy: 0.5106
Epoch 8/50
490/490 [==============================] - 1s 3ms/step - loss: 0.9843 -
accuracy: 0.5147
Epoch 9/50
490/490 [==============================] - 1s 3ms/step - loss: 0.9794 -
accuracy: 0.5154
Epoch 10/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9780 -
accuracy: 0.5193
Epoch 11/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9746 -
accuracy: 0.5211
Epoch 12/50
490/490 [==============================] - 1s 3ms/step - loss: 0.9712 -
accuracy: 0.5218
Epoch 13/50
490/490 [==============================] - 1s 3ms/step - loss: 0.9742 -
accuracy: 0.5219
Epoch 14/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9737 -
accuracy: 0.5237
Epoch 15/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9693 -
accuracy: 0.5264
Epoch 16/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9677 -
accuracy: 0.5277
Epoch 17/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9712 -
accuracy: 0.5257
Epoch 18/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9658 -
accuracy: 0.5293
Epoch 19/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9668 -
accuracy: 0.5279
Epoch 20/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9664 -
accuracy: 0.5339
Epoch 21/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9646 -
```

```
accuracy: 0.5337
Epoch 22/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9649 -
accuracy: 0.5350
Epoch 23/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9652 -
accuracy: 0.5339
Epoch 24/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9621 -
accuracy: 0.5343
Epoch 25/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9612 -
accuracy: 0.5373
Epoch 26/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9629 -
accuracy: 0.5367
Epoch 27/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9601 -
accuracy: 0.5389
Epoch 28/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9581 -
accuracy: 0.5383
Epoch 29/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9631 -
accuracy: 0.5341
Epoch 30/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9572 -
accuracy: 0.5435
Epoch 31/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9558 -
accuracy: 0.5433
Epoch 32/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9604 -
accuracy: 0.5353
Epoch 33/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9592 -
accuracy: 0.5403
Epoch 34/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9524 -
accuracy: 0.5413
Epoch 35/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9585 -
accuracy: 0.5416
Epoch 36/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9548 -
accuracy: 0.5388
Epoch 37/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9536 -
```

```
accuracy: 0.5436
Epoch 38/50
490/490 [==============================] - 1s 3ms/step - loss: 0.9533 -
accuracy: 0.5423
Epoch 39/50
490/490 [==============================] - 1s 3ms/step - loss: 0.9543 -
accuracy: 0.5420
Epoch 40/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9533 -
accuracy: 0.5452
Epoch 41/50
490/490 [==============================] - 1s 3ms/step - loss: 0.9516 -
accuracy: 0.5445
Epoch 42/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9528 -
accuracy: 0.5442
Epoch 43/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9511 -
accuracy: 0.5457
Epoch 44/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9504 -
accuracy: 0.5441
Epoch 45/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9472 -
accuracy: 0.5459
Epoch 46/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9507 -
accuracy: 0.5440
Epoch 47/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9471 -
accuracy: 0.5463
Epoch 48/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9476 -
accuracy: 0.5493
Epoch 49/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9482 -
accuracy: 0.5469
Epoch 50/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9485 -
accuracy: 0.5473
```

FastText with MLP's

[46]:
```
#Lets try to train our data using Skip-Gram
X_train =  pd.DataFrame(list(train_embeddings['FastText'])).
 ↪reset_index(drop=True)
y_train = pd.DataFrame(train_embeddings['author'])
```

[47]:
```
import tensorflow as tf
from tensorflow import keras
from keras.optimizers import Adam
from keras.losses import SparseCategoricalCrossentropy


mlp_model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=(200,)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(3, activation='softmax')
])

adam = Adam(learning_rate=0.01)
mlp_model.compile(loss=SparseCategoricalCrossentropy(), optimizer=adam,␣
 ↪metrics=['accuracy'])

#adam_history = model_ADAM.fit(X_train, y_train, epochs=50,␣
 ↪validation_data=(X_val, y_val))
fasttext_history = mlp_model.fit(X_train, y_train, epochs=50)
```

```
Epoch 1/50
490/490 [==============================] - 3s 4ms/step - loss: 1.0596 -
accuracy: 0.4386
Epoch 2/50
490/490 [==============================] - 1s 3ms/step - loss: 1.0309 -
accuracy: 0.4731
Epoch 3/50
490/490 [==============================] - 1s 3ms/step - loss: 1.0265 -
accuracy: 0.4751
Epoch 4/50
490/490 [==============================] - 1s 3ms/step - loss: 1.0179 -
accuracy: 0.4830
Epoch 5/50
490/490 [==============================] - 1s 2ms/step - loss: 1.0179 -
accuracy: 0.4846
Epoch 6/50
490/490 [==============================] - 1s 2ms/step - loss: 1.0160 -
accuracy: 0.4847
Epoch 7/50
490/490 [==============================] - 1s 2ms/step - loss: 1.0124 -
accuracy: 0.4905
Epoch 8/50
490/490 [==============================] - 1s 2ms/step - loss: 1.0119 -
accuracy: 0.4926
Epoch 9/50
490/490 [==============================] - 1s 2ms/step - loss: 1.0118 -
```

```
accuracy: 0.4918
Epoch 10/50
490/490 [==============================] - 1s 3ms/step - loss: 1.0099 -
accuracy: 0.4915
Epoch 11/50
490/490 [==============================] - 1s 3ms/step - loss: 1.0084 -
accuracy: 0.4913
Epoch 12/50
490/490 [==============================] - 1s 2ms/step - loss: 1.0099 -
accuracy: 0.4928
Epoch 13/50
490/490 [==============================] - 1s 2ms/step - loss: 1.0072 -
accuracy: 0.4905
Epoch 14/50
490/490 [==============================] - 1s 2ms/step - loss: 1.0079 -
accuracy: 0.4925
Epoch 15/50
490/490 [==============================] - 1s 2ms/step - loss: 1.0071 -
accuracy: 0.4932
Epoch 16/50
490/490 [==============================] - 1s 2ms/step - loss: 1.0068 -
accuracy: 0.4931
Epoch 17/50
490/490 [==============================] - 1s 2ms/step - loss: 1.0068 -
accuracy: 0.4958
Epoch 18/50
490/490 [==============================] - 1s 2ms/step - loss: 1.0063 -
accuracy: 0.4956
Epoch 19/50
490/490 [==============================] - 1s 2ms/step - loss: 1.0045 -
accuracy: 0.4961
Epoch 20/50
490/490 [==============================] - 1s 2ms/step - loss: 1.0058 -
accuracy: 0.4961
Epoch 21/50
490/490 [==============================] - 1s 2ms/step - loss: 1.0076 -
accuracy: 0.4944
Epoch 22/50
490/490 [==============================] - 1s 2ms/step - loss: 1.0046 -
accuracy: 0.4988
Epoch 23/50
490/490 [==============================] - 1s 2ms/step - loss: 1.0051 -
accuracy: 0.5004
Epoch 24/50
490/490 [==============================] - 1s 2ms/step - loss: 1.0020 -
accuracy: 0.4992
Epoch 25/50
490/490 [==============================] - 2s 3ms/step - loss: 1.0010 -
```

```
accuracy: 0.4983
Epoch 26/50
490/490 [==============================] - 1s 3ms/step - loss: 1.0015 -
accuracy: 0.4986
Epoch 27/50
490/490 [==============================] - 2s 4ms/step - loss: 1.0004 -
accuracy: 0.5014
Epoch 28/50
490/490 [==============================] - 1s 3ms/step - loss: 1.0002 -
accuracy: 0.4975
Epoch 29/50
490/490 [==============================] - 1s 2ms/step - loss: 1.0034 -
accuracy: 0.5012
Epoch 30/50
490/490 [==============================] - 1s 2ms/step - loss: 1.0009 -
accuracy: 0.4993
Epoch 31/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9990 -
accuracy: 0.5034
Epoch 32/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9996 -
accuracy: 0.5032
Epoch 33/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9983 -
accuracy: 0.5049
Epoch 34/50
490/490 [==============================] - 1s 3ms/step - loss: 0.9985 -
accuracy: 0.5011
Epoch 35/50
490/490 [==============================] - 1s 2ms/step - loss: 1.0009 -
accuracy: 0.5030
Epoch 36/50
490/490 [==============================] - 1s 3ms/step - loss: 0.9981 -
accuracy: 0.5018
Epoch 37/50
490/490 [==============================] - 1s 3ms/step - loss: 0.9998 -
accuracy: 0.4986
Epoch 38/50
490/490 [==============================] - 1s 3ms/step - loss: 1.0008 -
accuracy: 0.5012
Epoch 39/50
490/490 [==============================] - 1s 3ms/step - loss: 0.9976 -
accuracy: 0.5047
Epoch 40/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9969 -
accuracy: 0.5037
Epoch 41/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9972 -
```

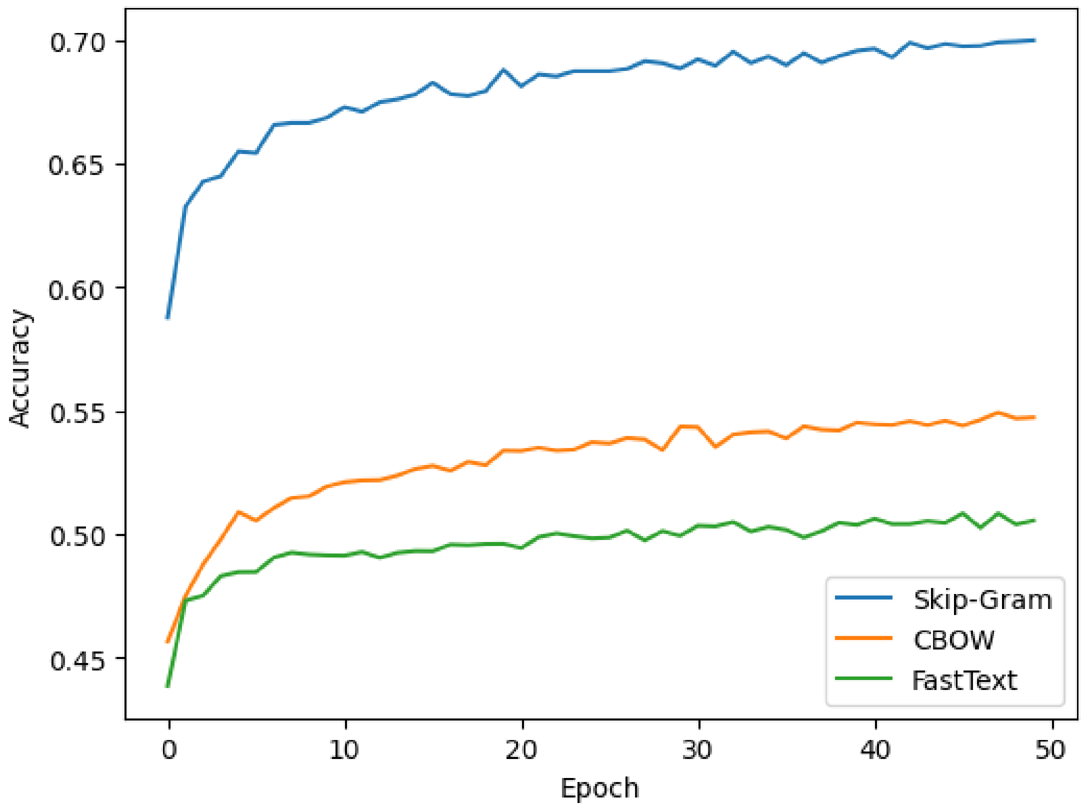
```
accuracy: 0.5063
Epoch 42/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9962 -
accuracy: 0.5041
Epoch 43/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9960 -
accuracy: 0.5041
Epoch 44/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9962 -
accuracy: 0.5053
Epoch 45/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9959 -
accuracy: 0.5046
Epoch 46/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9934 -
accuracy: 0.5084
Epoch 47/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9974 -
accuracy: 0.5026
Epoch 48/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9949 -
accuracy: 0.5084
Epoch 49/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9957 -
accuracy: 0.5040
Epoch 50/50
490/490 [==============================] - 1s 2ms/step - loss: 0.9966 -
accuracy: 0.5055
```

1.1.4 Results

[54]:
```
# Plot the training history for each optimizer
import matplotlib.pyplot as plt

plt.plot(pd.DataFrame(skip_history.history)[['accuracy']], label='Skip-Gram')
plt.plot(pd.DataFrame(cbow_history.history)[['accuracy']], label='CBOW')
plt.plot(pd.DataFrame(fasttext_history.history)[['accuracy',]],
 ↪label='FastText')

plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



0.70
0.65
0.60
0.55
0.50
0.45
Accuracy
0
10
20
30
40
50
Epoch
Skip-Gram
CBOW
FastText