# Assignment 02

## Assignment Overview: Clustering and Classification

### Part 1: Clustering

1. **Objective**:
   - Group the documents (articles) using clustering algorithms and evaluate the results.
2. **Input**:
   - Four document matrices per vectorization technique (Doc2Vec, BERT, Sentence-BERT), each with dimensions (100, 600).
3. **Tasks**:
   - Combine the four matrices into a single matrix for each technique.
   - Apply clustering using:
     - **K-Means** (with `k=4` for 4 journals).
     - **DBSCAN** (select `eps` and `min_samples` heuristically).
     - **Gaussian Mixture Model**.
   - Evaluate the clusters using:
     - Metrics: Precision, Recall, F1-Score, Accuracy.
     - Visualization: Use UMAP, t-SNE, or other tools (e.g., Seaborn).

### Part 2: Classification

1. **Objective**:
   - Build classifiers to predict the journal group.
2. **Algorithms**:
   - **Artificial Neural Network (ANN)** (two architectures provided):
     - ANN Architecture 1: RELU activation layers.
     - ANN Architecture 2: GELU activation layers.
   - **Other Classifiers**: Naive Bayes (NB), Support Vector Machine (SVM), Logistic Regression (LoR), Random Forest (RF).
3. **Tasks**:
   - Perform 10-fold cross-validation for all classifiers (except ANN).
   - Identify and rank the top 20 most important features for NB, RF, SVM, LoR.
   - Write explanations for feature importance in a README document and include the ranked lists in an Excel file.
4. **ANN Specifics**:
   - Split data: Train (80%, with 10% validation from the train set) and Test (20%).
   - Use the given ANN architectures with specific configurations:
     - Maximum 15 epochs.
     - Batch size: 32.
     - Early stopping after 3 validation iterations without improvement.
     - Save the best model (ModelCheckpoint).

### Deliverables

1. Python code files.
2. A detailed README with explanations, tables, plots, and insights.
3. Excel files with feature importance rankings.
4. All files zipped into a directory named after the student group.

# word2vec - Example

# Clustering

## Objective

Take the document (meaning vector representation of the document, the output of the last assignment `Doc2Vec`, `BERT`, `Sentence-BERT`, times 4 group from each) and cluster them into groups and compare the results with the actual division form each publication.

# Input

4 Groups of matrices each line represent a document in it's vector form, from:

1. `Doc2Vec`
2. `BERT`
3. `Sentence-BERT`

## Task

- Combine the four matrices into a single matrix for each technique.
- Apply clustering using:
- **K-Means** (with `k=4` for 4 journals).
- **DBSCAN** (select `eps` and `min_samples` heuristically).
- **Gaussian Mixture Model**.
- Evaluate the clusters using:
- Metrics: Precision, Recall, F1-Score, Accuracy.
- Visualization: Use UMAP, t-SNE, or other tools (e.g., Seaborn).

## Output

- The plot of the real clusters vs. the clusters from the 3 methods mentioned above.
- The metrics for each clustering method.

# Word2Vec Matrices

We have 4 `.csv` files with each $(100, \approx 600)$ and we need to combine them into one big matrix and then cluster them.

> Note that the code is the same for all other matrices

# Plan

1. Download the files from my `GitHub`.
2. Add a `'cluster'` column for each file (`=0` for AJ etc...).
3. Cluster with `Kmeans` for `k=4`.
4. Write a function to find the right parameters for 4 clusters for `DBSCAN` (i.e. the `eps` and `min_samples` parameters).
5. Cluster with `DBSCAN`
6. Cluster with `GMM`
7. Use `t-SNE` to visualize the cluster in $\mathbb{R}^2$
8. Output the plot for each clustering methods + original
9. Measure each method using the metric mentioned above.

```
import warnings

warnings.filterwarnings("ignore")
```

```
base_url = "https://raw.githubusercontent.com/dattali18/IR_Assignments/refs/heads/main/Assig

file_names = ["aj", "bbc", "jp", "nyt"]

cluster_map =  {'aj' : 0, 'bbc': 1, 'jp' : 2, 'nyt': 3}

links = [f"{base_url}/{name}_doc2vec.csv" for name in file_names]
```

```
import pandas as pd

dfs = {}

for name, link in zip(file_names, links):
df = pd.read_csv(link)
# take all the col from 0 - 99 and put them into a numpy array
df_cpy = pd.DataFrame()
df_cpy['vector'] = df.iloc[:, :100].to_numpy().tolist()
df_cpy["cluster"] = cluster_map[name]
dfs[name] = df_cpy
```

```
dfs['aj'].head()
```

| | vector | cluster |
|---|---|---|
| **0** | [-0.16861272, 0.13619465, 0.118086584, 0.04930... | 0 |
| **1** | [-0.1352571, 0.11357225, 0.09048813, 0.0408173... | 0 |
| **2** | [-0.061645806, 0.055493645, 0.045198712, 0.019... | 0 |
| **3** | [-0.11881534, 0.09667818, 0.07898884, 0.034127... | 0 |
| **4** | [-0.04435978, 0.04531823, 0.03259423, 0.011060... | 0 |

```
# merge all of the df into one df

df = pd.concat(dfs.values(), ignore_index=True)
```

```
# standerdize the data mean=0 std=1

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

#apply to each line of the df

df['std_vector'] = df['vector'].apply(lambda x: scaler.fit_transform(np.array(x).reshape(-1,
df['std_vector'].head()
```

| | std_vector |
|---|---|
| **0** | [-0.9614479972007641, 0.6957987565937652, 0.59... |
| **1** | [-0.9792962849264775, 0.7375863705340073, 0.57... |
| **2** | [-0.9016816118502328, 0.7442563640622617, 0.59... |
| **3** | [-1.0004149361896462, 0.7354988385819545, 0.59... |
| **4** | [-0.8834392389517144, 0.8111157942320139, 0.57... |

**dtype:** object

```
from sklearn.cluster import KMeans, DBSCAN
from sklearn.mixture import GaussianMixture

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns
```

```
# visualize the real cluster using t-SNE

from sklearn.manifold import TSNE

tsne = TSNE(n_components=2, random_state=0)

# transofrm the df['vector'] to dataframe with freatuer 0 - 99 for
df_copy = df["std_vector"].apply(pd.Series)

df_tsne = tsne.fit_transform(df_copy)

df_tsne = pd.DataFrame(df_tsne, columns=["x", "y"])

df_tsne["cluster"] = df["cluster"]
```

```
# plot the df_tsne

reverse_cluster_map = {v: k for k, v in cluster_map.items()}

plt.figure(figsize=(10, 10))
# add labels
plt.title('Real Clustering')

# make color scheme red, blue, green etc


df_tsne['cluster'] = df_tsne['cluster'].map(reverse_cluster_map)

sns.scatterplot(data=df_tsne, x='x', y='y', hue='cluster')

plt.show()
```


## Kmeans

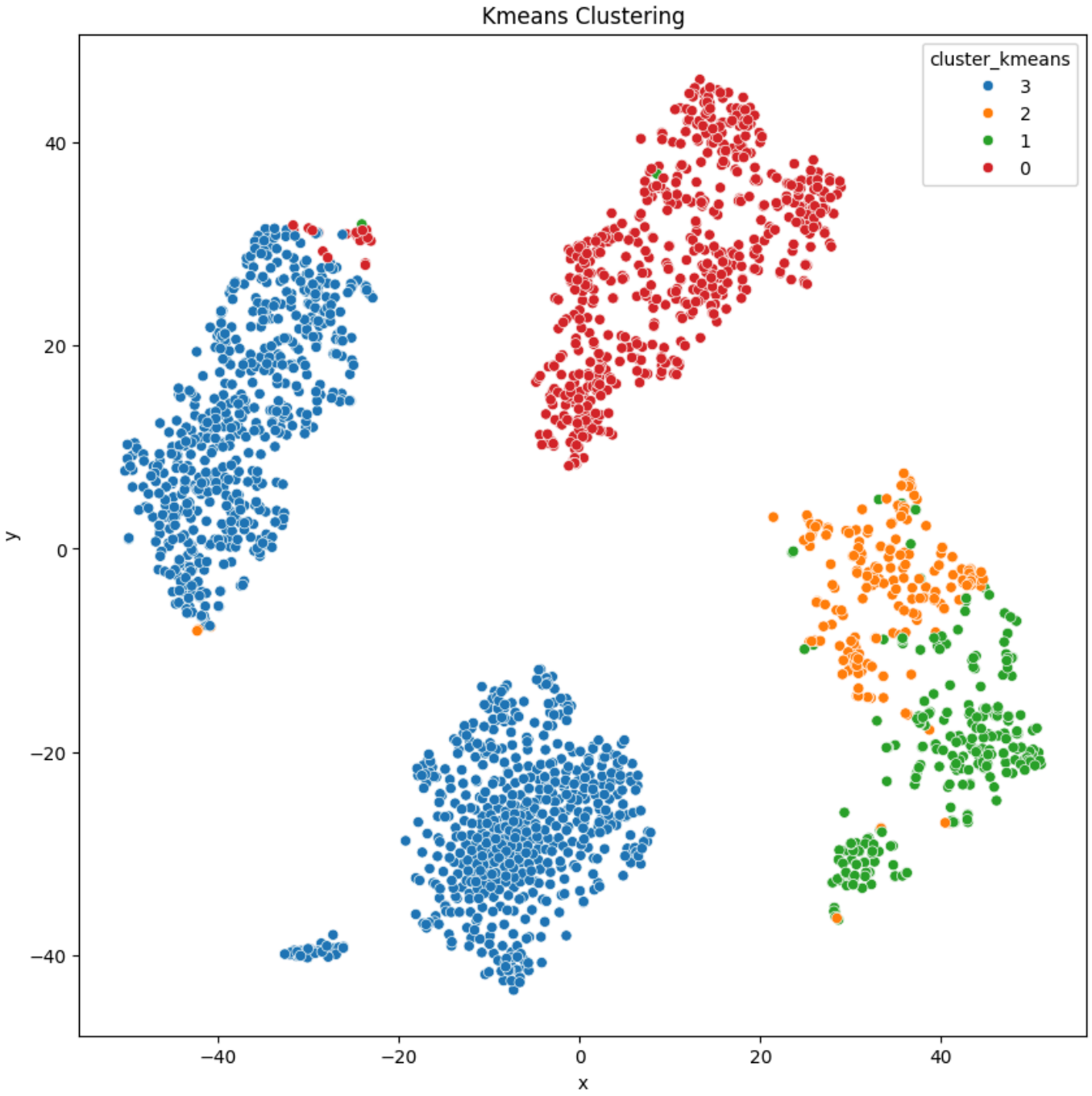
```
kmeans = KMeans(n_clusters=4, random_state=0).fit(df['std_vector'].tolist())

df['cluster_kmeans'] = kmeans.labels_

df['cluster_kmeans'].head()
```

| | cluster_kmeans |
|---|---|
| **0** | 3 |
| **1** | 3 |
| **2** | 3 |

| | cluster_kmeans |
|---|---|
| **3** | 3 |
| **4** | 3 |

**dtype:** int32

```
# visutalize the cluster using the t-SNE df

df_tsne['cluster_kmeans'] = df['cluster_kmeans']

reverse_cluster_map = {v: k for k, v in cluster_map.items()}

plt.figure(figsize=(10, 10))

plt.title('Kmeans Clustering')

# make color scheme red, blue, green etc


df_tsne['cluster_kmeans'] = df_tsne['cluster_kmeans'].map(str)

sns.scatterplot(data=df_tsne, x='x', y='y', hue='cluster_kmeans')

plt.show()
```




## DBSCAN

For choosing the right params for `eps`, and `min_sample` we use some heuristics

1. For `eps` we used **KNN** and `keed` to find the best best parameter
2. For `min_sample` we choose the heuristic of $2 \times \text{Attributes}$

```
import numpy as np
import pandas as pd
from sklearn.neighbors import NearestNeighbors
from kneed import KneeLocator
from sklearn.cluster import DBSCAN

def get_parameters(df, num_clusters=4, eps_adjustment=1.0, min_samples_adjustment=1):
```

```
X = np.array(df)

# Use NearestNeighbors to find the nearest neighbors
neighbors = NearestNeighbors(n_neighbors=2 * X.shape[1] - 1)
neighbors_fit = neighbors.fit(X)
distances, indices = neighbors_fit.kneighbors(X)
distances = np.sort(distances, axis=0)
distances = distances[:, 1]

# Use KneeLocator to find the "elbow" point in the k-distance graph
kneedle = KneeLocator(range(len(distances)), distances, S=1.0, curve="convex", direction="in
eps = distances[kneedle.elbow] * eps_adjustment

# Set min_samples to 2 * dimensions, another common heuristic
min_samples = 2 * X.shape[1] * min_samples_adjustment

return eps, min_samples

def find_best_parameters(df, num_clusters=4):
best_eps = None
best_min_samples = None
best_num_clusters = 0

for eps_adjustment in np.arange(0.5, 2.0, 0.1):
    for min_samples_adjustment in range(1, 5):
        eps, min_samples = get_parameters(df, num_clusters, eps_adjustment, min_samples_adju
        db = DBSCAN(eps=eps, min_samples=min_samples).fit(df)
        labels = db.labels_
        num_clusters_found = len(set(labels)) - (1 if -1 in labels else 0)

        if num_clusters_found == num_clusters:
            return eps, min_samples

        if num_clusters_found > best_num_clusters:
            best_eps = eps
            best_min_samples = min_samples
            best_num_clusters = num_clusters_found

return best_eps, best_min_samples

eps, min_samples = find_best_parameters(df['std_vector'].tolist(), num_clusters=4)
print(f"Best eps: {eps}, Best min_samples: {min_samples}")
```

```
Best eps: 2.744350773278325, Best min_samples: 200
```

```
# use DBSCAN

dbscan = DBSCAN(eps=eps, min_samples=min_samples).fit(df['std_vector'].tolist())

df['cluster_dbscan'] = dbscan.labels_
```

```
# count the number of cluster

num_clusters = len(df['cluster_dbscan'].unique())

print(f"Number of clusters: {num_clusters}")
```

```
Number of clusters: 3
```

```
# visualize data

# visutalize the cluster using the t-SNE df

df_tsne['cluster_dbscan'] = df['cluster_dbscan']

reverse_cluster_map = {v: k for k, v in cluster_map.items()}

plt.figure(figsize=(10, 10))

plt.title('DBSCAN Clustering')

# make color scheme red, blue, green etc


df_tsne['cluster_dbscan'] = df_tsne['cluster_dbscan'].map(str)

sns.scatterplot(data=df_tsne, x='x', y='y', hue='cluster_dbscan')

plt.show()
```

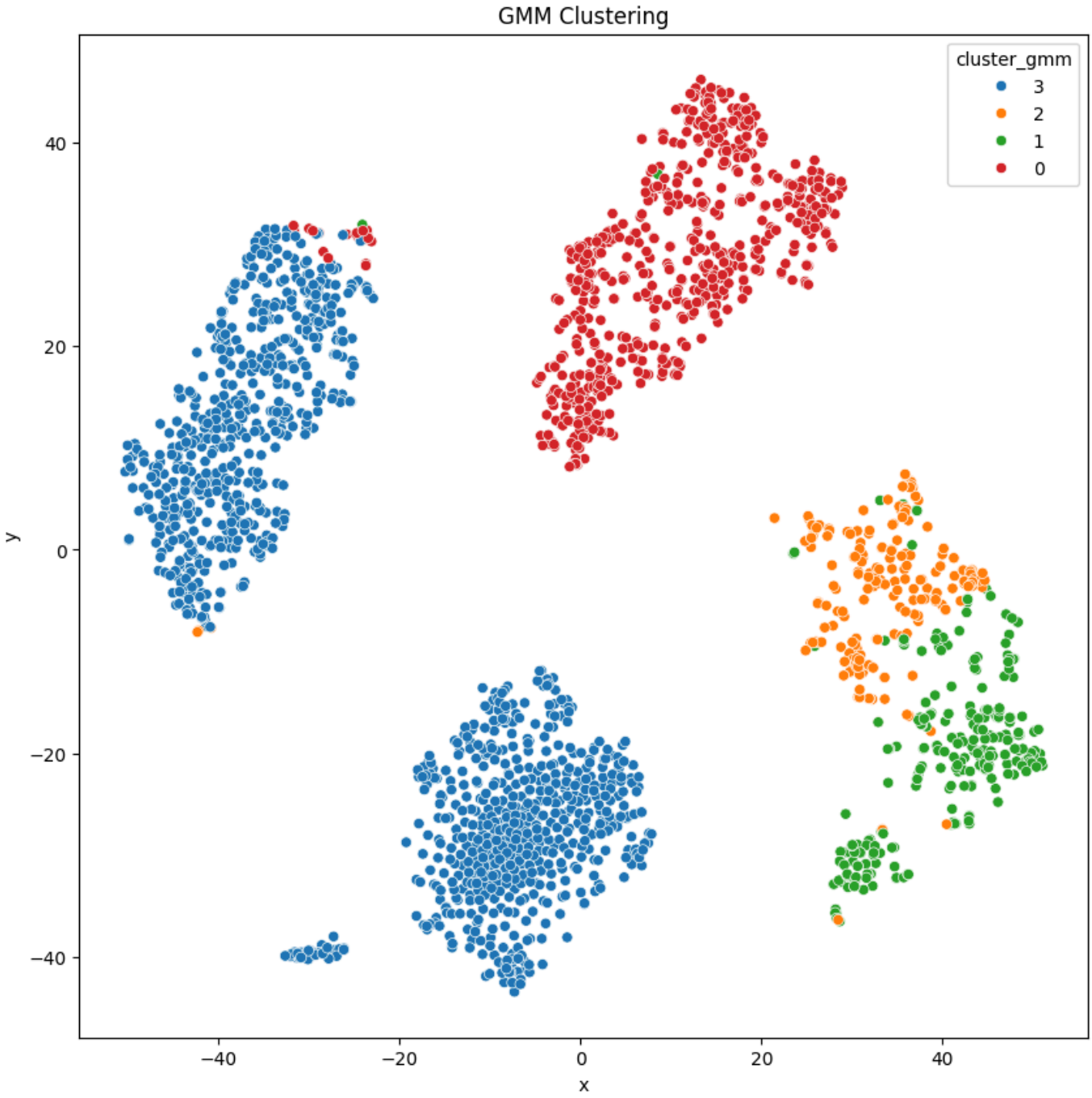

## GMM

```
# apply GMM

gmm = GaussianMixture(n_components=4, random_state=0).fit(df['std_vector'].tolist())

df['cluster_gmm'] = gmm.predict(df['std_vector'].tolist())

df['cluster_gmm'].head()
```

| | cluster_gmm |
|---|---|
| **0** | 3 |
| **1** | 3 |
| **2** | 3 |
| **3** | 3 |
| **4** | 3 |

**dtype:** int64

```
# visualize data

# visutalize the cluster using the t-SNE df

df_tsne['cluster_gmm'] = df['cluster_gmm']

reverse_cluster_map = {v: k for k, v in cluster_map.items()}

plt.figure(figsize=(10, 10))

plt.title('GMM Clustering')

# make color scheme red, blue, green etc


df_tsne['cluster_gmm'] = df_tsne['cluster_gmm'].map(str)

sns.scatterplot(data=df_tsne, x='x', y='y', hue='cluster_gmm')

plt.show()
```




## Measurements

For measuring we thought of some heuristic to help us better score them

1. Counting - but this didn't work
2. Using KNN and then giving a name, but still didn't work

So we chose to just use simple real vs. prediction without any heuristic.

The better way to judge was visually using T-SNE but for some of the matrices it was not possible to judge because we didn't have distinct blobs.

```
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
```

```
def evluate_model(real, pred):
precision = precision_score(real, pred, average='macro')
recall = recall_score(real, pred, average='macro')
f1 = f1_score(real, pred, average='macro')
accuracy = accuracy_score(real, pred)

return precision, recall, f1, accuracy
```

```
kmeans_evalutation_df = pd.DataFrame(columns=['precision', 'recall', 'f1', 'accuracy'])

kmeans_evalutation_df.loc['kmeans'] = evluate_model(df['cluster'], df['cluster_kmeans'])

kmeans_evalutation_df
```

| | precision | recall | f1 | accuracy |
|---|---|---|---|---|
| **kmeans** | 0.37066 | 0.374432 | 0.337265 | 0.370844 |

```
# same for DBSCAN but map -1 to 2
df['cluster_dbscan'] = df['cluster_dbscan'].map(lambda x: x if x != -1 else 2)

dbscan_evalutation_df = pd.DataFrame(columns=['precision', 'recall', 'f1', 'accuracy'])

dbscan_evalutation_df.loc['dbscan'] = evluate_model(df['cluster'], df['cluster_dbscan'])

dbscan_evalutation_df
```

| | precision | recall | f1 | accuracy |
|---|---|---|---|---|
| **dbscan** | 0.362509 | 0.498331 | 0.404344 | 0.508951 |

```
# same for GMM

gmm_evalutation_df = pd.DataFrame(columns=['precision', 'recall', 'f1', 'accuracy'])

gmm_evalutation_df.loc['gmm'] = evluate_model(df['cluster'], df['cluster_gmm'])

gmm_evalutation_df
```

| | precision | recall | f1 | accuracy |
|---|---|---|---|---|
| **gmm** | 0.370775 | 0.375305 | 0.337838 | 0.371697 |

## Summary

| | Precision | Recall | F1-Score | Accuracy |
|---|---|---|---|---|
| KMeans | 0.37066 | 0.374432 | 0.337265 | 0.370844 |
| DBSCAN | 0.362509 | 0.498331 | 0.404344 | 0.508951 |
| GMM | 0.370775 | 0.375305 | 0.337838 | 0.371697 |

# word2vec

> Note - the idea is the same for any other matrices

# Classification

## Objective:

- Build classifiers to predict the journal group.

## Algorithms:

- **Artificial Neural Network (ANN)** (two architectures provided):
  - ANN Architecture 1: RELU activation layers.
  - ANN Architecture 2: GELU activation layers.
- **Other Classifiers**: Naive Bayes (NB), Support Vector Machine (SVM), Logistic Regression (LoR), Random Forest (RF).

## Tasks:

- Perform 10-fold cross-validation for all classifiers (except ANN).
- Identify and rank the top 20 most important features for NB, RF, SVM, LoR.
- Write explanations for feature importance in a README document and include the ranked lists in an Excel file.
- Check what is the top 20 most important features for the ANN models.

## ANN Specifics:

- Split data: Train (80%, with 10% validation from the train set) and Test (20%).
- Use the given ANN architectures with specific configurations:
  - Maximum 15 epochs.
  - Batch size: 32.
  - Early stopping after 3 validation iterations without improvement.
  - Save the best model (ModelCheckpoint).

```
import warnings

warnings.filterwarnings("ignore")
```

```
base_url = "https://raw.githubusercontent.com/dattali18/IR_Assignments/refs/heads/main/Assig

file_names = ["aj", "bbc", "jp", "nyt"]

cluster_map =  {'aj' : 0, 'bbc': 1, 'jp' : 2, 'nyt': 3}

links = [f"{base_url}/{name}_doc2vec.csv" for name in file_names]
```

```
import pandas as pd

dfs = {}

for name, link in zip(file_names, links):
df = pd.read_csv(link)
# take all the col from 0 - 99 and put them into a numpy array
df_cpy = pd.DataFrame()
df_cpy['vector'] = df.iloc[:, :100].to_numpy().tolist()
df_cpy["cluster"] = str(cluster_map[name])
dfs[name] = df_cpy
```

```
# merge all of the df into one df

df = pd.concat(dfs.values(), ignore_index=True)
```

```
# standerdize the data mean=0 std=1
import numpy as np

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

# apply to each line of the df

df['std_vector'] = df['vector'].apply(lambda x: scaler.fit_transform(np.array(x).reshape(-1,
df['std_vector'].head()
```

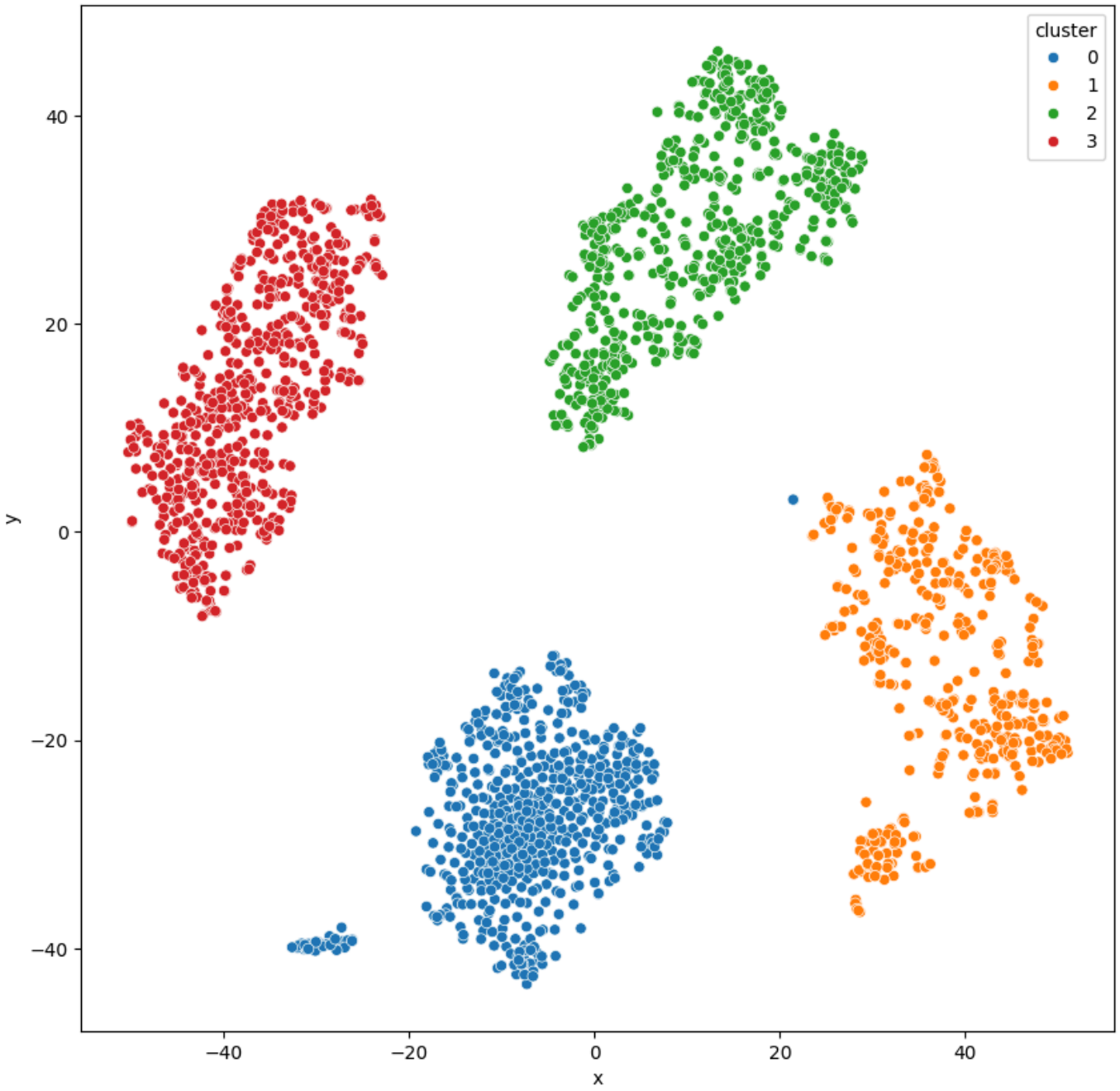
| | std_vector |
|---|---|
| **0** | [-0.9614479972007641, 0.6957987565937652, 0.59... |
| **1** | [-0.9792962849264775, 0.7375863705340073, 0.57... |
| **2** | [-0.9016816118502328, 0.7442563640622617, 0.59... |
| **3** | [-1.0004149361896462, 0.7354988385819545, 0.59... |
| **4** | [-0.8834392389517144, 0.8111157942320139, 0.57... |

**dtype:** object

```
# visualize the real cluster using t-SNE
```

```
from sklearn.manifold import TSNE

tsne = TSNE(n_components=2, random_state=0)

# transofrm the df['vector'] to dataframe with freatuer 0 - 99 for
df_copy = df['std_vector'].apply(pd.Series)

df_tsne = tsne.fit_transform(df_copy)

df_tsne = pd.DataFrame(df_tsne, columns=['x', 'y'])

df_tsne['cluster'] = df['cluster']
```

```
# plot the data
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 10))

sns.scatterplot(data=df_tsne, x="x", y="y", hue="cluster")

plt.show()

# save the data
df.to_csv("doc2vec_tsne.csv", index=False)
```




```
# import all the the needed libraries NaiveBayes, SVM, LoR, RF
data = df['std_vector'].tolist()
```

```
data = np.array(data)
```

```
type(data)
```

```
numpy.ndarray
```

```
data.shape
```

```
(2346, 100)
```

## Naive Bayes Classifier

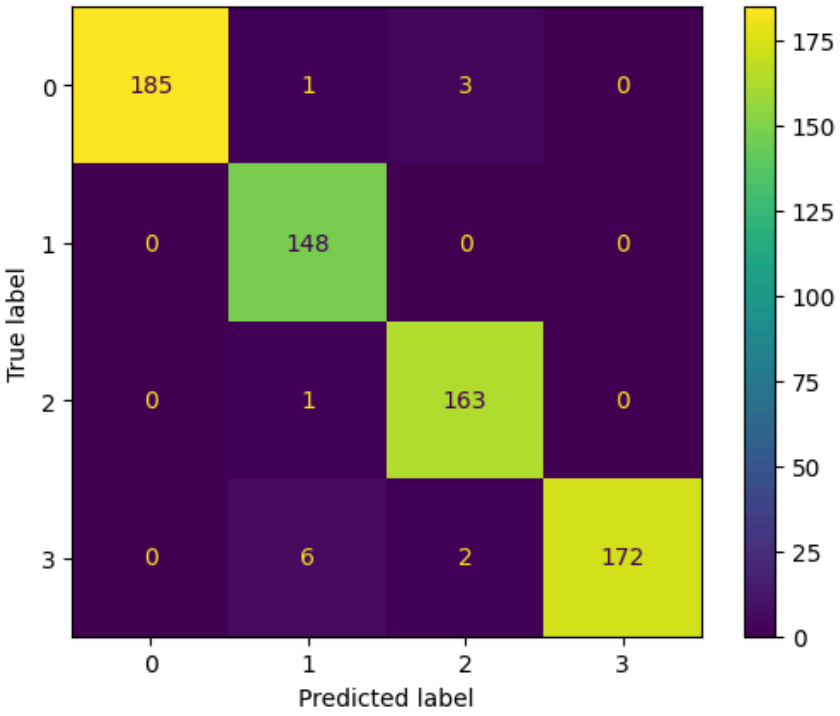
```
# naive bayes
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB

X = data
y = df['cluster'].to_numpy()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.29, random_state=42)

print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_train type:", type(X_train))
print("y_train type:", type(y_train))
```

```
X_train shape: (1665, 100)
y_train shape: (1665,)
X_train type: <class 'numpy.ndarray'>
y_train type: <class 'numpy.ndarray'>
```

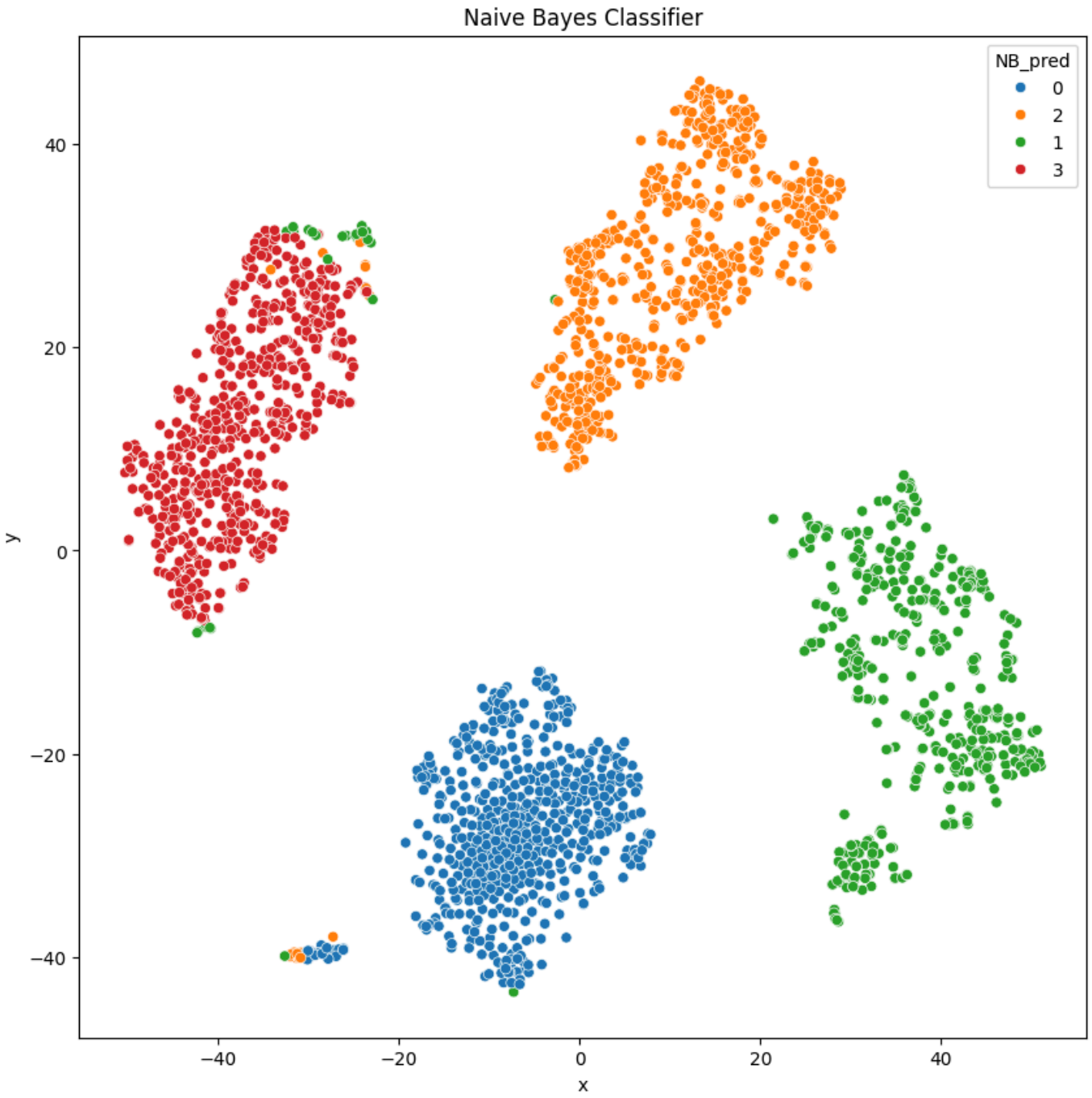
```
# use Naive Bayes with 10-fold cross validation
from sklearn.model_selection import cross_val_score

gnb = GaussianNB()

scores = cross_val_score(gnb, X_train, y_train, cv=10)

print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Accuracy: 0.97 (+/- 0.02)
```

```
from sklearn.metrics import ConfusionMatrixDisplay

gnb.fit(X_train, y_train)

disp = ConfusionMatrixDisplay.from_estimator(gnb, X_test, y_test)

disp.plot()

plt.show()
```




```
# get the calssification report for all X from the model and color the results using the tsn
df_tsne['NB_pred'] = gnb.predict(X)


plt.figure(figsize=(10, 10))

#  add title
plt.title("Naive Bayes Classifier")

sns.scatterplot(data=df_tsne, x="x", y="y", hue="NB_pred")

plt.show()
```

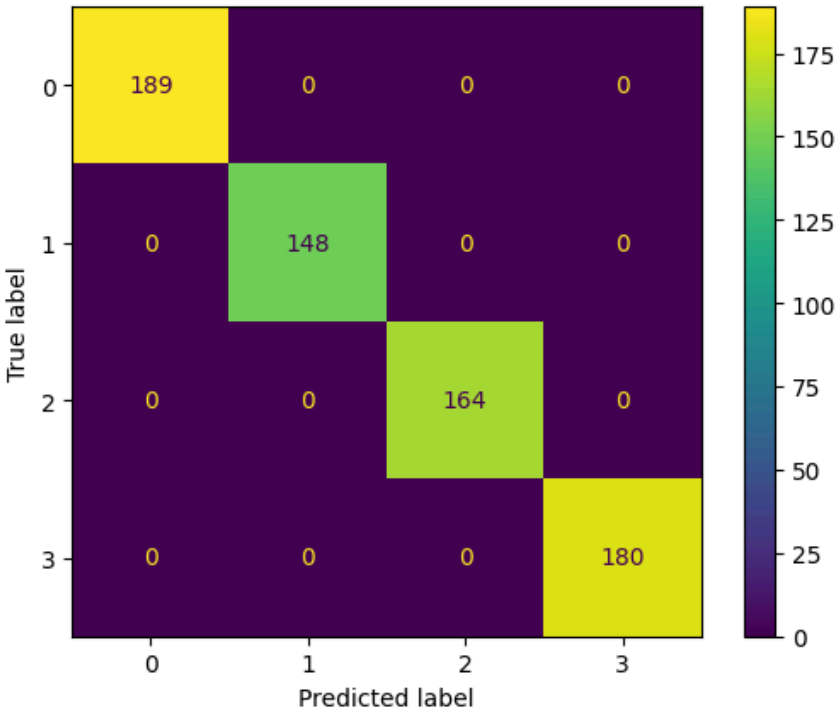



## SVM - Support Vector Machine

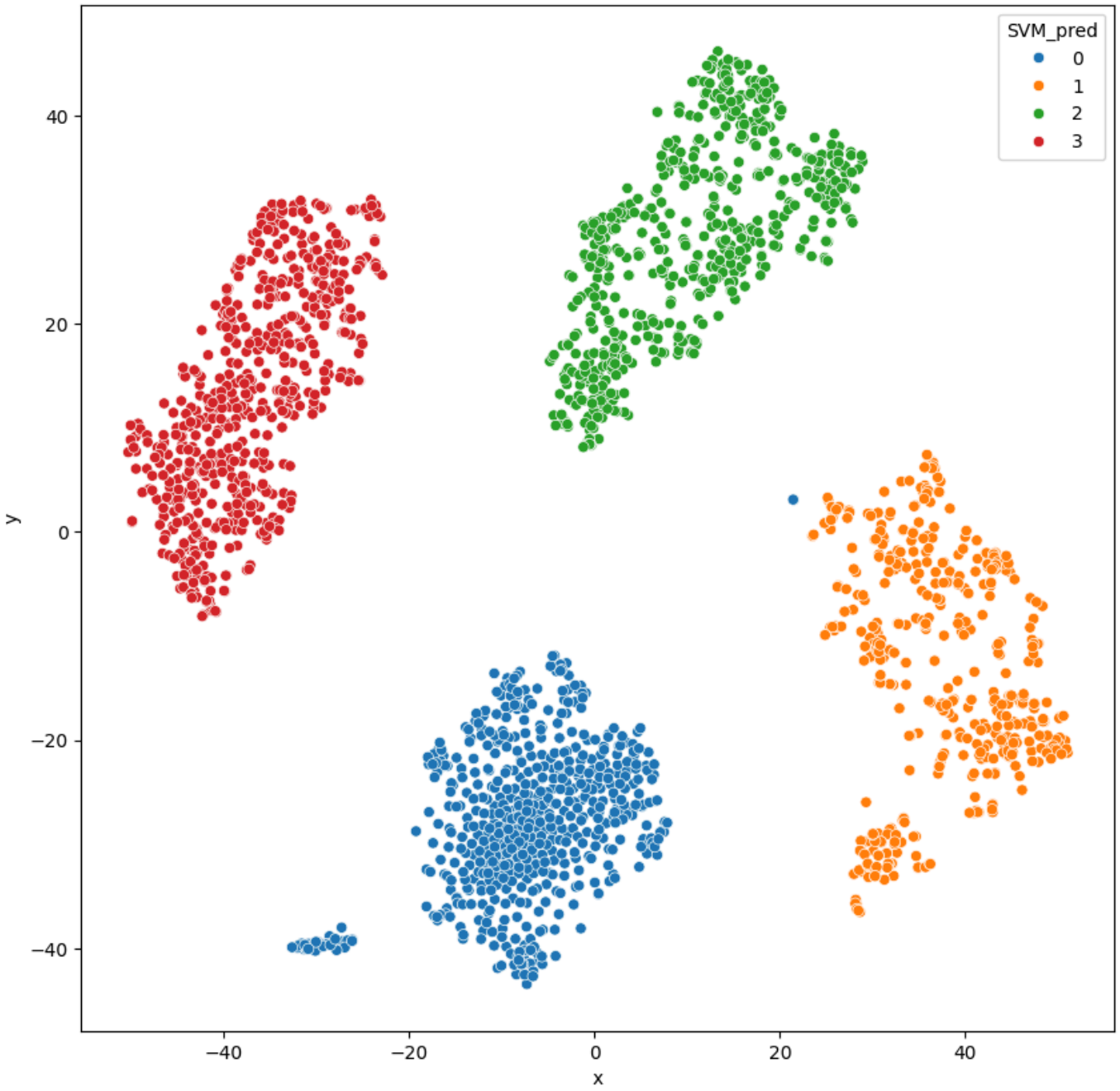
```
# use SVM with 10-fold cross validation
from sklearn.svm import SVC

svc = SVC()

scores = cross_val_score(svc, X_train, y_train, cv=10)

print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Accuracy: 1.00 (+/- 0.00)
```

```
# same as NB

svc.fit(X_train, y_train)

disp = ConfusionMatrixDisplay.from_estimator(svc, X_test, y_test)

disp.plot()

plt.show()
```




```
# get the calssification report for all X from the model and color the results using the tsn
df_tsne["SVM_pred"] = svc.predict(X)


plt.figure(figsize=(10, 10))

sns.scatterplot(data=df_tsne, x="x", y="y", hue="SVM_pred")

plt.show()
```

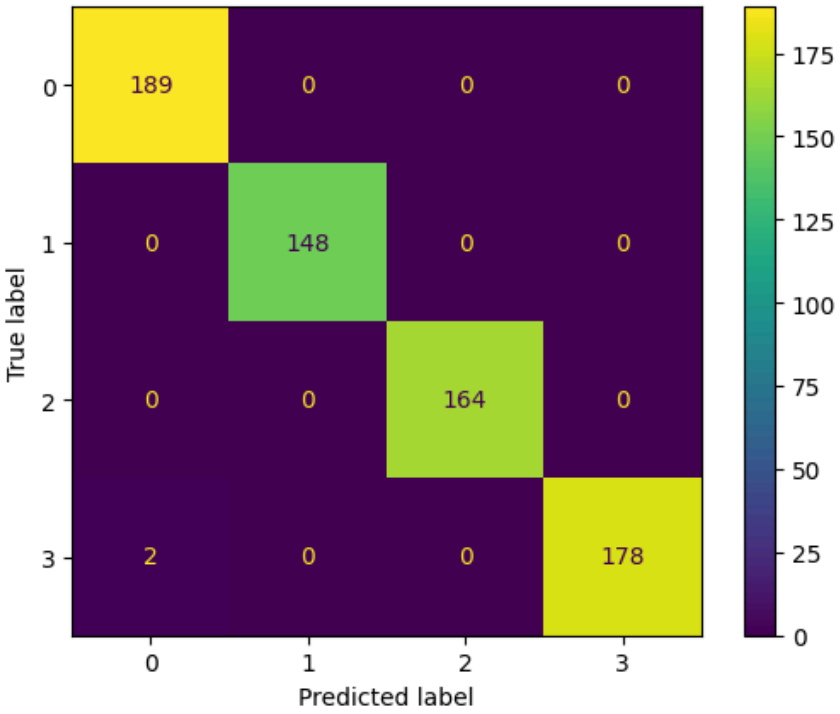



## Logistic Regression

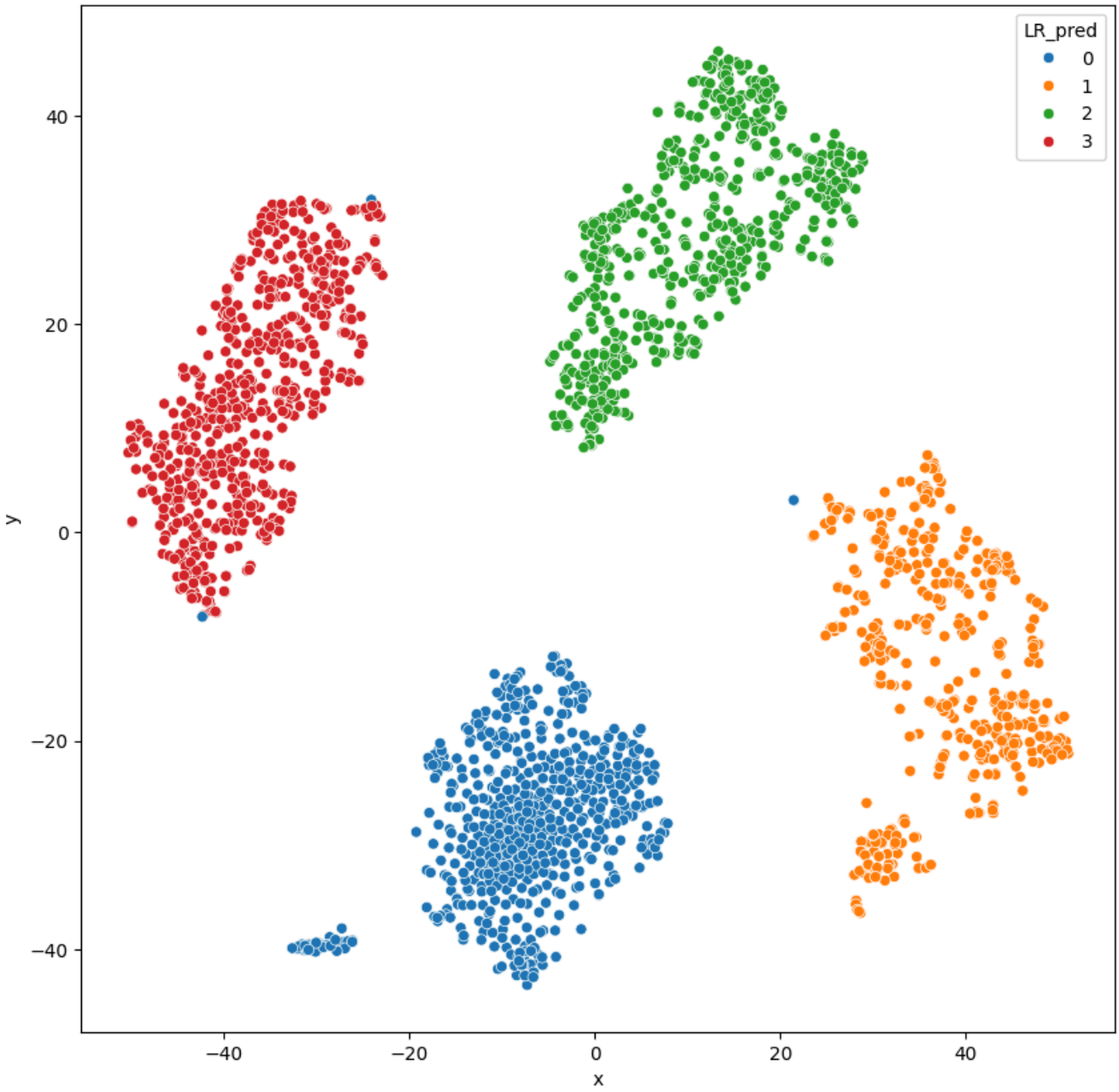
```
# use Logistic Regression with 10-fold cross validation

from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()

scores = cross_val_score(lr, X_train, y_train, cv=10)

print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Accuracy: 1.00 (+/- 0.01)
```

```
# same

# visualize the results of the classification for all the X

lr.fit(X_train, y_train)

disp = ConfusionMatrixDisplay.from_estimator(lr, X_test, y_test)

disp.plot()

plt.show()
```




```
# plot the results using tsne

df_tsne["LR_pred"] = lr.predict(X)

plt.figure(figsize=(10, 10))

sns.scatterplot(data=df_tsne, x="x", y="y", hue="LR_pred")

plt.show()
```

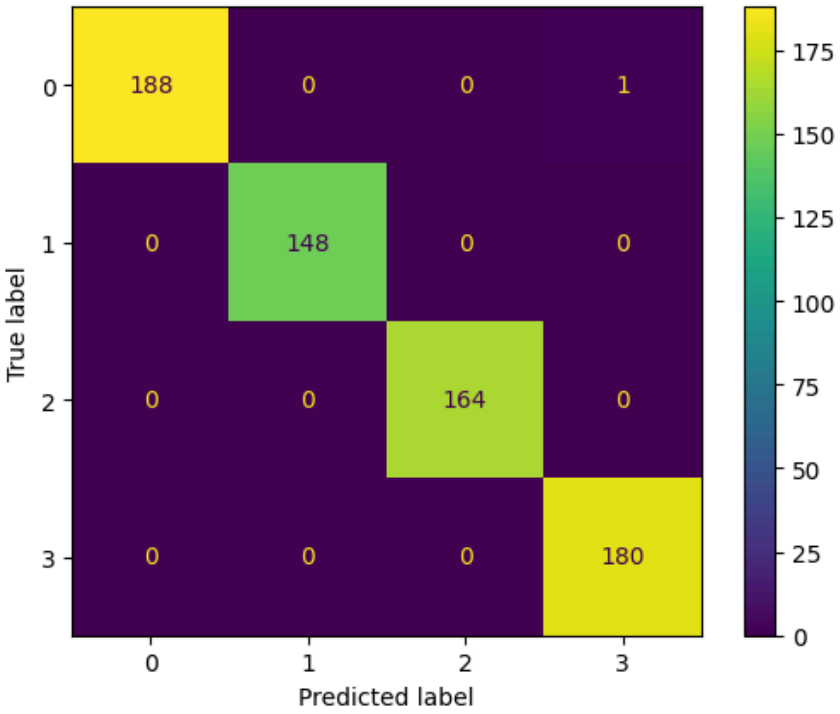



# RF - Random Forest Classifier

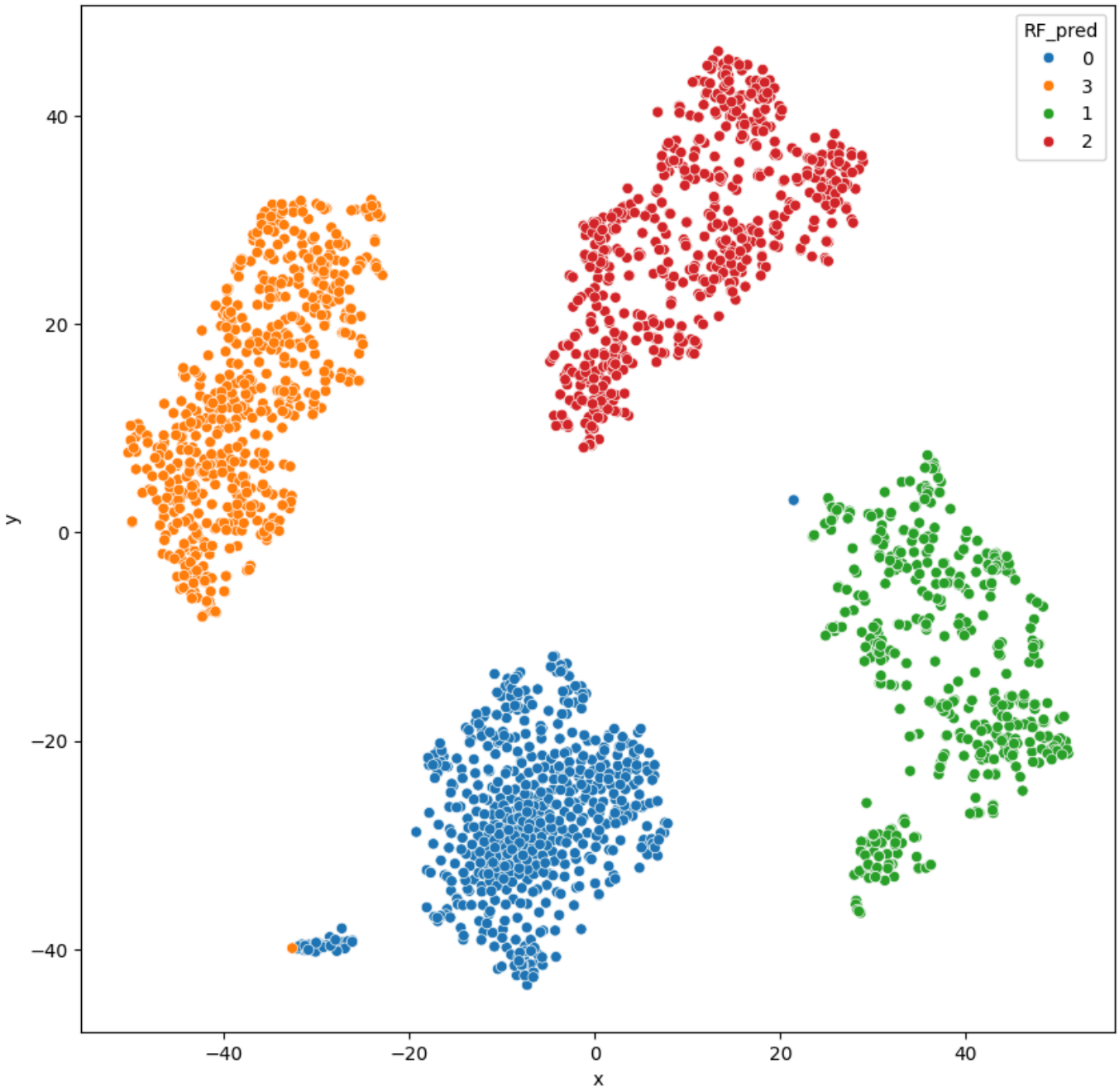
```
# use Random Forest with 10-fold cross validation

from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()

scores = cross_val_score(rf, X_train, y_train, cv=10)

print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Accuracy: 1.00 (+/- 0.01)
```

```
# same

# visualize the results of the classification for all the X

rf.fit(X_train, y_train)

disp = ConfusionMatrixDisplay.from_estimator(rf, X_test, y_test)

disp.plot()

plt.show()
```




```
# plot the results using tsne

df_tsne["RF_pred"] = rf.predict(X)

plt.figure(figsize=(10, 10))

sns.scatterplot(data=df_tsne, x="x", y="y", hue="RF_pred")

plt.show()
```




# ANN - Artificial Neural Network Classifier

We will build a NN using `tensorflow` and `keras` to classify the journal group.

The architecture of the NN is as follows:

- Embedding layer with 100 input dimensions.
- Hidden layer with 10 node and `relu` activation function.
- Hidden layer with 10 node and `relu` activation function.
- Hidden layer with 7 node and `relu` activation function.
- Output layer with 4 nodes and `softmax` activation function. (4 classes)

Seconde architecture:

- Embedding layer with 100 input dimensions.
- Hidden layer with 10 node and `gelu` activation function.
- Hidden layer with 10 node and `gelu` activation function.
- Hidden layer with 7 node and `gelu` activation function.
- Output layer with 4 nodes and `softmax` activation function. (4 classes)

```
X = X.astype(np.float32)
y = y.astype(int)
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

model_1 = Sequential([
Dense(100, activation='relu', input_shape=(100,)),
Dense(10, activation='relu'),
Dense(10, activation='relu'),
Dense(7, activation='relu'),
Dense(4, activation='softmax')
])

# compile the model
model_1.compile(optimizer='adam',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])

# fit the model
history = model_1.fit(X_train, y_train, epochs=15, batch_size=32, validation_split=0.2)
```

```
Epoch 1/15
 [1m47/47 [0m  [32m━━━━━━━━━━━━━━━━━━━━[0m [37m [0m  [1m8s [0m 89ms/step - accuracy: 0.5008 -
loss: 1.1708 - val_accuracy: 0.7261 - val_loss: 0.4780
Epoch 2/15
 [1m47/47 [0m  [32m━━━━━━━━━━━━━━━━━━━━[0m [37m [0m  [1m4s [0m 4ms/step - accuracy: 0.8816 -
loss: 0.3191 - val_accuracy: 1.0000 - val_loss: 0.0458
Epoch 3/15
 [1m47/47 [0m  [32m━━━━━━━━━━━━━━━━━━━━[0m [37m [0m  [1m0s [0m 3ms/step - accuracy: 0.9998 -
loss: 0.0277 - val_accuracy: 1.0000 - val_loss: 0.0115
Epoch 4/15
 [1m47/47 [0m  [32m━━━━━━━━━━━━━━━━━━━━[0m [37m [0m  [1m0s [0m 3ms/step - accuracy: 0.9993 -
loss: 0.0073 - val_accuracy: 1.0000 - val_loss: 0.0070
Epoch 5/15
 [1m47/47 [0m  [32m━━━━━━━━━━━━━━━━━━━━[0m [37m [0m  [1m0s [0m 3ms/step - accuracy: 1.0000 -
```

```
loss: 0.0031 - val_accuracy: 1.0000 - val_loss: 0.0052
Epoch 6/15
 [1m47/47 [0m [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m [1m0s [0m 4ms/step - accuracy: 1.0000 -
loss: 0.0027 - val_accuracy: 1.0000 - val_loss: 0.0042
Epoch 7/15
 [1m47/47 [0m [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m [1m0s [0m 3ms/step - accuracy: 1.0000 -
loss: 0.0019 - val_accuracy: 1.0000 - val_loss: 0.0035
Epoch 8/15
 [1m47/47 [0m [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m [1m0s [0m 3ms/step - accuracy: 1.0000 -
loss: 0.0015 - val_accuracy: 1.0000 - val_loss: 0.0032
Epoch 9/15
 [1m47/47 [0m [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m [1m0s [0m 3ms/step - accuracy: 1.0000 -
loss: 0.0013 - val_accuracy: 1.0000 - val_loss: 0.0029
Epoch 10/15
 [1m47/47 [0m [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m [1m0s [0m 4ms/step - accuracy: 1.0000 -
loss: 0.0013 - val_accuracy: 1.0000 - val_loss: 0.0025
Epoch 11/15
 [1m47/47 [0m [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m [1m0s [0m 3ms/step - accuracy: 1.0000 -
loss: 5.3030e-04 - val_accuracy: 1.0000 - val_loss: 0.0023
Epoch 12/15
 [1m47/47 [0m [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m [1m0s [0m 3ms/step - accuracy: 1.0000 -
loss: 4.8079e-04 - val_accuracy: 1.0000 - val_loss: 0.0022
Epoch 13/15
 [1m47/47 [0m [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m [1m0s [0m 3ms/step - accuracy: 1.0000 -
loss: 4.3031e-04 - val_accuracy: 1.0000 - val_loss: 0.0020
Epoch 14/15
 [1m47/47 [0m [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m [1m0s [0m 3ms/step - accuracy: 1.0000 -
loss: 3.6769e-04 - val_accuracy: 1.0000 - val_loss: 0.0019
Epoch 15/15
 [1m47/47 [0m [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m [1m0s [0m 4ms/step - accuracy: 1.0000 -
loss: 3.2712e-04 - val_accuracy: 1.0000 - val_loss: 0.0018
```

```python
# model 2

model_2 = Sequential(
[
    Dense(100, activation="gelu", input_shape=(100,)),
    Dense(10, activation="gelu"),
    Dense(10, activation="gelu"),
    Dense(7, activation="gelu"),
    Dense(4, activation="softmax"),
]
)

# compile the model
model_2.compile(optimizer='adam',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])
```

```
# fit the model
history = model_2.fit(X_train, y_train, epochs=15, batch_size=32, validation_split=0.2)
```

```
Epoch 1/15
 [1m47/47 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m7s [0m 78ms/step - accuracy: 0.4899 -
loss: 1.1962 - val_accuracy: 0.9096 - val_loss: 0.5046
Epoch 2/15
 [1m47/47 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m0s [0m 3ms/step - accuracy: 0.9698 -
loss: 0.4108 - val_accuracy: 0.9947 - val_loss: 0.3087
Epoch 3/15
 [1m47/47 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m0s [0m 3ms/step - accuracy: 1.0000 -
loss: 0.2932 - val_accuracy: 0.9947 - val_loss: 0.2548
Epoch 4/15
 [1m47/47 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m0s [0m 3ms/step - accuracy: 1.0000 -
loss: 0.2515 - val_accuracy: 1.0000 - val_loss: 0.1801
Epoch 5/15
 [1m47/47 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m0s [0m 3ms/step - accuracy: 1.0000 -
loss: 0.1421 - val_accuracy: 0.9973 - val_loss: 0.0240
Epoch 6/15
 [1m47/47 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m0s [0m 4ms/step - accuracy: 1.0000 -
loss: 0.0121 - val_accuracy: 0.9973 - val_loss: 0.0079
Epoch 7/15
 [1m47/47 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m0s [0m 3ms/step - accuracy: 1.0000 -
loss: 0.0028 - val_accuracy: 0.9973 - val_loss: 0.0060
Epoch 8/15
 [1m47/47 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m0s [0m 6ms/step - accuracy: 1.0000 -
loss: 0.0017 - val_accuracy: 0.9973 - val_loss: 0.0052
Epoch 9/15
 [1m47/47 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m0s [0m 5ms/step - accuracy: 1.0000 -
loss: 0.0010 - val_accuracy: 0.9973 - val_loss: 0.0047
Epoch 10/15
 [1m47/47 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m0s [0m 5ms/step - accuracy: 1.0000 -
loss: 7.6045e-04 - val_accuracy: 0.9973 - val_loss: 0.0044
Epoch 11/15
 [1m47/47 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m0s [0m 5ms/step - accuracy: 1.0000 -
loss: 6.8052e-04 - val_accuracy: 0.9973 - val_loss: 0.0042
Epoch 12/15
 [1m47/47 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m0s [0m 6ms/step - accuracy: 1.0000 -
loss: 3.9661e-04 - val_accuracy: 0.9973 - val_loss: 0.0040
Epoch 13/15
 [1m47/47 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m0s [0m 5ms/step - accuracy: 1.0000 -
loss: 4.0260e-04 - val_accuracy: 0.9973 - val_loss: 0.0038
Epoch 14/15
 [1m47/47 [0m  [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m  [1m0s [0m 5ms/step - accuracy: 1.0000 -
loss: 3.1882e-04 - val_accuracy: 0.9973 - val_loss: 0.0037
Epoch 15/15
```

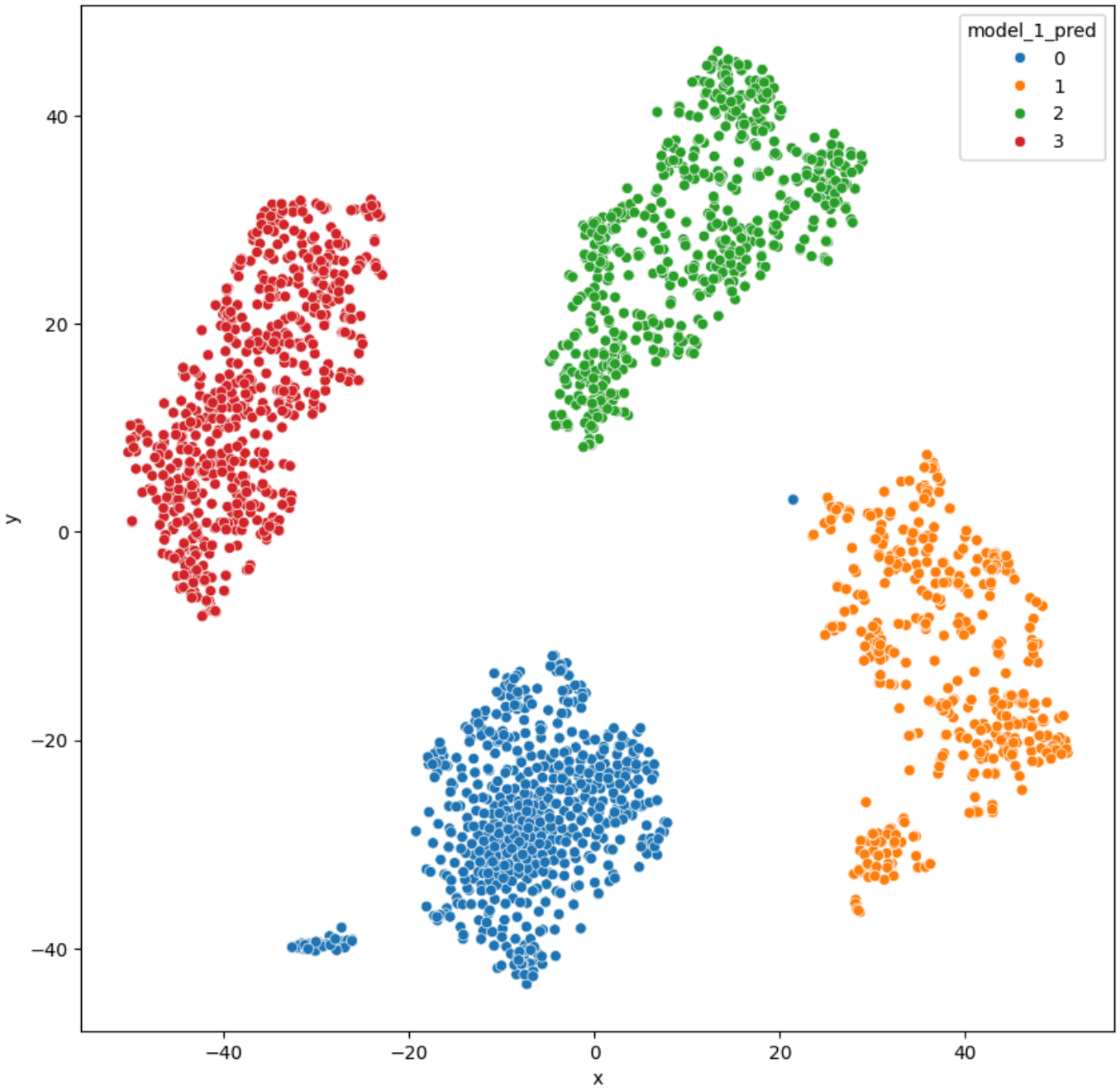
```
 [1m47/47 [0m  [32m━━━━━━━━━━━━━━━━━━━━[0m [37m [0m  [1m0s [0m 5ms/step - accuracy: 1.0000 - loss: 2.9605e-04 - val_accuracy: 0.9973 - val_loss: 0.0036
```

```
# check the accuracy of the models

# model 1
loss, accuracy = model_1.evaluate(X_test, y_test)

print("Model 1 Accuracy: ", accuracy)

# model 2

loss, accuracy = model_2.evaluate(X_test, y_test)

print("Model 2 Accuracy: ", accuracy)
```

```
 [1m15/15 [0m  [32m━━━━━━━━━━━━━━━━━━━━[0m [37m [0m  [1m1s [0m 65ms/step - accuracy: 1.0000 - loss: 0.0032
Model 1 Accuracy:  1.0
 [1m15/15 [0m  [32m━━━━━━━━━━━━━━━━━━━━[0m [37m [0m  [1m1s [0m 51ms/step - accuracy: 1.0000 - loss: 0.0027
Model 2 Accuracy:  1.0
```

```
# plot the prediction for model 1

predictions = model_1.predict(X)

# Convert probabilities to class labels
predicted_classes = np.argmax(predictions, axis=1)


df_tsne["model_1_pred"] = [str(cls) for cls in predicted_classes]

plt.figure(figsize=(10, 10))

sns.scatterplot(data=df_tsne, x="x", y="y", hue="model_1_pred")

plt.show()
```

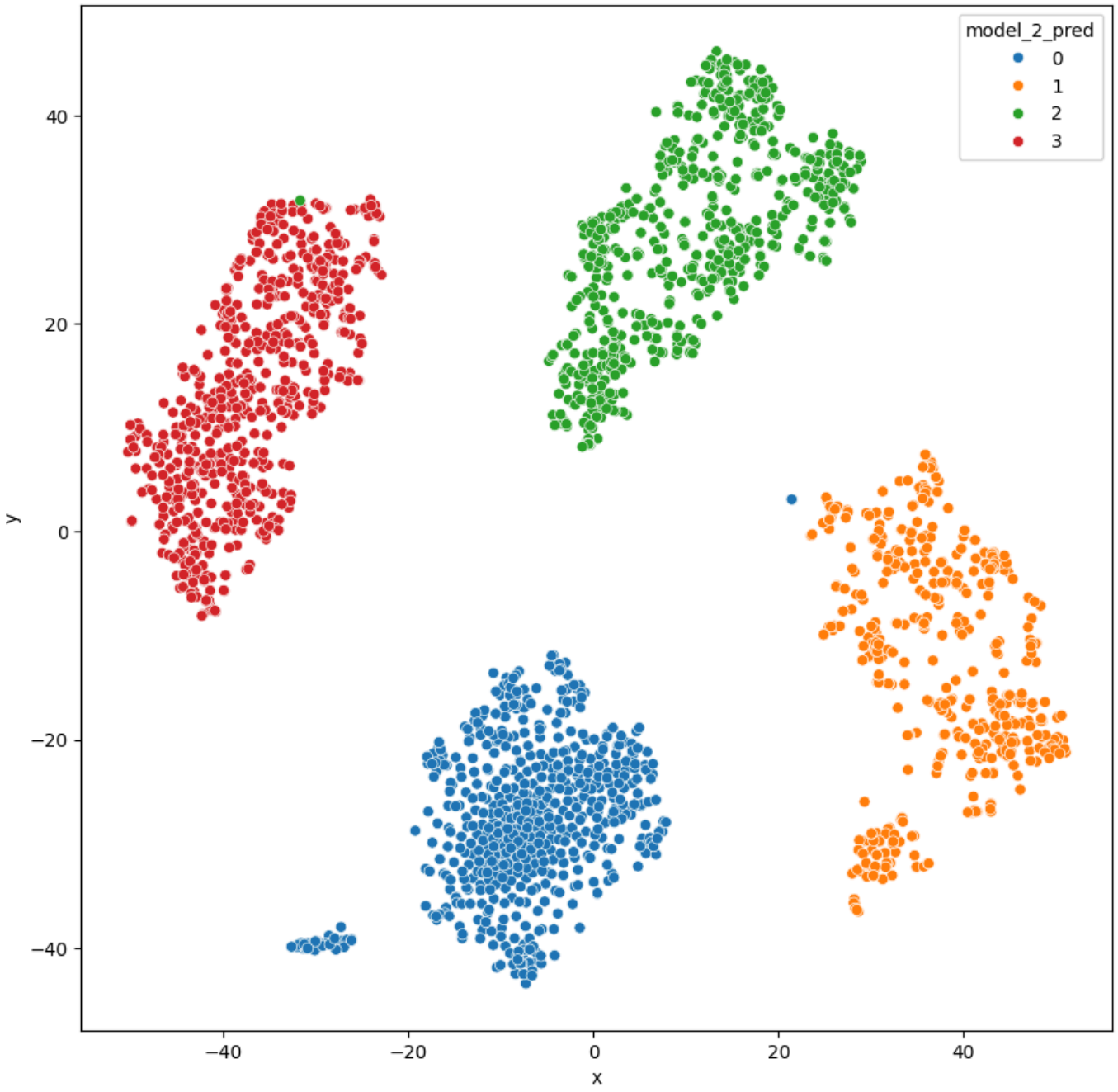
```
 [1m74/74 [0m  [32m━━━━━━━━━━━━━━━━━━━━[0m [37m [0m  [1m0s [0m 1ms/step
```




```
# plot the prediction for model 1

predictions = model_2.predict(X)

# Convert probabilities to class labels
predicted_classes = np.argmax(predictions, axis=1)


df_tsne["model_2_pred"] = [str(cls) for cls in predicted_classes]

plt.figure(figsize=(10, 10))

sns.scatterplot(data=df_tsne, x="x", y="y", hue="model_2_pred")
```

```
plt.show()
```

```
[1m74/74 [0m [32m━━━━━━━━━━━━━━━━━━━━ [0m [37m [0m [1m1s [0m 5ms/step
```




```
# save the model into a file

model_1.save("model_1.h5")

model_2.save("model_2.h5")
```

# Top 20 Feature

We took the top 20 feature for the TF-IDF lemma for example

## Naive Bayes

```
Feature 62: city
Feature 112: protest
Feature 1: say
Feature 158: student
Feature 89: hospital
Feature 154: arrest
Feature 22: day
Feature 23: force
Feature 116: university
Feature 45: rafah
Feature 17: us
Feature 21: palestinians
Feature 39: al
Feature 58: un
Feature 74: continue
Feature 6: palestinian
Feature 8: attack
Feature 12: kill
Feature 0: israel
Feature 2: gaza
```

## SVM

```
Feature 1800: occupied
Feature 102: ceasefire
Feature 436: joe
Feature 2310: kingdom
Feature 59: world
Feature 250: chief
Feature 243: uk
Feature 357: states
Feature 304: palestine
Feature 1841: guide
Feature 34: report
Feature 48: country
Feature 487: anti
Feature 50: international
Feature 32: state
Feature 958: icj
Feature 17: us
Feature 58: un
Feature 2: gaza
```

As we can see the for the are the most important are the one we would expect like *UN* , *Gaza*, *Palestine*, ... this mean that the clustering did work as intended.