In [1]:
```
'''import torch

if torch.cuda.is_available():
    device = torch.device("cuda")
    print(f'There are {torch.cuda.device_count()} GPU(s) available.')
    print('Device name:', torch.cuda.get_device_name(0))
else:
    print('No GPU available, using the CPU instead.')
    device = torch.device("cpu")'''
```

Out[1]:
```
'import torch\n\nif torch.cuda.is_available():       \n    device = torch.dev
ice("cuda")\n    print(f\'There are {torch.cuda.device_count()} GPU(s) availa
ble.\')\n    print(\'Device name:\', torch.cuda.get_device_name(0))\nelse:\n
print(\'No GPU available, using the CPU instead.\')\n    device = torch.devic
e("cpu")'
```

In [2]:
```
import pandas as pd
import os
from sklearn.model_selection import train_test_split
import time

# Load the data
documents_path = os.path.expanduser('~\\Documents')
os.chdir(documents_path)
df = pd.read_excel('final_data.xlsx')

# Define complaints and categories
complaints = df["Complaint"]
categories = df["Category Level 1"]

# Reduce data
subset_ratio = 1
subset_size = int(len(complaints) * subset_ratio)
complaints = complaints[:subset_size]
categories = categories[:subset_size]

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(complaints, categories, te
```

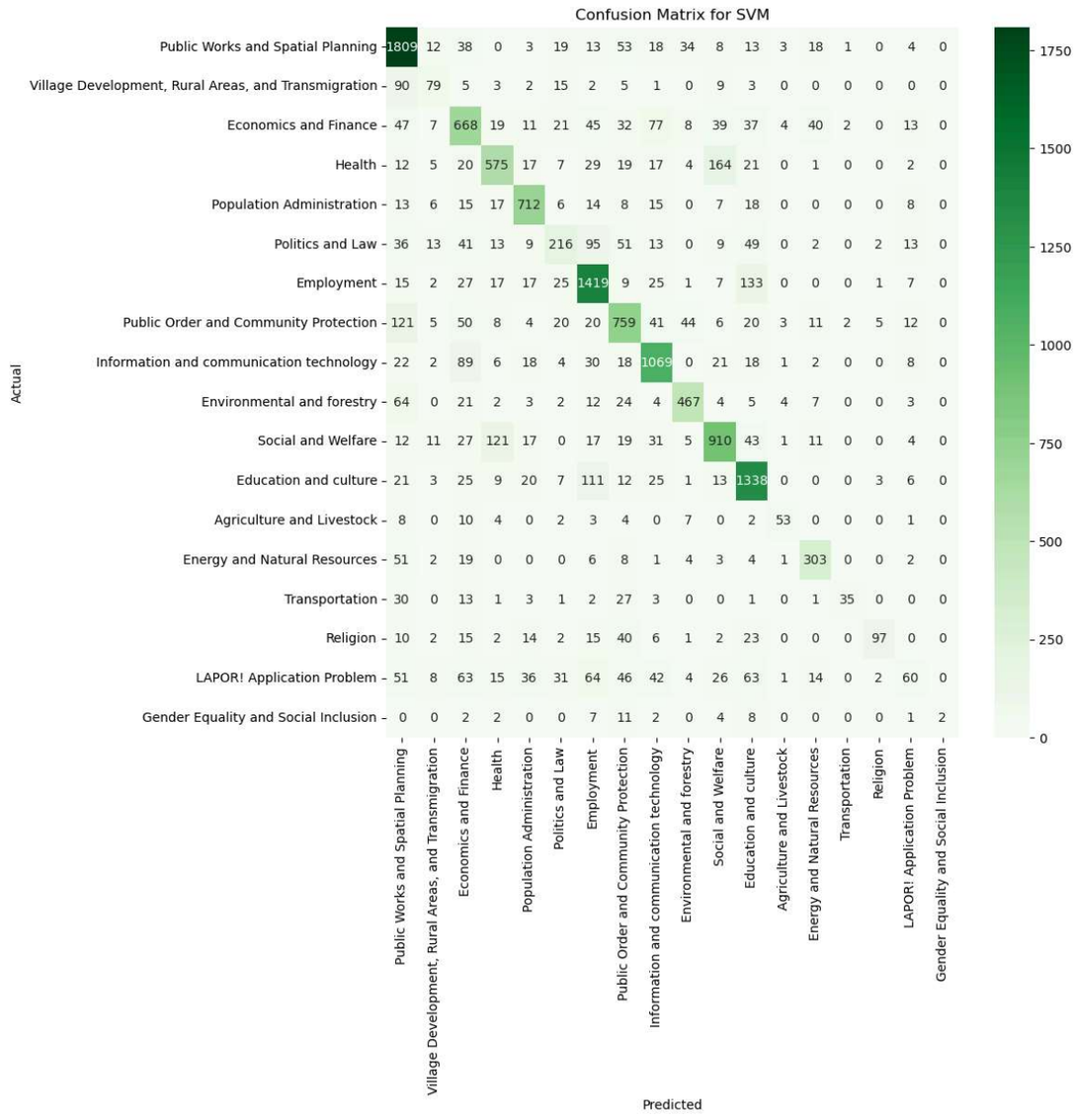
In [3]:
```
import nltk
import re
from nltk.corpus import stopwords
from Sastrawi.Stemmer.StemmerFactory import StemmerFactory

# Start time
start_time = time.time()

# Download necessary NLTK data
nltk.download("punkt")
nltk.download("averaged_perceptron_tagger")
nltk.download("tagsets")
nltk.download("stopwords")

# Define stopwords and stemmer
stops = stopwords.words("indonesian")
factory = StemmerFactory()
stemmer = factory.create_stemmer()

# Define preprocessing functions
def extract_words(text):
    words = [word.lower() for word in re.findall("[^\d\W]+", text)]
    return words

def filter_words(words, stops, stemmer):
    filtered_words = [word for word in words if word not in stops]
    stemmed_words = [stemmer.stem(word) for word in filtered_words]
    return stemmed_words

def tokenise(text, stops, stemmer):
    return filter_words(extract_words(text), stops, stemmer)

# Prepare tokeniser for TF-IDF Vectorizer
from functools import partial
from sklearn.feature_extraction.text import TfidfVectorizer

tokeniser_partial = partial(tokenise, stemmer=stemmer, stops=stops)
tfidf_vectorizer = TfidfVectorizer(analyzer=tokeniser_partial)

end_time = time.time()
runtime = end_time - start_time
print(f"runtime: {runtime:.2f} seconds")
```

```
runtime: 0.17 seconds

[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\asus\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\asus\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package tagsets to
[nltk_data]     C:\Users\asus\AppData\Roaming\nltk_data...
[nltk_data]   Package tagsets is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\asus\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [4]:

```
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, confusion_matrix, classification_r

# Define SVM kernels to test
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
best_kernel = None
best_score = 0
results = {}

# Start time
start_time = time.time()

# Perform cross-validation for each kernel
for kernel in kernels:
    print(f"Evaluating kernel: {kernel}")
    pipeline_svm = Pipeline([
        ('vectorizer', tfidf_vectorizer),
        ('classifier', SVC(kernel=kernel))
    ])

    # Cross-validation
    skf = StratifiedKFold(n_splits=3, shuffle=True, random_state=123)
    cv_scores = cross_val_score(pipeline_svm, X_train, y_train, cv=skf, n_jobs

    mean_cv_score = cv_scores.mean()
    results[kernel] = mean_cv_score

    print(f"Mean CV Accuracy for kernel '{kernel}': {mean_cv_score:.4f}")

    # Select the best kernel
    if mean_cv_score > best_score:
        best_score = mean_cv_score
        best_kernel = kernel

print(f"Best kernel: {best_kernel} with CV accuracy: {best_score:.4f}")

# Train the final model on the entire training set using the best kernel
pipeline_svm = Pipeline([
    ('vectorizer', tfidf_vectorizer),
    ('classifier', SVC(kernel=best_kernel))
])
pipeline_svm.fit(X_train, y_train)

# Predict on the test set
y_pred_svm = pipeline_svm.predict(X_test)

# Get classification report for detailed evaluation
report = classification_report(y_test, y_pred_svm, labels=categories.unique())
print(report)

# Evaluate the model
accuracy_svm = accuracy_score(y_test, y_pred_svm)
print(f"SVM Accuracy on Test Set: {accuracy_svm }")

# Confusion matrix
conf_matrix_svm = confusion_matrix(y_test, y_pred_svm, labels=categories.uniqu
conf_matrix_df_svm = pd.DataFrame(conf_matrix_svm, index=categories.unique(),

plt.figure(figsize=(10, 10))
sns.heatmap(conf_matrix_df_svm, annot=True, fmt='d', cmap='Greens')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for SVM')
plt.show()

# End time
end_time = time.time()
runtime = end_time - start_time
print(f"runtime: {runtime:.2f} seconds")
```

```
Evaluating kernel: linear
Mean CV Accuracy for kernel 'linear': 0.7209
Evaluating kernel: poly
Mean CV Accuracy for kernel 'poly': 0.6091
Evaluating kernel: rbf
Mean CV Accuracy for kernel 'rbf': 0.7226
Evaluating kernel: sigmoid
Mean CV Accuracy for kernel 'sigmoid': 0.7176
Best kernel: rbf with CV accuracy: 0.7226
                                                    precision    recall  f1
-score   support

                    Public Works and Spatial Planning       0.75      0.88
0.81      2046
Village Development, Rural Areas, and Transmigration       0.50      0.37
0.43       214
                               Economics and Finance       0.58      0.62
0.60      1070
                                              Health       0.71      0.64
0.67       893
                          Population Administration       0.80      0.85
0.83       839
                                   Politics and Law       0.57      0.38
0.46       562
                                         Employment       0.75      0.83
0.79      1705
               Public Order and Community Protection       0.66      0.67
0.67      1131
            Information and communication technology       0.77      0.82
0.79      1308
                          Environmental and forestry       0.81      0.75
0.78       622
                                  Social and Welfare       0.74      0.74
0.74      1229
                               Education and culture       0.74      0.84
0.79      1594
                           Agriculture and Livestock       0.75      0.56
0.64        94
                       Energy and Natural Resources       0.74      0.75
0.74       404
                                     Transportation       0.88      0.30
0.45       117
                                           Religion       0.88      0.42
0.57       229
                         LAPOR! Application Problem       0.42      0.11
0.18       526
                Gender Equality and Social Inclusion       1.00      0.05
0.10        39

                                           accuracy
0.72     14622
                                          macro avg       0.72      0.59
0.61     14622
                                       weighted avg       0.71      0.72
0.71     14622

SVM Accuracy on Test Set: 0.7229517165914375
```



runtime: 7083.05 seconds

In [5]:
```
'''from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import time

# Define the pipeline with NMF
pipeline_nb = Pipeline([
    ('vectorizer', tfidf_vectorizer),
    ('classifier', MultinomialNB())
])
# Start time
start_time = time.time()

# Train the pipeline
pipeline_nb.fit(X_train, y_train)

# Predict on the test set
y_pred_nb = pipeline_nb.predict(X_test)

# Evaluate accuracy
accuracy_nb = accuracy_score(y_test, y_pred_nb)
print(f"Naive Bayes Accuracy: {accuracy_nb * 100:.2f}%")

# Confusion matrix
conf_matrix_nb = confusion_matrix(y_test, y_pred_nb, labels=categories.unique(
conf_matrix_df_nb = pd.DataFrame(conf_matrix_nb, index=categories.unique(), co

plt.figure(figsize=(10, 10))
sns.heatmap(conf_matrix_df_nb, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for Naive Bayes')
plt.show()

# End time
end_time = time.time()
runtime = end_time - start_time
print(f"Runtime: {runtime:.2f} seconds")'''
```

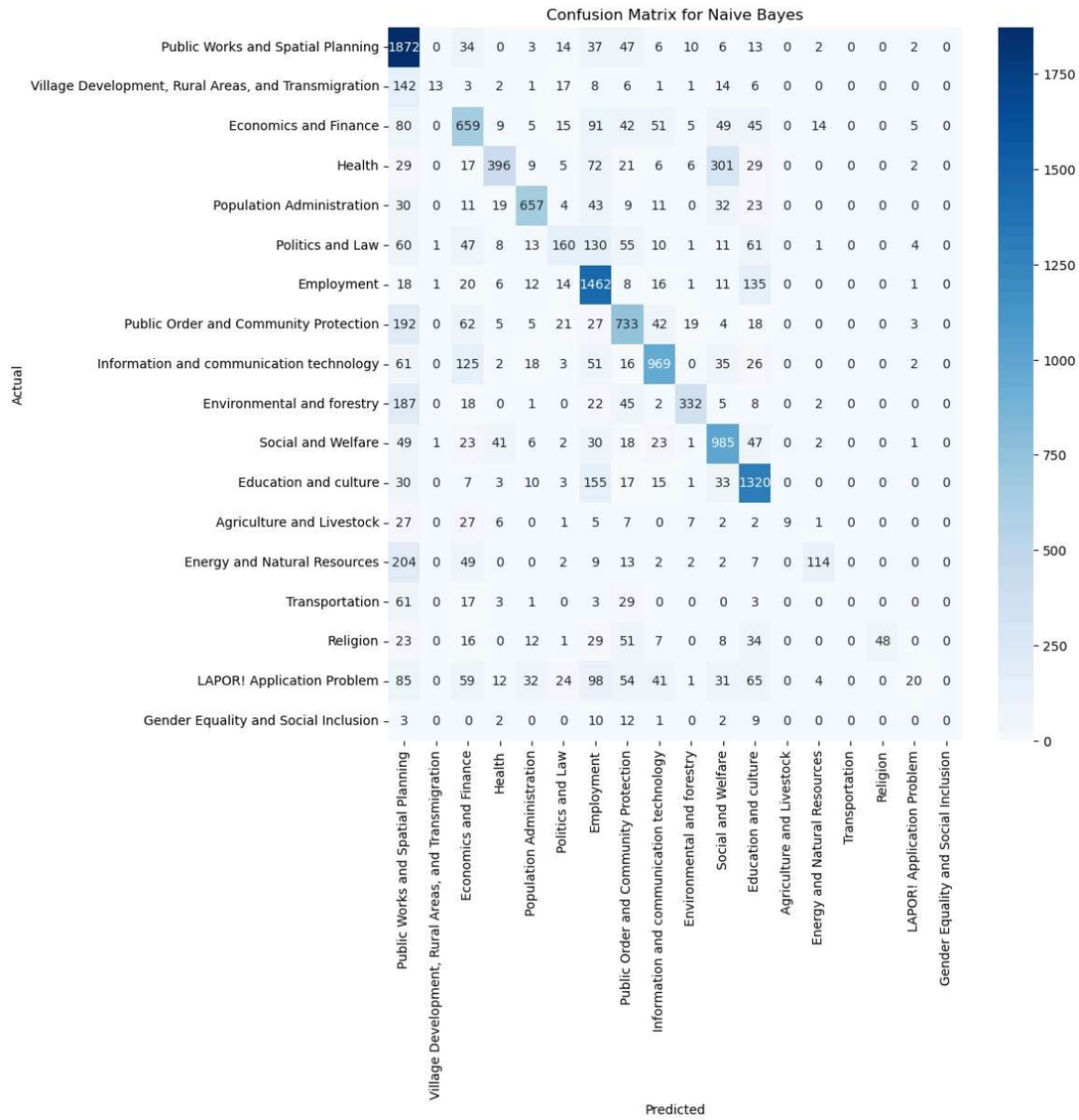
Out[5]:
```
'from sklearn.naive_bayes import MultinomialNB\nfrom sklearn.pipeline import
Pipeline\nfrom sklearn.metrics import accuracy_score, confusion_matrix\nimpor
t seaborn as sns\nimport matplotlib.pyplot as plt\nimport time\n\n# Define th
e pipeline with NMF\npipeline_nb = Pipeline([\n    (\'vectorizer\', tfidf_vec
torizer),\n    (\'classifier\', MultinomialNB())\n])\n# Start time\nstart_tim
e = time.time()\n\n# Train the pipeline\npipeline_nb.fit(X_train, y_train)\n
\n# Predict on the test set\ny_pred_nb = pipeline_nb.predict(X_test)\n\n# Eva
luate accuracy\naccuracy_nb = accuracy_score(y_test, y_pred_nb)\nprint(f"Naiv
e Bayes Accuracy: {accuracy_nb * 100:.2f}%")\n\n# Confusion matrix\nconf_matr
ix_nb = confusion_matrix(y_test, y_pred_nb, labels=categories.unique())\nconf
_matrix_df_nb = pd.DataFrame(conf_matrix_nb, index=categories.unique(), colum
ns=categories.unique())\n\nplt.figure(figsize=(10, 10))\nsns.heatmap(conf_mat
rix_df_nb, annot=True, fmt=\'d\', cmap=\'Blues\')\nplt.xlabel(\'Predicted\')
\nplt.ylabel(\'Actual\')\nplt.title(\'Confusion Matrix for Naive Bayes\')\npl
t.show()\n\n# End time\nend_time = time.time()\nruntime = end_time - start_ti
me\nprint(f"Runtime: {runtime:.2f} seconds")'
```

In [6]:

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, confusion_matrix, classification_r
from sklearn.model_selection import StratifiedKFold, GridSearchCV
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import time

# Define SVM kernels to test
alphas = [0.1, 1.0, 10.0]
best_alpha = None
best_score = 0
results = {}

# Start time
start_time = time.time()

# Perform cross-validation for each kernel
for alpha in alphas:
    print(f"Evaluating alpha: {alpha}")
    pipeline_nb = Pipeline([
        ('vectorizer', tfidf_vectorizer),
        ('classifier', MultinomialNB(alpha=alpha))  # Pass alpha as a keyword
    ])

    # Cross-validation
    skf = StratifiedKFold(n_splits=3, shuffle=True, random_state=123)
    cv_scores = cross_val_score(pipeline_nb, X_train, y_train, cv=skf, n_jobs=

    mean_cv_score = cv_scores.mean()
    results[kernel] = mean_cv_score

    print(f"Mean CV Accuracy for alpha '{alpha}': {mean_cv_score:.4f}")

    # Select the best kernel
    if mean_cv_score > best_score:
        best_score = mean_cv_score
        best_alpha = alpha

print(f"Best alpha: {best_alpha} with CV accuracy: {best_score:.4f}")

# Train the final model on the entire training set using the best kernel
pipeline_nb = Pipeline([
    ('vectorizer', tfidf_vectorizer),
    ('classifier', MultinomialNB(alpha=best_alpha))
])
pipeline_nb.fit(X_train, y_train)

# Predict on the test set
y_pred_nb = pipeline_nb.predict(X_test)

# Get classification report for detailed evaluation
report = classification_report(y_test, y_pred_nb, labels=categories.unique())
print(report)

# Evaluate accuracy
accuracy_nb = accuracy_score(y_test, y_pred_nb)
print(f"Naive Bayes Accuracy: {accuracy_nb * 100:.2f}%")

# Confusion matrix
conf_matrix_nb = confusion_matrix(y_test, y_pred_nb, labels=categories.unique(
conf_matrix_df_nb = pd.DataFrame(conf_matrix_nb, index=categories.unique(), co

plt.figure(figsize=(10, 10))
sns.heatmap(conf_matrix_df_nb, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for Naive Bayes')
plt.show()

# End time
end_time = time.time()
runtime = end_time - start_time
```

```
print(f"Runtime: {runtime:.2f} seconds")
```

```
Evaluating alpha: 0.1
Mean CV Accuracy for alpha '0.1': 0.6638
Evaluating alpha: 1.0
Mean CV Accuracy for alpha '1.0': 0.5858
Evaluating alpha: 10.0
Mean CV Accuracy for alpha '10.0': 0.5107
Best alpha: 0.1 with CV accuracy: 0.6638

C:\Users\asus\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:
1469: UndefinedMetricWarning: Precision and F-score are ill-defined and being
set to 0.0 in labels with no predicted samples. Use `zero_division` parameter
to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\asus\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:
1469: UndefinedMetricWarning: Precision and F-score are ill-defined and being
set to 0.0 in labels with no predicted samples. Use `zero_division` parameter
to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\asus\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:
1469: UndefinedMetricWarning: Precision and F-score are ill-defined and being
set to 0.0 in labels with no predicted samples. Use `zero_division` parameter
to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

                                                    precision    recall  f1
-score   support

                   Public Works and Spatial Planning       0.59      0.91
0.72      2046
Village Development, Rural Areas, and Transmigration       0.81      0.06
0.11       214
                              Economics and Finance       0.55      0.62
0.58      1070
                                             Health       0.77      0.44
0.56       893
                         Population Administration       0.84      0.78
0.81       839
                                  Politics and Law       0.56      0.28
0.38       562
                                         Employment       0.64      0.86
0.73      1705
              Public Order and Community Protection       0.62      0.65
0.63      1131
            Information and communication technology       0.81      0.74
0.77      1308
                        Environmental and forestry       0.86      0.53
0.66       622
                               Social and Welfare       0.64      0.80
0.71      1229
                              Education and culture       0.71      0.83
0.77      1594
                         Agriculture and Livestock       1.00      0.10
0.17        94
                     Energy and Natural Resources       0.81      0.28
0.42       404
                                    Transportation       0.00      0.00
0.00       117
                                         Religion       1.00      0.21
0.35       229
                       LAPOR! Application Problem       0.50      0.04
0.07       526
            Gender Equality and Social Inclusion       0.00      0.00
0.00        39

                                         accuracy
0.67     14622
                                        macro avg       0.65      0.45
0.47     14622
                                     weighted avg       0.68      0.67
0.64     14622

Naive Bayes Accuracy: 66.67%
```



Runtime: 314.79 seconds

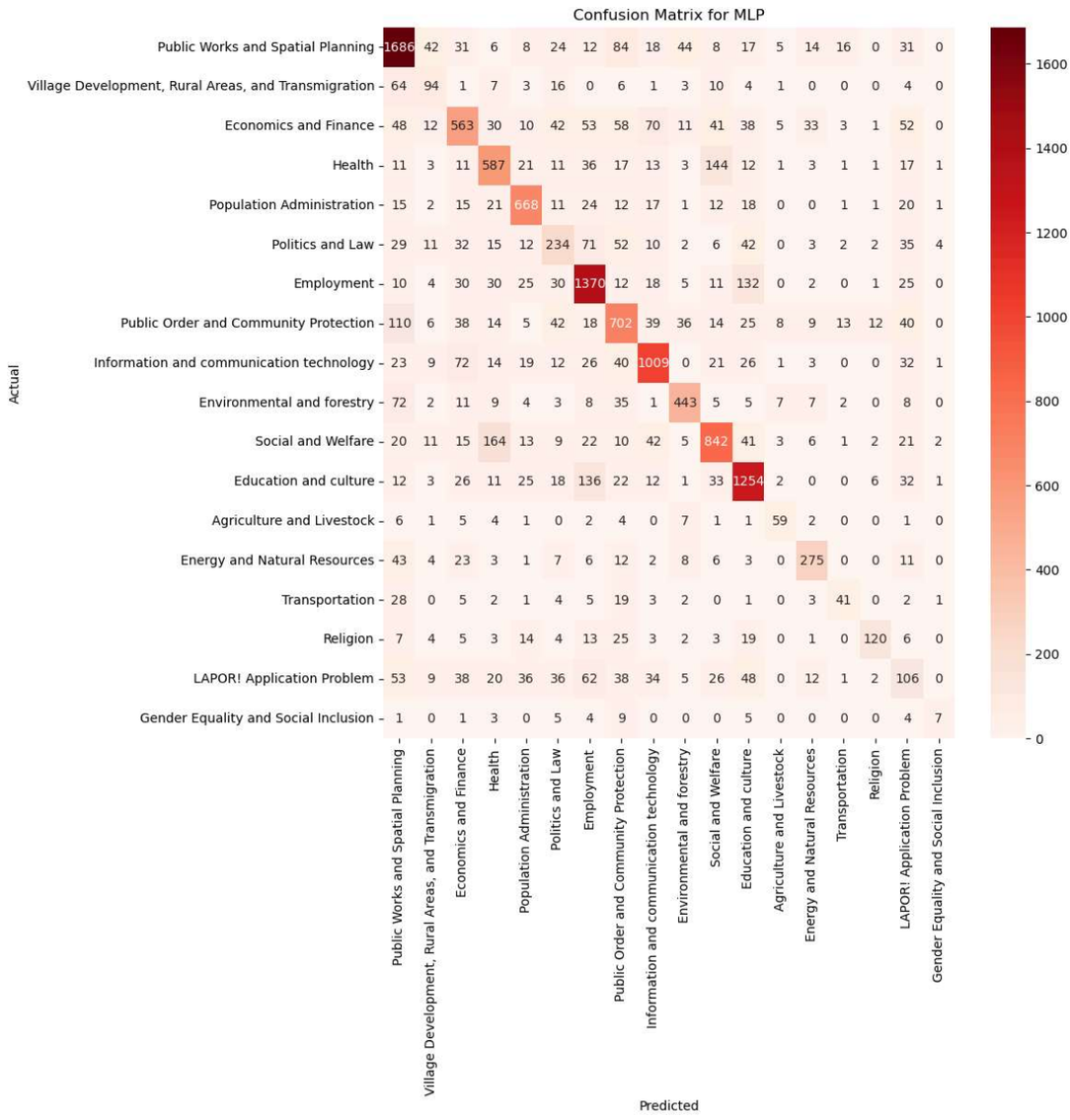
In [ ]:

In [7]:

```
from sklearn.neural_network import MLPClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import time

# Define the pipeline with NMF
pipeline_mlp = Pipeline([
    ('vectorizer', TfidfVectorizer()),
    ('classifier', MLPClassifier(hidden_layer_sizes=[512, 256],  # Three hidde
                                 activation='relu',
                                 solver='adam',
                                 alpha=0.01,
                                 batch_size='auto',
                                 max_iter=50,
                                 tol=0.01,
                                 verbose=2,
                                 n_iter_no_change=5,
                                 validation_fraction=0.2))
])

# Start time
start_time = time.time()

# Train the pipeline
pipeline_mlp.fit(X_train, y_train)

# Predict on the test set
y_pred_mlp = pipeline_mlp.predict(X_test)

# Get classification report for detailed evaluation
report = classification_report(y_test, y_pred_mlp, labels=categories.unique())
print(report)

# Evaluate accuracy
accuracy_mlp = accuracy_score(y_test, y_pred_mlp)
print(f"MLP Accuracy: {accuracy_mlp * 100:.2f}%")

# Confusion matrix
conf_matrix_mlp = confusion_matrix(y_test, y_pred_mlp, labels=categories.uniqu
conf_matrix_df_mlp = pd.DataFrame(conf_matrix_mlp, index=categories.unique(),

plt.figure(figsize=(10, 10))
sns.heatmap(conf_matrix_df_mlp, annot=True, fmt='d', cmap='Reds')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for MLP')
plt.show()

# End time
end_time = time.time()
runtime = end_time - start_time
print(f"Runtime: {runtime:.2f} seconds")
```

```
Iteration 1, loss = 1.38151054
Iteration 2, loss = 0.78811433
Iteration 3, loss = 0.55854076
Iteration 4, loss = 0.44619541
Iteration 5, loss = 0.40268850
Iteration 6, loss = 0.38515672
Iteration 7, loss = 0.38207038
Iteration 8, loss = 0.36923627
Iteration 9, loss = 0.34574955
Iteration 10, loss = 0.33099436
Iteration 11, loss = 0.31661369
Iteration 12, loss = 0.30917904
Iteration 13, loss = 0.29893337
Iteration 14, loss = 0.28571294
Iteration 15, loss = 0.27988227
Iteration 16, loss = 0.26737875
Iteration 17, loss = 0.26226488
Iteration 18, loss = 0.25411599
Iteration 19, loss = 0.24340156
Iteration 20, loss = 0.23738113
Iteration 21, loss = 0.23048493
Iteration 22, loss = 0.22051514
Iteration 23, loss = 0.21778708
Iteration 24, loss = 0.22133749
Iteration 25, loss = 0.21735168
Training loss did not improve more than tol=0.010000 for 5 consecutive epoch
s. Stopping.
                                                    precision    recall  f1
-score   support

                   Public Works and Spatial Planning       0.75      0.82
0.79      2046
Village Development, Rural Areas, and Transmigration       0.43      0.44
0.44       214
                             Economics and Finance       0.61      0.53
0.57      1070
                                            Health       0.62      0.66
0.64       893
                         Population Administration       0.77      0.80
0.78       839
                                Politics and Law       0.46      0.42
0.44       562
                                        Employment       0.73      0.80
0.77      1705
             Public Order and Community Protection       0.61      0.62
0.61      1131
           Information and communication technology       0.78      0.77
0.78      1308
                        Environmental and forestry       0.77      0.71
0.74       622
                               Social and Welfare       0.71      0.69
0.70      1229
                             Education and culture       0.74      0.79
0.76      1594
                          Agriculture and Livestock       0.64      0.63
0.63        94
                       Energy and Natural Resources       0.74      0.68
0.71       404
                                    Transportation       0.51      0.35
0.41       117
                                          Religion       0.81      0.52
0.64       229
                       LAPOR! Application Problem       0.24      0.20
0.22       526
            Gender Equality and Social Inclusion       0.39      0.18
0.25        39

                                        accuracy
0.69     14622
                                       macro avg       0.63      0.59
0.60     14622
                                    weighted avg       0.68      0.69
0.68     14622
```

MLP Accuracy: 68.80%



Runtime: 27213.58 seconds

In [ ]:

In [ ]:

In [8]:
```
'''
import time
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.preprocessing import LabelEncoder
import numpy as np

# Encode labels
le = LabelEncoder()
y_train_enc = le.fit_transform(y_train)
y_test_enc = le.transform(y_test)

# Start time
start_time = time.time()

# Convert text data to TF-IDF vectors (keeping them as sparse matrices)
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

# Convert sparse matrices to sparse tensors
X_train_sparse = tf.sparse.SparseTensor(
    indices=np.vstack((X_train_tfidf.nonzero()[0], X_train_tfidf.nonzero()[1])
    values=X_train_tfidf.data,
    dense_shape=X_train_tfidf.shape
)

X_test_sparse = tf.sparse.SparseTensor(
    indices=np.vstack((X_test_tfidf.nonzero()[0], X_test_tfidf.nonzero()[1])).
    values=X_test_tfidf.data,
    dense_shape=X_test_tfidf.shape
)

# Define the neural network architecture
model = Sequential()
model.add(tf.keras.layers.InputLayer(input_shape=(X_train_tfidf.shape[1],), sp
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(len(le.classes_), activation='softmax'))

model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metric

# Train the model
model.fit(X_train_sparse, y_train_enc, epochs=5, batch_size=32, validation_spl

# Evaluate the model
loss, accuracy = model.evaluate(X_test_sparse, y_test_enc)
print(f"Neural Network Accuracy: {accuracy * 100:.2f}%")

end_time = time.time()
runtime = end_time - start_time
print(f"Runtime: {runtime:.2f} seconds")

# Obtain predictions
y_pred_enc = model.predict(X_test_sparse)
y_pred = np.argmax(y_pred_enc, axis=1)

# Create confusion matrix
conf_matrix = confusion_matrix(y_test_enc, y_pred)
conf_matrix_df = pd.DataFrame(conf_matrix, index=le.classes_, columns=le.class

# Visualize confusion matrix
plt.figure(figsize=(10, 10))
sns.heatmap(conf_matrix_df, annot=True, fmt='d', cmap='Reds')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for Neural Network')
plt.show()'''
```

Out[8]:
```
'\nimport time\nimport tensorflow as tf\nfrom tensorflow.keras.models import
Sequential\nfrom tensorflow.keras.layers import Dense, Dropout\nfrom sklearn.
preprocessing import LabelEncoder\nimport numpy as np\n\n# Encode labels\nle
= LabelEncoder()\ny_train_enc = le.fit_transform(y_train)\ny_test_enc = le.tr
ansform(y_test)\n\n# Start time\nstart_time = time.time()\n\n# Convert text d
ata to TF-IDF vectors (keeping them as sparse matrices)\nX_train_tfidf = tfid
f_vectorizer.fit_transform(X_train)\nX_test_tfidf = tfidf_vectorizer.transfor
m(X_test)\n\n# Convert sparse matrices to sparse tensors\nX_train_sparse = t
f.sparse.SparseTensor(\n    indices=np.vstack((X_train_tfidf.nonzero()[0], X_
train_tfidf.nonzero()[1])).T,\n    values=X_train_tfidf.data,\n    dense_shap
e=X_train_tfidf.shape\n)\n\nX_test_sparse = tf.sparse.SparseTensor(\n    indi
ces=np.vstack((X_test_tfidf.nonzero()[0], X_test_tfidf.nonzero()[1])).T,\n
values=X_test_tfidf.data,\n    dense_shape=X_test_tfidf.shape\n)\n\n# Define
the neural network architecture\nmodel = Sequential()\nmodel.add(tf.keras.lay
ers.InputLayer(input_shape=(X_train_tfidf.shape[1],), sparse=True))\nmodel.ad
d(Dense(512, activation=\'relu\'))\nmodel.add(Dropout(0.5))\nmodel.add(Dense
(256, activation=\'relu\'))\nmodel.add(Dropout(0.5))\nmodel.add(Dense(len(le.
classes_), activation=\'softmax\'))\n\nmodel.compile(loss=\'sparse_categorica
l_crossentropy\', optimizer=\'adam\', metrics=[\'accuracy\'])\n\n# Train the
model\nmodel.fit(X_train_sparse, y_train_enc, epochs=5, batch_size=32, valida
tion_split=0.2)\n\n# Evaluate the model\nloss, accuracy = model.evaluate(X_te
st_sparse, y_test_enc)\nprint(f"Neural Network Accuracy: {accuracy * 100:.2
f}%")\n\nend_time = time.time()\nruntime = end_time - start_time\nprint(f"Run
time: {runtime:.2f} seconds")\n\n# Obtain predictions\ny_pred_enc = model.pre
dict(X_test_sparse)\ny_pred = np.argmax(y_pred_enc, axis=1)\n\n# Create confu
sion matrix\nconf_matrix = confusion_matrix(y_test_enc, y_pred)\nconf_matrix_
df = pd.DataFrame(conf_matrix, index=le.classes_, columns=le.classes_)\n\n# V
isualize confusion matrix\nplt.figure(figsize=(10, 10))\nsns.heatmap(conf_mat
rix_df, annot=True, fmt=\'d\', cmap=\'Reds\')\nplt.xlabel(\'Predicted\')\npl
t.ylabel(\'Actual\')\nplt.title(\'Confusion Matrix for Neural Network\')\npl
t.show()'
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [9]:
```
'''
from sklearn.decomposition import NMF
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, confusion_matrix

# Define SVM kernels to test
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
best_kernel = None
best_score = 0
results = {}

# Start time
start_time = time.time()

# Perform cross-validation for each kernel
for kernel in kernels:
    print(f"Evaluating kernel: {kernel}")
    pipeline_svm = Pipeline([
        ('vectorizer', tfidf_vectorizer),
        ('nmf', NMF(n_components=1400, init='random', random_state=123)),
        ('classifier', SVC(kernel=kernel))
    ])

    # Cross-validation
    skf = StratifiedKFold(n_splits=3, shuffle=True, random_state=123)
    cv_scores = cross_val_score(pipeline_svm, X_train, y_train, cv=skf, n_jobs

    mean_cv_score = cv_scores.mean()
    results[kernel] = mean_cv_score

    print(f"Mean CV Accuracy for kernel '{kernel}': {mean_cv_score:.4f}")

    # Select the best kernel
    if mean_cv_score > best_score:
        best_score = mean_cv_score
        best_kernel = kernel

print(f"Best kernel: {best_kernel} with CV accuracy: {best_score:.4f}")

# Train the final model on the entire training set using the best kernel
pipeline_svm = Pipeline([
    ('vectorizer', tfidf_vectorizer),
    ('nmf', NMF(n_components=1400, init='random', random_state=123)),
    ('classifier', SVC(kernel=best_kernel))
])
pipeline_svm.fit(X_train, y_train)

# Predict on the test set
y_pred_svm = pipeline_svm.predict(X_test)

# Evaluate the model
accuracy_svm = accuracy_score(y_test, y_pred_svm)
print(f"SVM Accuracy on Test Set: {accuracy_svm * 100}%")

# Confusion matrix
conf_matrix_svm = confusion_matrix(y_test, y_pred_svm, labels=categories.uniqu
conf_matrix_df_svm = pd.DataFrame(conf_matrix_svm, index=categories.unique(),

plt.figure(figsize=(10, 10))
sns.heatmap(conf_matrix_df_svm, annot=True, fmt='d', cmap='Greens')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for SVM')
plt.show()

# End time
end_time = time.time()
runtime = end_time - start_time
```

```
print(f"runtime: {runtime:.2f} seconds")'''
```

Out[9]:

```
'\nfrom sklearn.decomposition import NMF\nfrom sklearn.model_selection import StratifiedKFold, cross_val_score\nfrom sklearn.pipeline import Pipeline\nfrom sklearn.svm import SVC\nimport seaborn as sns\nimport matplotlib.pyplot as plt\nfrom sklearn.metrics import accuracy_score, confusion_matrix\n\n# Define SVM kernels to test\nkernels = [\'linear\', \'poly\', \'rbf\', \'sigmoid\']\nbest_kernel = None\nbest_score = 0\nresults = {}\n\n# Start time\nstart_time = time.time()\n\n# Perform cross-validation for each kernel\nfor kernel in kernels:\n    print(f"Evaluating kernel: {kernel}")\n    pipeline_svm = Pipeline([\n        (\'vectorizer\', tfidf_vectorizer),\n        (\'nmf\', NMF(n_components=1400, init=\'random\', random_state=123)),  \n        (\'classifier\', SVC(kernel=kernel))\n    ])\n    \n    # Cross-validation\n    skf = StratifiedKFold(n_splits=3, shuffle=True, random_state=123)\n    cv_scores = cross_val_score(pipeline_svm, X_train, y_train, cv=skf, n_jobs=-1)\n    \n    mean_cv_score = cv_scores.mean()\n    results[kernel] = mean_cv_score\n    \n    print(f"Mean CV Accuracy for kernel \'{kernel}\': {mean_cv_score:.4f}")\n    \n    # Select the best kernel\n    if mean_cv_score > best_score:\n        best_score = mean_cv_score\n        best_kernel = kernel\n\nprint(f"Best kernel: {best_kernel} with CV accuracy: {best_score:.4f}")\n\n# Train the final model on the entire training set using the best kernel\npipeline_svm = Pipeline([\n    (\'vectorizer\', tfidf_vectorizer),\n    (\'nmf\', NMF(n_components=1400, init=\'random\', random_state=123)),  \n    (\'classifier\', SVC(kernel=best_kernel))\n])\npipeline_svm.fit(X_train, y_train)\n\n# Predict on the test set\ny_pred_svm = pipeline_svm.predict(X_test)\n\n# Evaluate the model\naccuracy_svm = accuracy_score(y_test, y_pred_svm)\nprint(f"SVM Accuracy on Test Set: {accuracy_svm * 100}%")\n\n# Confusion matrix\nconf_matrix_svm = confusion_matrix(y_test, y_pred_svm, labels=categories.unique())\nconf_matrix_df_svm = pd.DataFrame(conf_matrix_svm, index=categories.unique(), columns=categories.unique())\n\nplt.figure(figsize=(10, 10))\nsns.heatmap(conf_matrix_df_svm, annot=True, fmt=\'d\', cmap=\'Greens\')\nplt.xlabel(\'Predicted\')\nplt.ylabel(\'Actual\')\nplt.title(\'Confusion Matrix for SVM\')\nplt.show()\n\n# End time\nend_time = time.time()\nruntime = end_time - start_time\nprint(f"runtime: {runtime:.2f} seconds")'
```

In [10]:

```
'''
import gensim.downloader as api

# Load pre-trained word vectors
word_vectors = api.load("glove-wiki-gigaword-100")

# Print the first 10 words in the vocabulary
print("First 10 words in the vocabulary:")
for i, word in enumerate(word_vectors.index_to_key[:1000]):
    print(f"{i + 1}: {word}")

# Print the vector for a specific word
word = 'computer'
if word in word_vectors:
    print(f"\nVector for the word '{word}':\n{word_vectors[word]}")
else:
    print(f"\nThe word '{word}' is not in the vocabulary.")

# Check if a specific word exists in the vocabulary
word_to_check = 'technology'
if word_to_check in word_vectors:
    print(f"\n'{word_to_check}' is in the vocabulary.")
else:
    print(f"\n'{word_to_check}' is not in the vocabulary.")
''''
```

```
  Cell In[10], line 25
    ''''
       ^
SyntaxError: unterminated string literal (detected at line 25)
```

In [ ]:
```
'''
# Print the first 10 words in the vocabulary
print("First 10 words in the vocabulary:")
for i, word in enumerate(word_vectors.index_to_key[:1000]):
    print(f"{i + 1}: {word}")
'''
```

In [ ]:
```
'''
import numpy as np
import gensim.downloader as api

# Load pre-trained word vectors
word_vectors = api.load("glove-wiki-gigaword-100")

def get_average_word2vec(tokens_list, vector, generate_missing=False, k=100):
    if len(tokens_list) < 1:
        return np.zeros(k)
    if generate_missing:
        vectorized = [vector[word] if word in vector else np.random.rand(k) fo
    else:
        vectorized = [vector[word] if word in vector else np.zeros(k) for word
    return np.mean(vectorized, axis=0)

def word2vec_vectorizer(text, vector, k=100):
    tokens = extract_words(text)
    return get_average_word2vec(tokens, vector, k=k)

X_train_w2v = np.array([word2vec_vectorizer(text, word_vectors) for text in X_
X_test_w2v = np.array([word2vec_vectorizer(text, word_vectors) for text in X_t

# Train an SVM classifier
svm = SVC(kernel='linear')
svm.fit(X_train_w2v, y_train)
y_pred_w2v = svm.predict(X_test_w2v)

accuracy_w2v = accuracy_score(y_test, y_pred_w2v)
print(f"Word2Vec + SVM Accuracy: {accuracy_w2v * 100}%")'''
```

In [ ]:

```
'''
from sklearn.feature_extraction.text import TfidfVectorizer
from collections import defaultdict
import numpy as np

# Preprocess and Vectorize Data
tfidf_matrix = tfidf_vectorizer.fit_transform(complaints)
feature_names = tfidf_vectorizer.get_feature_names_out()

# Aggregate Data by Category
category_docs = defaultdict(list)
for doc, category in zip(tfidf_matrix, categories):
    category_docs[category].append(doc)

# Function to get top 10 words for a category
def get_top_words(category_docs, feature_names, top_n=10):
    top_words_by_category = {}
    for category, docs in category_docs.items():
        # Sum TF-IDF scores for all documents in the category
        sum_tfidf = np.sum(docs, axis=0)
        # Convert to dense array
        sum_tfidf_array = sum_tfidf.A1 if hasattr(sum_tfidf, "A1") else sum_tf
        # Get top n indices
        top_n_indices = np.argsort(sum_tfidf_array)[::-1][:top_n]
        # Get corresponding words and scores
        top_words = [(feature_names[i], sum_tfidf_array[i]) for i in top_n_ind
        top_words_by_category[category] = top_words
    return top_words_by_category

# Get the top 10 words for each category
top_words_by_category = get_top_words(category_docs, feature_names)

# Print the top 10 words for each category
for category, top_words in top_words_by_category.items():
    print(f"Top 10 Words for Category: {category}")
    for word, score in top_words:
        print(f"\t- {word} ({score:.4f})")
    print()

# Calculate runtime
end_time = time.time()
runtime = end_time - start_time
print(f"runtime: {runtime:.2f} seconds")'''
```

In [ ]:
```
'''import numpy as np
from sklearn.decomposition import TruncatedSVD
from sklearn.feature_extraction.text import TfidfVectorizer
import matplotlib.pyplot as plt

# Vectorize the text data
tfidf_vectorizer = TfidfVectorizer(analyzer=tokeniser_partial)
X_tfidf = tfidf_vectorizer.fit_transform(complaints)

# Fit TruncatedSVD
svd = TruncatedSVD(n_components=min(2000, X_tfidf.shape[1] - 1))
svd.fit(X_tfidf)
explained_variance_ratio = np.cumsum(svd.explained_variance_ratio_)

# Plot explained variance ratio
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(explained_variance_ratio) + 1), explained_variance_ratio
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Explained Variance vs Number of Components')
plt.grid()
plt.show()

# Determine the number of components for 90% and 95% variance
n_components_90 = np.argmax(explained_variance_ratio >= 0.80) + 1
n_components_95 = np.argmax(explained_variance_ratio >= 0.90) + 1

print(f"Number of components for 90% variance: {n_components_90}")
print(f"Number of components for 95% variance: {n_components_95}")
'''
```

In [ ]:
```
'''
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.preprocessing import LabelEncoder

# Encode labels
le = LabelEncoder()
y_train_enc = le.fit_transform(y_train)
y_test_enc = le.transform(y_test)

#start time
start_time = time.time()

# Convert text data to TF-IDF vectors
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train).toarray()
X_test_tfidf = tfidf_vectorizer.transform(X_test).toarray()

# Define the neural network architecture
model = Sequential()
model.add(Dense(512, input_shape=(X_train_tfidf.shape[1],), activation='relu')
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(len(le.classes_), activation='softmax'))

model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metric

# Train the model
model.fit(X_train_tfidf, y_train_enc, epochs=5, batch_size=32, validation_spli

# Evaluate the model
loss, accuracy = model.evaluate(X_test_tfidf, y_test_enc)
print(f"Neural Network Accuracy: {accuracy * 100}%")

end_time = time.time()
runtime = end_time - start_time
print (f"runtime: {runtime:.2f} second")'''
```

In [ ]: