In [17]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
import string
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer
from sklearn.decomposition import NMF
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import matplotlib.colors as mcolors
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.exceptions import ConvergenceWarning
import warnings
import nltk
import re
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer

# Download nltk resources
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
```

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

Out[17]: True

In [22]:

```
# Load the CSV file into a DataFrame
csv_file_path = r'C:\Users\abcd\--------\2016_2020_final.csv'
data = pd.read_csv(csv_file_path, encoding='ISO-8859-1')

# Set up stopwords
stop_words = set(stopwords.words('english'))

# Define custom stopwords
custom_stop_words = {'paper', 'mori', 'mekong', 'atoll'}

# Combine custom stopwords with the NLTK stopwords
all_stop_words = stop_words.union(custom_stop_words)

# Function to preprocess text
def preprocess_text(text):
    if pd.isnull(text):
        return ""
    else:
        # Convert to lowercase
        text = text.lower()

        # Remove punctuation
        text = ''.join([c for c in text if c not in string.punctuation and c != '’' and c != '‘' and c != '”' and c != '“'])

        # Tokenize the text into words
        words = nltk.word_tokenize(text)

        # Remove stopwords and words with less than three letters
        words = [word for word in words if word not in all_stop_words and len(word) > 3]

        # Lemmatize the words
        lemmatizer = WordNetLemmatizer()
        words = [lemmatizer.lemmatize(word) for word in words]

        # Join the preprocessed words back into a single text
        text = ' '.join(words)

        return text
```

```python
# Function to remove non-alphabetic characters
def remove_non_alphabets(text):
    # Remove non-alphabetic characters using regular expression
    text_without_non_alphabets = re.sub(r'[^a-zA-Z\s]', '', text)
    return text_without_non_alphabets

# Function to remove numbers
def remove_numbers(text):
    # Remove numbers using regular expression
    text_without_numbers = re.sub(r'\b\d+\b', '', text)
    return text_without_numbers

# Function to remove texts with encoding issues
def remove_texts_with_encoding_issues(texts):
    cleaned_texts = []

    for text in texts:
        # Remove non-ASCII characters
        cleaned_text = text.encode('ascii', 'ignore').decode('utf-8')

        # Check if the cleaned text is empty or contains only whitespace
        if cleaned_text.strip():
            cleaned_texts.append(cleaned_text)

    return cleaned_texts

# Function to remove conjunctions
def remove_conjunctions(text):
    # Tokenize the text into words
    tokens = word_tokenize(text)

    # Define the conjunction words to be removed
    conjunction_words = ['and', 'or', 'through', 'that', 'with', 'but', 'while', 'because']

    # Remove conjunction words
    filtered_text = [word for word in tokens if word.lower() not in conjunction_words]

    # Join the filtered words back into a single text
    filtered_text = ' '.join(filtered_text)

    return filtered_text

# Function to remove wh-questions
def remove_wh_questions(text):
    # Define the pattern to match wh-questions
    pattern = r'\b(what|where|when|who|which|why|how)\b'

    # Remove wh-questions using regular expression substitution
    filtered_text = re.sub(pattern, '', text, flags=re.IGNORECASE)

    return filtered_text

# Function to remove clauses
def remove_clauses(text):
    # Define the pattern to match clauses (dependent clauses)
    pattern = r'\b(when|where|while|although|because|if|since|unless|until|that|who|which|whom|whose)\b'

    # Remove clauses using regular expression substitution
    filtered_text = re.sub(pattern, '', text, flags=re.IGNORECASE)

    return filtered_text

# Function to remove stop words
def remove_stop_words(text):
    # Tokenize the text into words
    tokens = word_tokenize(text)

    # Remove stop words
    filtered_text = [word for word in tokens if word.lower() not in stop_words]

    # Join the filtered words back into a single text
    filtered_text = ' '.join(filtered_text)

    return filtered_text

# Function to lemmatize a word based on its part-of-speech tag
def lemmatize_word(word, pos):
    lemmatizer = WordNetLemmatizer()
    pos_tag = get_wordnet_pos(pos)
    if pos_tag is None:
        # If the part-of-speech tag is not recognized, default to noun (for better coverage)
        pos_tag = wordnet.NOUN
    return lemmatizer.lemmatize(word, pos=pos_tag)

# Function to get WordNet POS tags from Penn Treebank POS tags
def get_wordnet_pos(treebank_tag):
    if treebank_tag.startswith('J'):
        return wordnet.ADJ
    elif treebank_tag.startswith('V'):
        return wordnet.VERB
    elif treebank_tag.startswith('N'):
        return wordnet.NOUN
    elif treebank_tag.startswith('R'):
        return wordnet.ADV
    else:
        return None


    # Apply preprocessing to the 'Combined' column
data['processed_Combined'] = data['Combined'].apply(preprocess_text)

# Remove non-alphabetic characters
data['processed_Combined'] = data['processed_Combined'].apply(remove_non_alphabets)

# Remove numbers
data['processed_Combined'] = data['processed_Combined'].apply(remove_numbers)

# Remove texts with encoding issues
data['processed_Combined'] = remove_texts_with_encoding_issues(data['processed_Combined'])

# Remove conjunctions
data['processed_Combined'] = data['processed_Combined'].apply(remove_conjunctions)

# Remove wh-questions
data['processed_Combined'] = data['processed_Combined'].apply(remove_wh_questions)

# Remove clauses
data['processed_Combined'] = data['processed_Combined'].apply(remove_clauses)

# Remove stop words
```

```
data['processed_Combined'] = data['processed_Combined'].apply(remove_stop_words)

# Tokenize and Lemmatize
data['Combined_tokens'] = data['processed_Combined'].apply(word_tokenize)
#data['Combined_tokens'] = data['Combined_tokens'].apply(preprocess_tokens)

# Function to convert list of words into a single string
def list_to_string(words_list):
    return ' '.join(words_list)

# Convert the lists of lemmatized words to strings
data['Combined_lemmatized_words'] = data['Combined_tokens'].apply(list_to_string)

# Create TF-IDF matrix
corpus = data['Combined_lemmatized_words']
vectorizer = TfidfVectorizer(min_df=1, max_features=2000)  # Adjust min_df and max_features as needed
X = vectorizer.fit_transform(corpus)
```

In [25]:

```
# Set the maximum number of clusters to evaluate
max_num_clusters = 25

# Set the desired maximum number of iterations
max_iter = 1000

# Initialize lists to store the silhouette scores and filtered number of topics
silhouette_scores = []
filtered_num_topics_list = []

# Regularization parameter (l1_ratio) for NMF sparsity regularization
l1_ratio = 0.9 # You can experiment with different values between 0 and 1

# Iterate over the topic range and calculate the silhouette score for each number of topics
for num_topics in range(2, max_num_clusters + 1):
    # Fit NMF model with the current number of topics and apply sparsity regularization
    nmf = NMF(n_components=num_topics, max_iter=max_iter, random_state=42, l1_ratio=l1_ratio)
    nmf.fit(X)

    # Perform NMF with the current number of clusters
    cluster_labels = nmf.transform(X).argmax(axis=1)
    # Calculate the number of documents associated with each topic
    topic_counts = np.bincount(cluster_labels)

    # Filter out topics that have fewer documents than the threshold
    min_documents_threshold = 30
    filtered_topics = [topic_idx for topic_idx, count in enumerate(topic_counts) if count >= min_documents_threshold]

    # Calculate the silhouette score only if the number of filtered clusters is greater than 1
    if len(filtered_topics) > 1:
        silhouette_avg = silhouette_score(X, cluster_labels)
        silhouette_scores.append(silhouette_avg)
        filtered_num_topics_list.append(num_topics)

                # Create a DataFrame to store the silhouette scores
silhouette_df = pd.DataFrame({'Num_Clusters': filtered_num_topics_list, 'Silhouette_Score': silhouette_scores})
```

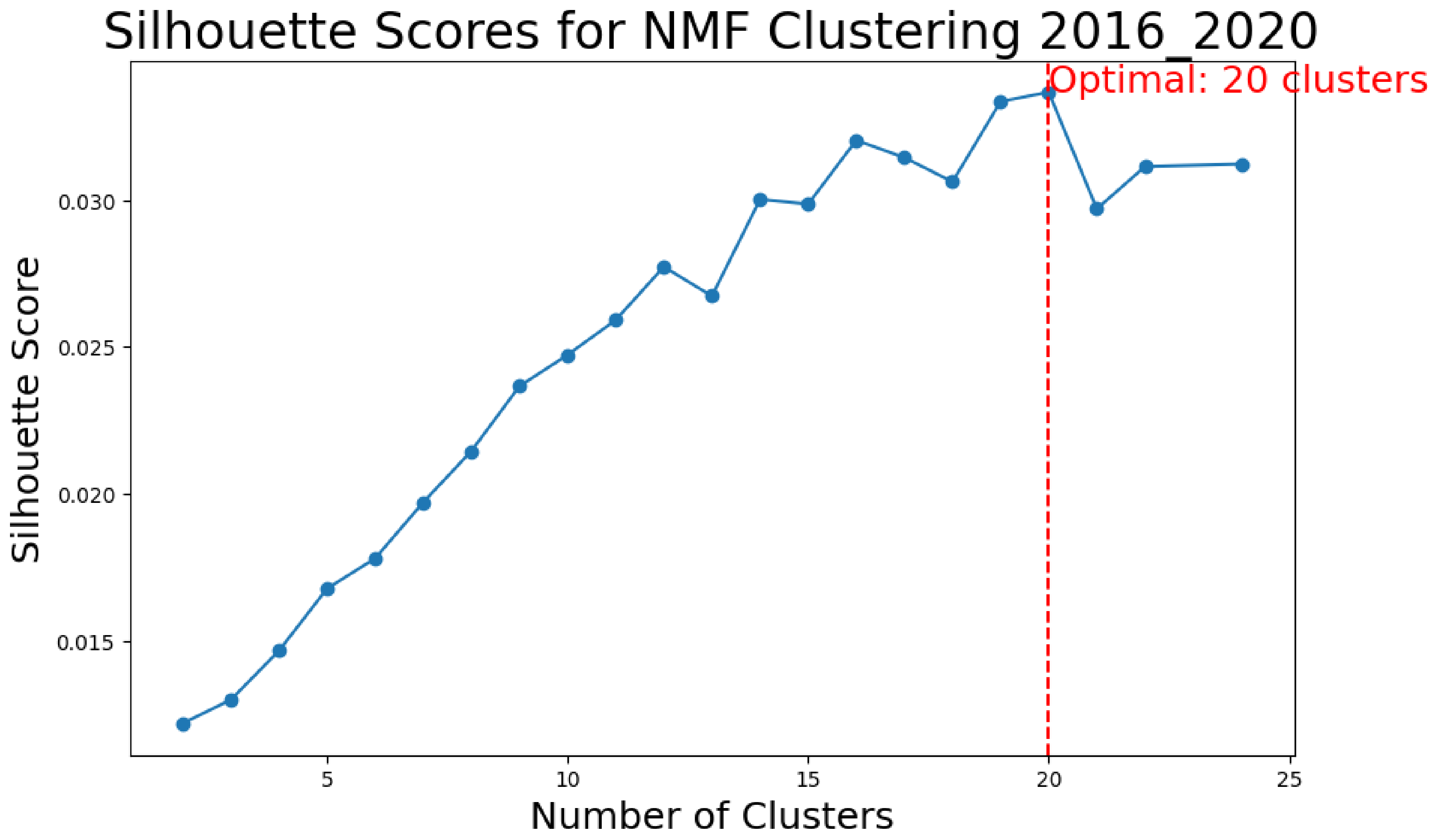
In [26]:

```
# Plot the data
plt.figure(figsize=(10, 6))
plt.plot(df['Num_Clusters'], df['Silhouette_Score'], marker='o')
plt.xlabel('Number of Clusters', fontsize=18)
plt.ylabel('Silhouette Score', fontsize=18)
plt.title('Silhouette Scores for NMF Clustering 2016_2020', fontsize=24)

# Annotate the optimal point
optimal_num_clusters = 20  # Change this to the actual optimal number of clusters
plt.annotate(f'Optimal: {optimal_num_clusters} clusters', xy=(optimal_num_clusters, max(df['Silhouette_Score'])),
             xytext=(optimal_num_clusters, max(df['Silhouette_Score']) + 0), color='red', fontsize=18)

# Add a vertical line at the optimal point
plt.axvline(x=optimal_num_clusters, linestyle='--', color='red')

plt.show()
```



In [27]:

```
# Fit NMF model with the optimal number of topics and apply sparsity regularization
num_topics = optimal_num_topics
nmf = NMF(n_components=num_topics, max_iter=max_iter, random_state=42, l1_ratio=l1_ratio)
nmf.fit(X)

# Get the top terms for each topic
feature_names = vectorizer.get_feature_names_out()
top_terms_dict = {}
```

```
for topic_idx, topic in enumerate(nmf.components_):
    top_terms = [feature_names[i] for i in topic.argsort()[:-10:-1]]
    top_terms_dict[topic_idx] = top_terms
    print(f"C {topic_idx + 1}: {top_terms}\n")

# Transform documents to topic distribution
document_topics = nmf.transform(X)

# Get the document-topic matrix from NMF
document_topics = nmf.transform(X)

# Find the dominant topic for each document
dominant_topics = np.argmax(document_topics, axis=1)

# Count the occurrences of each topic label
topic_counts = np.bincount(dominant_topics)

# Print the number of documents in each cluster
for cluster_id, count in enumerate(topic_counts, start=1):
    print(f"C {cluster_id}: {count} articles")
```

```
C 1: ['research', 'transdisciplinary', 'project', 'knowledge', 'stakeholder', 'process', 'learning', 'researcher', 'problem']

C 2: ['climate', 'adaptation', 'change', 'risk', 'uncertainty', 'mindfulness', 'planning', 'assessment', 'mitigation']

C 3: ['value', 'social', 'relational', 'valuation', 'concept', 'intrinsic', 'people', 'theory', 'perspective']

C 4: ['sdgs', 'goal', 'development', 'sustainable', 'indicator', 'target', 'progress', 'agenda', 'assessment']

C 5: ['scenario', 'future', 'geoengineering', 'positive', 'population', 'land', 'alternative', 'change', 'landuse']

C 6: ['sustainability', 'science', 'digital', 'transition', 'system', 'discipline', 'research', 'concept', 'thinking']

C 7: ['landscape', 'agroforestry', 'management', 'tree', 'mediterranean', 'system', 'land', 'landuse', 'conservation']

C 8: ['water', 'governance', 'nexus', 'supply', 'management', 'resource', 'demand', 'capability', 'mining']

C 9: ['blue', 'growth', 'economy', 'degrowth', 'fishery', 'marine', 'coastal', 'smallscale', 'fishing']

C 10: ['delta', 'vulnerability', 'amazon', 'region', 'coastal', 'river', 'population', 'flood', 'change']

C 11: ['place', 'meaning', 'sense', 'placeshaping', 'transformative', 'attachment', 'placebased', 'stewardship', 'transformat
ion']

C 12: ['cultural', 'selection', 'multilevel', 'evolution', 'evolutionary', 'group', 'institution', 'grouplevel', 'resource']

C 13: ['conflict', 'environmental', 'justice', 'movement', 'social', 'distribution', 'violence', 'injustice', 'ecological']

C 14: ['indigenous', 'knowledge', 'science', 'local', 'river', 'western', 'community', 'traditional', 'protocol']

C 15: ['food', 'household', 'security', 'consumption', 'production', 'crop', 'healthy', 'health', 'sharing']

C 16: ['service', 'ecosystem', 'assessment', 'biodiversity', 'ecological', 'payment', 'restoration', 'underuse', 'provisionin
g']

C 17: ['urban', 'city', 'myth', 'public', 'citizen', 'experiment', 'transition', 'governance', 'data']

C 18: ['education', 'future', 'program', 'learning', 'educational', 'competency', 'curriculum', 'sustainable', 'student']

C 19: ['trap', 'socialecological', 'human', 'response', 'system', 'ecological', 'lake', 'policy', 'subjectivity']

C 20: ['resilience', 'community', 'disturbance', 'framework', 'socialecological', 'social', 'adaptive', 'system', 'data']
```

```
C 21: ['forest', 'conservation', 'biomass', 'local', 'bioenergy', 'redd', 'policy', 'private', 'protection']

C 22: ['model', 'system', 'policy', 'dynamic', 'sustainable', 'dimension', 'business', 'simulation', 'triple']

C 23: ['capital', 'natural', 'wealth', 'inclusive', 'index', 'region', 'rural', 'japan', 'asset']

C 1: 19 articles
C 2: 20 articles
C 3: 23 articles
C 4: 28 articles
C 5: 14 articles
C 6: 54 articles
C 7: 15 articles
C 8: 18 articles
C 9: 15 articles
C 10: 16 articles
C 11: 19 articles
C 12: 12 articles
C 13: 20 articles
C 14: 20 articles
C 15: 16 articles
C 16: 13 articles
C 17: 14 articles
C 18: 25 articles
C 19: 6 articles
C 20: 18 articles
C 21: 23 articles
C 22: 31 articles
C 23: 19 articles
```

In [ ]:

```python
from collections import Counter

# Flatten the top terms from each topic
all_top_terms = [term for terms in top_terms_dict.values() for term in terms]

# Count the frequency of each top term
term_counts = Counter(all_top_terms)

# Print the top terms for each topic along with their frequencies
for topic_idx, top_terms in top_terms_dict.items():
    sorted_terms = sorted(top_terms, key=lambda term: term_counts[term], reverse=True)
    term_frequency = [f"{term} ({term_counts[term]})" for term in sorted_terms]
    print(f"Topic {topic_idx + 1}: {', '.join(term_frequency)}\n")
```

```
Topic 1: research (2), knowledge (2), learning (2), transdisciplinary (1), project (1), stakeholder (1), process (1), researc
her (1), problem (1)

Topic 2: change (3), assessment (3), climate (1), adaptation (1), risk (1), uncertainty (1), mindfulness (1), planning (1), m
itigation (1)

Topic 3: social (3), concept (2), value (1), relational (1), valuation (1), intrinsic (1), people (1), theory (1), perspectiv
e (1)

Topic 4: sustainable (3), assessment (3), sdgs (1), goal (1), development (1), indicator (1), target (1), progress (1), agend
a (1)

Topic 5: change (3), future (2), population (2), land (2), landuse (2), scenario (1), geoengineering (1), positive (1), alter
native (1)

Topic 6: system (5), science (2), transition (2), research (2), concept (2), sustainability (1), digital (1), discipline (1),
thinking (1)

Topic 7: system (5), management (2), land (2), landuse (2), conservation (2), landscape (1), agroforestry (1), tree (1), medi
terranean (1)

Topic 8: governance (2), management (2), resource (2), water (1), nexus (1), supply (1), demand (1), capability (1), mining
(1)

Topic 9: coastal (2), blue (1), growth (1), economy (1), degrowth (1), fishery (1), marine (1), smallscale (1), fishing (1)

Topic 10: change (3), region (2), coastal (2), river (2), population (2), delta (1), vulnerability (1), amazon (1), flood (1)

Topic 11: place (1), meaning (1), sense (1), placeshaping (1), transformative (1), attachment (1), placebased (1), stewardshi
p (1), transformation (1)

Topic 12: resource (2), cultural (1), selection (1), multilevel (1), evolution (1), evolutionary (1), group (1), institution
(1), grouplevel (1)

Topic 13: social (3), ecological (3), conflict (1), environmental (1), justice (1), movement (1), distribution (1), violence
(1), injustice (1)

Topic 14: knowledge (2), science (2), local (2), river (2), community (2), indigenous (1), western (1), traditional (1), prot
ocol (1)

Topic 15: food (1), household (1), security (1), consumption (1), production (1), crop (1), healthy (1), health (1), sharing
```

```
(1)

Topic 16: assessment (3), ecological (3), service (1), ecosystem (1), biodiversity (1), payment (1), restoration (1), underus
e (1), provisioning (1)

Topic 17: transition (2), governance (2), data (2), urban (1), city (1), myth (1), public (1), citizen (1), experiment (1)

Topic 18: sustainable (3), future (2), learning (2), education (1), program (1), educational (1), competency (1), curriculum
(1), student (1)

Topic 19: system (5), ecological (3), policy (3), socialecological (2), trap (1), human (1), response (1), lake (1), subjecti
vity (1)

Topic 20: system (5), social (3), community (2), socialecological (2), data (2), resilience (1), disturbance (1), framework
(1), adaptive (1)

Topic 21: policy (3), conservation (2), local (2), forest (1), biomass (1), bioenergy (1), redd (1), private (1), protection
(1)

Topic 22: system (5), policy (3), sustainable (3), model (1), dynamic (1), dimension (1), business (1), simulation (1), tripl
e (1)

Topic 23: region (2), capital (1), natural (1), wealth (1), inclusive (1), index (1), rural (1), japan (1), asset (1)
```