PROBLEM 1

In [1]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.model_selection import ParameterGrid

clickbait = pd.read_csv("clickbait.txt", sep="\t", header=None, names =['dataset'] )
notclickbait = pd.read_csv("not-clickbait.txt", sep="\t", header=None, names =['dataset'])
clickbait['Y'] = 1
notclickbait['Y'] = 0
#print(clickbait)
#print(notclickbait)

# Combine the content of both files
combine = pd.concat([clickbait, notclickbait])
arr = combine.to_numpy()
#print(combine)
# Shuffle the list of lines
np.random.shuffle(arr)
#print(arr)
shuffle = pd.DataFrame(arr, columns = ['dataset', 'Y'])
#print(shuffle)
#lenght of test,train and validate
test_len = int((20/100)* len(arr))
print("Length of test data:",test_len)
train_len = int((72/100)* len(arr))
print("Length of train data:",train_len)
validate_len = int((8/100)* len(arr))
print("Length of validate data:",validate_len)
print("\n")

test_data = shuffle.iloc[0:(test_len)]
train_data = shuffle.loc[(test_len):(test_len+train_len)]
```

```
validate_data = shuffle.iloc[(test_len+train_len+1):]
#print(test_data)
#print(train_data)
#print(validate_data)

test_target_rate = (test_data['Y'].mean())*100
print("Target rate of test data:", test_target_rate,'%')
train_target_rate =(train_data['Y'].mean())*100
print("Target rate of train data:", train_target_rate,'%')
validation_target_rate =(validate_data['Y'].mean())*100
print("Target rate of validation data:", validation_target_rate,'%')
print("\n")
```

```
Length of test data: 477
Length of train data: 1719
Length of validate data: 191


Target rate of test data: 36.477987421383645 %
Target rate of train data: 34.127906976744185 %
Target rate of validation data: 27.748691099476442 %
```

PROBLEM 3

In [2]:

```
pipeline = Pipeline([
    ('vectorizer', CountVectorizer(ngram_range=(1, 2))),  # Include unigrams and bigrams
    ('classifier', MultinomialNB())  # Naive Bayes classifier
])

X_train = train_data['dataset']
y_train = train_data['Y'].astype(int)
X_validate = validate_data['dataset']
y_validate = validate_data['Y'].astype(int)
X_test = test_data['dataset']
y_test = test_data['Y'].astype(int)
# Fit the classifier on the training set
pipeline.fit(X_train, y_train)

# Predict on training and validation sets
```

```
y_train_pred = pipeline.predict(X_train)
y_validate_pred = pipeline.predict(X_validate)

# Calculate precision, recall, and F1-score for the training set
precision_train = precision_score(y_train, y_train_pred)
recall_train = recall_score(y_train, y_train_pred)
f1_train = f1_score(y_train, y_train_pred)

# Calculate precision, recall, and F1-score for the validation set
precision_validate = precision_score(y_validate, y_validate_pred)
recall_validate = recall_score(y_validate, y_validate_pred)
f1_validate = f1_score(y_validate, y_validate_pred)

# Print the results
print("Training Set Metrics:")
print(f"Precision: {precision_train:.2f}")
print(f"Recall: {recall_train:.2f}")
print(f"F1-Score: {f1_train:.2f}")
print("\nValidation Set Metrics:")
print(f"Precision: {precision_validate:.2f}")
print(f"Recall: {recall_validate:.2f}")
print(f"F1-Score: {f1_validate:.2f}")
```

```
Training Set Metrics:
Precision: 0.99
Recall: 1.00
F1-Score: 1.00

Validation Set Metrics:
Precision: 0.79
Recall: 0.91
F1-Score: 0.84
```

PROBLEM 4

In [4]:

```
param_grid = {
    'vectorizer__max_df': [0.5, 0.75, 1.0],          # Vary max_df
    'classifier__alpha': [0.1, 0.5, 1.0],            # Vary alpha (smoothing)
    'vectorizer__ngram_range': [(1, 1), (1, 2)]      # Include or exclude bigrams
}
```

```python
# Initialize a list to store results
results = []

# Iterate through parameter combinations
for params in ParameterGrid(param_grid):
    # Create a pipeline with the specified parameters
    pipeline = Pipeline([
        ('vectorizer', CountVectorizer(max_df=params['vectorizer__max_df'], ngram_range=params['vectorizer__ngram_range'])),
        ('classifier', MultinomialNB(alpha=params['classifier__alpha']))
    ])

    # Fit the pipeline on the training data
    pipeline.fit(X_train, y_train)

    # Make predictions on the validation set
    y_validate_pred = pipeline.predict(X_validate)

    # Calculate metrics
    precision = precision_score(y_validate, y_validate_pred)
    recall = recall_score(y_validate, y_validate_pred)
    f1 = f1_score(y_validate, y_validate_pred)

    # Store results
    results.append({
        'max_df': params['vectorizer__max_df'],
        'alpha': params['classifier__alpha'],
        'include_bigrams': 'Yes' if params['vectorizer__ngram_range'] == (1, 2) else 'No',
        'precision': precision,
        'recall': recall,
        'f1': f1
    })

# Create a DataFrame with the results
results_df = pd.DataFrame(results)

# Sort the DataFrame by F1-score in descending order
sorted_results = results_df.sort_values(by='f1', ascending=False)

# Display the highest and lowest results
print("Highest F1-Score Configuration:")
print(sorted_results.head(1))
print("\n")
print("\nLowest F1-Score Configuration:")
print(sorted_results.tail(1))
print("\n")
print(sorted_results[['max_df', 'alpha', 'include_bigrams', 'precision', 'recall', 'f1']])
```

```
Highest F1-Score Configuration:
    max_df  alpha include_bigrams  precision   recall        f1
17     1.0    1.0             Yes   0.786885  0.90566  0.842105



Lowest F1-Score Configuration:
   max_df  alpha include_bigrams  precision    recall        f1
3    0.75    0.1             Yes   0.731343  0.924528  0.816667


    max_df  alpha include_bigrams  precision    recall        f1
17    1.00    1.0             Yes   0.786885  0.905660  0.842105
15    0.75    1.0             Yes   0.786885  0.905660  0.842105
13    0.50    1.0             Yes   0.786885  0.905660  0.842105
11    1.00    0.5             Yes   0.765625  0.924528  0.837607
16    1.00    1.0              No   0.765625  0.924528  0.837607
14    0.75    1.0              No   0.765625  0.924528  0.837607
12    0.50    1.0              No   0.765625  0.924528  0.837607
9     0.75    0.5             Yes   0.765625  0.924528  0.837607
7     0.50    0.5             Yes   0.765625  0.924528  0.837607
8     0.75    0.5              No   0.753846  0.924528  0.830508
10    1.00    0.5              No   0.753846  0.924528  0.830508
6     0.50    0.5              No   0.753846  0.924528  0.830508
4     1.00    0.1              No   0.742424  0.924528  0.823529
2     0.75    0.1              No   0.742424  0.924528  0.823529
0     0.50    0.1              No   0.742424  0.924528  0.823529
1     0.50    0.1             Yes   0.731343  0.924528  0.816667
5     1.00    0.1             Yes   0.731343  0.924528  0.816667
3     0.75    0.1             Yes   0.731343  0.924528  0.816667
```

PROBLEM 5

```
In [5]: best_model_params = sorted_results.iloc[0]
```

```
# Create a pipeline with the parameters of the selected model
selected_model_pipeline = Pipeline([
    ('vectorizer', CountVectorizer(max_df=best_model_params['max_df'], ngram_range=(1, 2))),
    ('classifier', MultinomialNB(alpha=best_model_params['alpha']))
])

# Fit the selected model on the training data
selected_model_pipeline.fit(X_train, y_train)

# Make predictions on the test set
y_test_pred = selected_model_pipeline.predict(X_test)

# Calculate precision, recall, and F1-score on the test set
precision_test = precision_score(y_test, y_test_pred)
recall_test = recall_score(y_test, y_test_pred)
f1_test = f1_score(y_test, y_test_pred)

# Display the results
print("Test Set Metrics for the Selected Model:")
print(f"Precision: {precision_test:.2f}")
print(f"Recall: {recall_test:.2f}")
print(f"F1-Score: {f1_test:.2f}")
```

```
Test Set Metrics for the Selected Model:
Precision: 0.88
Recall: 0.88
F1-Score: 0.88
```

PROBLEM 6

In [6]:

```
vectorizer = selected_model_pipeline.named_steps['vectorizer']
vocabulary = vectorizer.get_feature_names_out()

# Get the log probabilities for each word (unigram) in the vocabulary
log_probabilities = selected_model_pipeline.named_steps['classifier'].feature_log_prob_[1]

# Calculate the log-probability differences between clickbait and non-clickbait classes
log_prob_differences = log_probabilities - selected_model_pipeline.named_steps['classifier'].feature_log_prob_[0]

# Create a DataFrame to store the words and their log-probability differences
import pandas as pd
```

```python
word_prob_df = pd.DataFrame({'word': vocabulary, 'log_prob_difference': log_prob_differences})

# Sort the DataFrame by log-probability difference in descending order
sorted_word_prob_df = word_prob_df.sort_values(by='log_prob_difference', ascending=False)

# Get the top 5 words as strong clickbait indicators
top_clickbait_indicators = sorted_word_prob_df.head(5)

# Display the list of top clickbait indicators
print("Top 5 Clickbait Indicators:")
print(top_clickbait_indicators['word'].tolist())


# Get the log probabilities for each word (unigram) in the vocabulary
log_probabilities = selected_model_pipeline.named_steps['classifier'].feature_log_prob_[1]

# Create a DataFrame to store the words and their log-probabilities
import pandas as pd
word_prob_df = pd.DataFrame({'word': vocabulary, 'log_prob': log_probabilities})

# Sort the DataFrame by log-probability in descending order
sorted_word_prob_df = word_prob_df.sort_values(by='log_prob', ascending=False)

# Get the top 5 unigrams as strong clickbait indicators
top_unigram_clickbait_indicators = sorted_word_prob_df.head(5)

# Display the list of top unigram clickbait indicators
print("Top 5 Unigram Clickbait Indicators:")
print(top_unigram_clickbait_indicators['word'].tolist())
```

```
Top 5 Clickbait Indicators:
['you won', 'won believe', 'here', 'you ll', 'll never']
Top 5 Unigram Clickbait Indicators:
['the', 'you', 'to', 'this', 'is']
```

PROBLEM 7

In [12]:

```python
import re

# Define the top 5 keywords as a list
top_keywords = top_unigram_clickbait_indicators['word'].tolist()
```

```python
# Create a regular expression pattern to match any of the top keywords with word boundaries
pattern = r'\b(?:' + '|'.join(map(re.escape, top_keywords)) + r')\b'

true_positives = 0  # Correctly detected clickbait
false_positives = 0  # Incorrectly detected as clickbait
false_negatives = 0  # Clickbait not detected

for text, label in zip(X_test, y_test):
    match = re.search(pattern, text)
    if match:
        if label == 1:
            true_positives += 1
        else:
            false_positives += 1
    elif label == 1:
        false_negatives += 1

# Calculate precision and recall
precision = true_positives / (true_positives + false_positives)
recall = true_positives / (true_positives + false_negatives)

# Display the results
print("Precision:", precision)
print("Recall:", recall)
```

```
Precision: 0.35106382978723405
Recall: 0.3793103448275862
```

```
In [ ]:
```

```
In [ ]:
```