In [19]:
```
# Import necessary libraries
import subprocess
import sys
import argparse
import json
import os
import gc
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import PyPDF2
import re
import nltk
import emoji
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline
import logging
import traceback
```

In [20]:
```
# Download necessary NLTK data
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('vader_lexicon')
nltk.download('omw-1.4')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\ELITEBOOK\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\ELITEBOOK\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\ELITEBOOK\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\ELITEBOOK\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\ELITEBOOK\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

Out[20]:
```
True
```

In [21]:
```
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)
```

```
# Function to extract sentences from PDFs using PyPDF2
def read_pdf_sentences(file_path):
    sentences = []
    try:
        with open(file_path, "rb") as file:
            reader = PyPDF2.PdfReader(file)
            for page in reader.pages:
                text = page.extract_text()
                if text:
                    sentences.extend(sent_tokenize(text))
    except Exception as e:
        print(f"Error reading PDF file {file_path}: {str(e)}")
    return sentences

def extract_and_merge(pdf_path, csv_path):
    try:
        print(f"Attempting to read PDF files from: {pdf_path}")
        if not os.path.exists(pdf_path):
            raise FileNotFoundError(f"PDF directory not found: {pdf_path}")

        pdf_files = [os.path.join(pdf_path, file) for file in os.listdir(pdf_path) if
        print(f"Found {len(pdf_files)} PDF files")

        pdf_sentences = []
        for file in pdf_files:
            pdf_sentences.extend(read_pdf_sentences(file))

        print(f"Extracted {len(pdf_sentences)} sentences from PDF files")
        pdf_df = pd.DataFrame({'content': pdf_sentences})

        print(f"Attempting to read CSV file: {csv_path}")
        if not os.path.exists(csv_path):
            raise FileNotFoundError(f"CSV file not found: {csv_path}")

        news_data = pd.read_csv(csv_path, encoding='latin1')
        content_column = next((col for col in news_data.columns if col.lower().strip()
        if content_column is None:
            raise KeyError(f"No 'content' column found in the CSV file: {csv_path}")

        news_data_paragraphs = []
        for content in news_data[content_column].dropna():
            paragraphs = content.split('\n\n')
            news_data_paragraphs.extend(paragraphs)

        print(f"Extracted {len(news_data_paragraphs)} paragraphs from CSV file")
        news_df = pd.DataFrame({'content': news_data_paragraphs})

        merged_data = pd.concat([pdf_df, news_df], ignore_index=True)
        print(f"Merged data shape: {merged_data.shape}")

        return merged_data

    except Exception as e:
        print(f"Error in extract_and_merge: {str(e)}")
        print(f"Current working directory: {os.getcwd()}")
        print(f"Contents of current directory: {os.listdir('.')}")
        if os.path.exists(pdf_path):
            print(f"Contents of PDF directory: {os.listdir(pdf_path)}")
        raise
```

In [22]:
```
# Assign Sentiment Analyzer Score
sid = SentimentIntensityAnalyzer()

def assign_sentiment_scores(text):
    scores = sid.polarity_scores(text)
    return scores['compound']

def assign_scores(data):
    data['sentiment'] = data['content'].apply(assign_sentiment_scores)
    return data

# Function to assign direction and new_direction based on sentiment scores
def assign_directions(data):
    data['direction'] = data['sentiment'].apply(lambda x: 'bearish' if x < 0.0 else ('
    data['new_direction'] = data['sentiment'].apply(lambda x: 2 if x < 0.0 else (1 if
    return data

# Function to preprocess individual text
def preprocess_text(text):
    lemmatizer = WordNetLemmatizer()
    stop_words = set(stopwords.words('english'))

    # Lowercase the text
    text = text.lower()

    # Remove emojis
    text = emoji.replace_emoji(text, '')

    # Remove emoticons (this is a basic implementation, might need refinement)
    text = re.sub(r'[:;=]-?[()DPp]', '', text)

    # Remove punctuation and numbers
    text = re.sub(r'[^\w\s]', '', text)
    text = re.sub(r'\d+', '', text)

    # Remove extra spaces
    text = re.sub(r'\s+', ' ', text).strip()

    # Tokenize
    tokens = word_tokenize(text)

    # Remove stop words and lemmatize
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words]

    try:
        tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_wo
    except LookupError:
        # If lemmatization fails, just use the original tokens
        tokens = [word for word in tokens if word not in stop_words]

    return ' '.join(tokens)

# Function to preprocess the entire DataFrame
def preprocess_data(df):
    df_cleaned = df.copy()
    df_cleaned['content'] = df_cleaned['content'].apply(preprocess_text)
    return df_cleaned

# Count the number of bearish, bullish, and neutral sentiments
```

```
def sentiment_counts(data):
    return data['direction'].value_counts()
```

In [23]:
```
# Prepare Dataset Function
def prepare_dataset(data, sample_frac=0.1, random_state=42):
    print("Preparing dataset...")
    data = data.sample(frac=sample_frac, random_state=random_state).reset_index(drop=T

    X = data['content']
    y = data['new_direction']

    # TF-IDF Vectorization
    vectorizer = TfidfVectorizer(max_features=5000)
    X = vectorizer.fit_transform(X)

    # Split the data
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_st

    # Define resampling strategy
    over = SMOTE(sampling_strategy='auto', random_state=random_state)
    under = RandomUnderSampler(sampling_strategy='auto', random_state=random_state)

    # Create a pipeline with SMOTE and RandomUnderSampler
    resampling = Pipeline([('over', over), ('under', under)])

    # Apply resampling
    X_train_resampled, y_train_resampled = resampling.fit_resample(X_train, y_train)

    print(f"Dataset prepared with train size: {X_train_resampled.shape[0]} and test si
    return X_train_resampled, X_test, y_train_resampled, y_test, vectorizer
```

In [24]:
```
import sklearn
def train_and_evaluate(X_train, X_test, y_train, y_test):
    print("Training and evaluating model...")

    try:
        # Initialize AdaBoost classifier
        base_estimator = DecisionTreeClassifier(max_depth=3)

        # Check scikit-learn version and use appropriate parameter
        if sklearn.__version__ >= '0.22':
            model = AdaBoostClassifier(estimator=base_estimator, n_estimators=50, ranc
        else:
            model = AdaBoostClassifier(base_estimator=base_estimator, n_estimators=50,

        # Train the model
        model.fit(X_train, y_train)

        # Make predictions
        y_pred = model.predict(X_test)

        # Calculate accuracy
        accuracy = accuracy_score(y_test, y_pred)

        # Generate classification report
        report = classification_report(y_test, y_pred, target_names=['bullish', 'neutr
        report_df = pd.DataFrame(report).transpose()

        # Generate confusion matrix
        cm = confusion_matrix(y_test, y_pred)

        return model, {'accuracy': accuracy}, report_df, y_test, y_pred, cm

    except Exception as e:
        print(f"An error occurred in train_and_evaluate: {str(e)}")
        print(f"Error details: {traceback.format_exc()}")
        return None
```

In [25]:
```
def create_comprehensive_report(company_name, metrics, report_df, cm):
    # Create confusion matrix DataFrame
    cm_df = pd.DataFrame(cm, index=['True Bullish', 'True Neutral', 'True Bearish'],
                         columns=['Pred Bullish', 'Pred Neutral', 'Pred Bearish'])

    # Prepare data for the comprehensive report
    report_data = {
        'Company': company_name,
        'Accuracy': metrics['accuracy'],
        'Confusion Matrix': cm_df.to_json(),
    }

    # Add precision, recall, and F1-score for each class
    for class_label, class_name in zip(['bullish', 'neutral', 'bearish'], ['Bullish',
        if class_label in report_df.index:
            report_data.update({
                f'Precision ({class_name})': report_df.loc[class_label, 'precision'],
                f'Recall ({class_name})': report_df.loc[class_label, 'recall'],
                f'F1-Score ({class_name})': report_df.loc[class_label, 'f1-score'],
            })
        else:
            report_data.update({
                f'Precision ({class_name})': None,
                f'Recall ({class_name})': None,
                f'F1-Score ({class_name})': None,
            })

    return pd.DataFrame([report_data])
```

In [26]:
```
def main(company_name, pdf_path, csv_path):
    try:
        logger.info(f"Processing {company_name}...")

        # Load and preprocess data
        raw_data = extract_and_merge(pdf_path, csv_path)
        data_with_sentiment = assign_scores(raw_data)
        data_with_directions = assign_directions(data_with_sentiment)
        cleaned_data = preprocess_data(data_with_directions)

        # Display sentiment counts
        counts = sentiment_counts(cleaned_data)
        logger.info(f"{company_name} Sentiment Counts:")
        logger.info(counts)

        # Prepare dataset
        X_train, X_test, y_train, y_test, vectorizer = prepare_dataset(cleaned_data)

        # Train and evaluate
        result = train_and_evaluate(X_train, X_test, y_train, y_test)
        if result is None:
            logger.error(f"Training and evaluation failed for {company_name}")
            return None

        model, metrics, report_df, y_test, y_pred, cm = result

        # Display the evaluation metrics
        logger.info(f"Evaluation Metrics for {company_name}:")
        logger.info(f"Accuracy: {metrics['accuracy']}")

        # Display the classification report
        logger.info(f"Classification Report for {company_name}:")
        logger.info(report_df)

        # Create comprehensive report
        comprehensive_report = create_comprehensive_report(company_name, metrics, repo

        return comprehensive_report

    except Exception as e:
        logger.error(f"Error processing {company_name}: {str(e)}")
        logger.error(traceback.format_exc())
        return None
```

In [27]:
```
if __name__ == "__main__":
    # Define paths for each company
    companies = {
        'Lloyds': {
            'pdf_path': 'data/lloyds',
            'csv_path': 'data/lloyds/lloyds_news.csv'
        },
        'IAG': {
            'pdf_path': 'data/iag',
            'csv_path': 'data/iag/iag_news.csv'
        },
        'Vodafone': {
            'pdf_path': 'data/vodafone',
            'csv_path': 'data/vodafone/vodafone_news.csv'
        }
    }

    all_reports = []

    for company_name, paths in companies.items():
        try:
            logger.info(f"Starting processing for {company_name}")
            company_report = main(company_name, paths['pdf_path'], paths['csv_path'])

            if company_report is not None:
                all_reports.append(company_report)

        except Exception as e:
            logger.error(f"Failed to process {company_name}: {str(e)}")

    # Combine all reports into a single DataFrame
    if all_reports:
        combined_report = pd.concat(all_reports, ignore_index=True)
        combined_report.to_csv('comprehensive_classification_report_adaboost.csv', inc
        logger.info("Comprehensive classification report for all companies saved to CS
    else:
        logger.warning("No reports were generated.")
```

```
INFO:__main__:Starting processing for Lloyds
INFO:__main__:Processing Lloyds...
Attempting to read PDF files from: data/lloyds
Found 20 PDF files
Extracted 66875 sentences from PDF files
Attempting to read CSV file: data/lloyds/lloyds_news.csv
Extracted 1834 paragraphs from CSV file
Merged data shape: (68709, 1)
INFO:__main__:Lloyds Sentiment Counts:
INFO:__main__:direction
neutral    27490
bullish    26909
bearish    14310
Name: count, dtype: int64
Preparing dataset...
Dataset prepared with train size: 6618 and test size: 1375
Training and evaluating model...
C:\Users\ELITEBOOK\anaconda3\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:5
27: FutureWarning: The SAMME.R algorithm (the default) is deprecated and will be remo
ved in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
INFO:__main__:Evaluation Metrics for Lloyds:
INFO:__main__:Accuracy: 0.7258181818181818
INFO:__main__:Classification Report for Lloyds:
INFO:__main__:              precision    recall  f1-score      support
bullish        0.817797  0.706960  0.758350   546.000000
neutral        0.658940  0.750943  0.701940   530.000000
bearish        0.715719  0.715719  0.715719   299.000000
accuracy       0.725818  0.725818  0.725818     0.725818
macro avg      0.730819  0.724541  0.725336  1375.000000
weighted avg   0.734368  0.725818  0.727336  1375.000000
INFO:__main__:Starting processing for IAG
INFO:__main__:Processing IAG...
Attempting to read PDF files from: data/iag
Found 11 PDF files
Extracted 34291 sentences from PDF files
Attempting to read CSV file: data/iag/iag_news.csv
Extracted 2037 paragraphs from CSV file
Merged data shape: (36328, 1)
INFO:__main__:IAG Sentiment Counts:
INFO:__main__:direction
neutral    17607
bullish    12229
bearish     6492
Name: count, dtype: int64
C:\Users\ELITEBOOK\anaconda3\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:5
27: FutureWarning: The SAMME.R algorithm (the default) is deprecated and will be remo
ved in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
Preparing dataset...
Dataset prepared with train size: 4182 and test size: 727
Training and evaluating model...
```

```
INFO:__main__:Evaluation Metrics for IAG:
INFO:__main__:Accuracy: 0.7042640990371389
INFO:__main__:Classification Report for IAG:
INFO:__main__:              precision    recall  f1-score     support
bullish        0.740196  0.592157  0.657952  255.000000
neutral        0.687943  0.841040  0.756827  346.000000
bearish        0.700000  0.555556  0.619469  126.000000
accuracy       0.704264  0.704264  0.704264    0.704264
macro avg      0.709380  0.662918  0.678083  727.000000
weighted avg   0.708361  0.704264  0.698340  727.000000
INFO:__main__:Starting processing for Vodafone
INFO:__main__:Processing Vodafone...
Attempting to read PDF files from: data/vodafone
Found 14 PDF files
Extracted 51164 sentences from PDF files
Attempting to read CSV file: data/vodafone/vodafone_news.csv
Extracted 0 paragraphs from CSV file
Merged data shape: (51164, 1)
INFO:__main__:Vodafone Sentiment Counts:
INFO:__main__:direction
neutral    24998
bullish    18868
bearish     7298
Name: count, dtype: int64
Preparing dataset...
Dataset prepared with train size: 5721 and test size: 1024
Training and evaluating model...
C:\Users\ELITEBOOK\anaconda3\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:5
27: FutureWarning: The SAMME.R algorithm (the default) is deprecated and will be remo
ved in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
INFO:__main__:Evaluation Metrics for Vodafone:
INFO:__main__:Accuracy: 0.7734375
INFO:__main__:Classification Report for Vodafone:
INFO:__main__:              precision    recall  f1-score      support
bullish        0.811209  0.747283  0.777935   368.000000
neutral        0.781885  0.826172  0.803419   512.000000
bearish        0.652778  0.652778  0.652778   144.000000
accuracy       0.773438  0.773438  0.773438     0.773438
macro avg      0.748624  0.742077  0.744711  1024.000000
weighted avg   0.774268  0.773438  0.773077  1024.000000
INFO:__main__:Comprehensive classification report for all companies saved to CSV.
```