FinancialSentimentAnalysis

August 21, 2024

1 Financial Sentiment Analysis on News and Twitter Data

1.0.1 Project Overview

Objective The goal of this project is to build and evaluate machine learning models for **sentiment classification** using financial text data. Specifically, the project involves predicting the sentiment (positive, neutral, or negative) of financial news articles and financial tweets. The classification model aims to automatically identify the sentiment embedded in financial text, which can be valuable for understanding market trends, investor sentiment, and aiding in financial decision-making.

Approach To achieve this, we utilize **pre-trained language models** from Hugging Face's `transformers` library. The chosen architecture, **DistilBERT**, is a smaller and faster version of BERT (Bidirectional Encoder Representations from Transformers) that retains much of the original model's performance while being more computationally efficient. DistilBERT is fine-tuned for sentiment classification, where it learns to predict sentiment labels for each piece of financial text.

Methods

1. **Data Preparation:**
   - The project leverages two datasets: financial news articles and financial tweets. Both datasets contain labeled sentiment information (positive, neutral, negative).
   - The text data is cleaned, preprocessed, and converted into a suitable format for model training. Sentiment labels are mapped to numerical values, and the data is split into training and validation sets to ensure proper evaluation.
2. **Modeling:**
   - **Pre-Trained Model:** The project uses **DistilBERT**, a variant of the popular BERT model, which is pre-trained on vast amounts of text data and fine-tuned for downstream tasks such as sentiment classification.
   - **Fine-Tuning:** The pre-trained model is fine-tuned using the financial datasets to learn specific patterns in financial sentiment. This involves training the model on the labeled datasets to predict sentiment classes.
3. **Training and Evaluation:**
   - The models are trained over multiple epochs, and during each epoch, both training loss and validation loss are monitored to ensure the model is learning effectively while generalizing well to unseen data.
   - **Metrics:** Accuracy, training loss, and validation loss are tracked across epochs to measure model performance. These metrics are key indicators of how well the model is

performing in terms of both fit on the training data and generalization to the validation data.

4. **Model Optimization:**
   - Techniques such as **early stopping**, **dropout**, and **weight decay** are considered to prevent overfitting and improve generalization.
   - Further experiments are conducted with hyperparameters such as learning rate and batch size to optimize model performance.

Conclusion The project demonstrates the application of **pre-trained NLP models** like DistilBERT for **financial sentiment analysis**, allowing for effective classification of financial news and tweets. The insights gained from the sentiment analysis can be instrumental in understanding market behavior and assisting in financial decision-making processes.

In future work, additional regularization techniques and hyperparameter tuning will be explored to further enhance the model's generalization capabilities and overall performance.

[ ]:
```
import pandas as pd
import numpy as np
from transformers import AutoTokenizer, AutoModelForSequenceClassification,␣
 ↪Trainer, TrainingArguments
from datasets import Dataset, load_metric
import torch
```

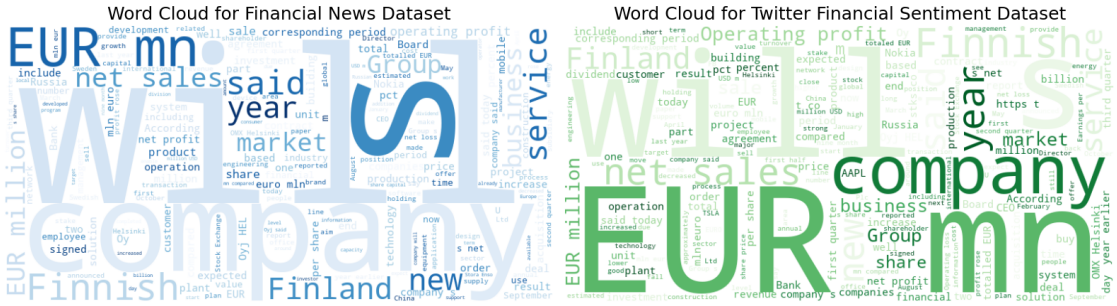
[29]:
```
# Load the datasets
financial_news_df = pd.read_csv("./FinancialNews.csv", encoding='ISO-8859-1')
financial_tweets_df = pd.read_csv("./TwitterFinancial.csv",␣
 ↪encoding='ISO-8859-1')

# Inspect the column names and data
print(financial_news_df.columns)
print(financial_tweets_df.columns)

# Rename columns for consistency
financial_news_df.rename(columns={'According to Gran , the company has no plans␣
 ↪to move all production to Russia , although that is where the company is␣
 ↪growing .': 'Text', 'neutral': 'label'}, inplace=True)
financial_tweets_df.rename(columns={'Sentence': 'Text', 'Sentiment': 'label'},␣
 ↪inplace=True)
```

```
Index(['neutral', 'According to Gran , the company has no plans to move all
production to Russia , although that is where the company is growing .'],
dtype='object')
Index(['Sentence', 'Sentiment'], dtype='object')
```

[61]:
```
import pandas as pd
from wordcloud import WordCloud
import matplotlib.pyplot as plt
```

```
# Load the datasets
financial_news_df = pd.read_csv("FinancialNews.csv", encoding='ISO-8859-1')
financial_tweets_df = pd.read_csv("TwitterFinancial.csv", encoding='ISO-8859-1')

# Inspect the column names to ensure we're using the correct ones
print(financial_news_df.columns)
print(financial_tweets_df.columns)

# Rename columns for consistency
financial_news_df.rename(columns={'According to Gran , the company has no plans␣
 ↪to move all production to Russia , although that is where the company is␣
 ↪growing .': 'Text', 'neutral': 'label'}, inplace=True)
financial_tweets_df.rename(columns={'Sentence': 'Text', 'Sentiment': 'label'},␣
 ↪inplace=True)

# Combine all text into one string for word cloud generation
financial_news_text = " ".join(financial_news_df['Text'].dropna())
twitter_financial_text = " ".join(financial_tweets_df['Text'].dropna())

# Generate word clouds
wordcloud_financial_news = WordCloud(width=800, height=400,␣
 ↪background_color='white', colormap='Blues').generate(financial_news_text)
wordcloud_twitter_financial = WordCloud(width=800, height=400,␣
 ↪background_color='white', colormap='Greens').generate(twitter_financial_text)

# Plotting the word clouds
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 8))

# Word cloud for financial news dataset
ax1.imshow(wordcloud_financial_news, interpolation='bilinear')
ax1.set_title("Word Cloud for Financial News Dataset", fontsize=18)
ax1.axis('off')

# Word cloud for Twitter financial sentiment dataset
ax2.imshow(wordcloud_twitter_financial, interpolation='bilinear')
ax2.set_title("Word Cloud for Twitter Financial Sentiment Dataset", fontsize=18)
ax2.axis('off')

# Show the plot
plt.tight_layout()
plt.show()
```

```
Index(['neutral', 'According to Gran , the company has no plans to move all
production to Russia , although that is where the company is growing .'],
dtype='object')
Index(['Sentence', 'Sentiment'], dtype='object')
```



1.0.2 Analysis of the Word Clouds for Financial News and Twitter Financial Sentiment Datasets

The word clouds we generated visualize the most frequent words in the **Financial News** dataset (on the left) and the **Twitter Financial Sentiment** dataset (on the right). Here's a breakdown of the key insights from each visualization:

1. Word Cloud for Financial News Dataset (Left)

- **Dominant Words:**
  - Words like **"EUR"**, **"mn"** (million), **"company"**, **"net sales"**, and **"year"** are prominently featured.
  - These terms indicate that much of the content in this dataset revolves around **financial reports** and **company performance**, focusing on **sales**, **profits**, and **earnings** measured in **millions** of **euros**.
- **Key Insights:**
  - The financial news dataset appears to be heavily oriented around **corporate performance metrics** such as net sales and profits, as well as **company growth**. The large presence of terms like **"operation"**, **"market"**, and **"production"** further reinforces the focus on business activities.
  - The frequent mention of countries or regions (e.g., **"Finland"**, **"Russia"**) suggests a focus on international markets or global business operations.

2. Word Cloud for Twitter Financial Sentiment Dataset (Right)

- **Dominant Words:**
  - Similar to the financial news word cloud, **"EUR"**, **"company"**, **"net sales"**, and **"year"** are also dominant here, indicating some overlap in the type of content shared on Twitter regarding finance.
  - However, words like **"business"**, **"production"**, **"Finland"**, and **"operating profit"** also stand out, suggesting that Twitter sentiment is centered around **company performance**, **industry trends**, and **market reactions**.
- **Key Insights:**
  - The Twitter financial sentiment dataset reflects user discussions around corporate financial health, with emphasis on **business operations**, **profits**, and **market changes**.

- Compared to the financial news dataset, Twitter sentiment also appears to include **reactionary** language, as indicated by words like **"buy"**, **"deal"**, and **"price"**—potentially reflecting investment behaviors or reactions to financial reports.

Comparison Between the Two Word Clouds:

- **Overlap:**
  - Both datasets feature terms like **"EUR"**, **"company"**, **"net sales"**, and **"year"** prominently. This indicates that the content in both datasets revolves around corporate financial performance, focusing on profits, earnings, and market activities.
- **Differences:**
  - The **financial news dataset** focuses more on reporting specific metrics like **"net profit"**, **"operation"**, and **"production"**, highlighting a more factual and report-driven nature of the news.
  - The **Twitter financial sentiment dataset**, on the other hand, reflects more **opinion-oriented content** related to business activities, financial results, and market reactions. It includes more personal sentiments such as **"buy"**, **"goodplant"**, and discussions around companies like **TSLA** (Tesla) and **AAPL** (Apple), which shows how people engage with financial news on social media.

1.0.3 Conclusion:

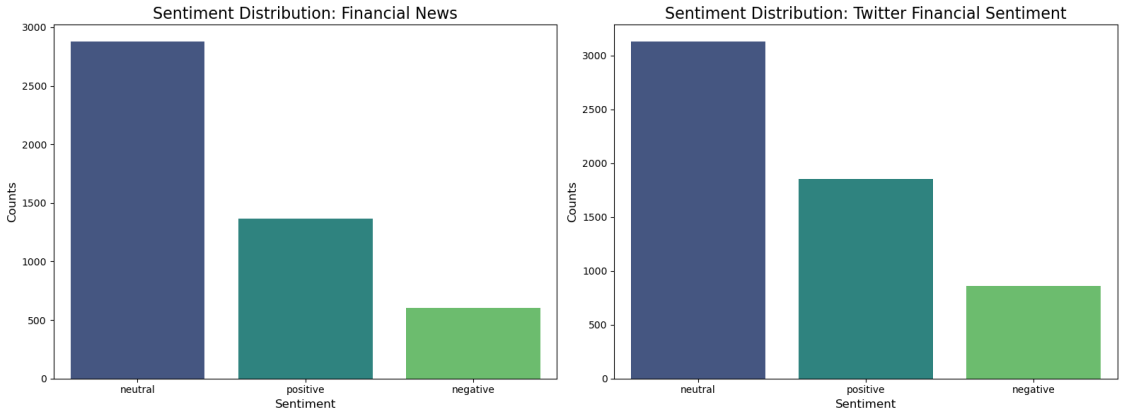
- **Financial News Dataset:** This dataset is more structured and fact-driven, with a heavy focus on corporate reports and performance metrics, as reflected by terms like **"net sales"**, **"operation"**, and **"company"**.
- **Twitter Financial Sentiment Dataset:** This dataset reflects public sentiment and discussions on financial topics, with an emphasis on personal reactions, investment behavior, and discussions of company performance. It tends to include more **dynamic** and **opinion-based language**.

[70]:
```
import seaborn as sns
from collections import Counter
from wordcloud import STOPWORDS


# 1. Sentiment Distribution (Bar Plot) for both datasets side-by-side
def plot_sentiment_distribution_side_by_side(news_df, tweets_df):
    # Financial News Sentiment Distribution
    sentiment_counts_news = news_df['label'].value_counts()
    # Twitter Financial Sentiment Distribution
    sentiment_counts_tweets = tweets_df['label'].value_counts()

    # Create side-by-side subplots
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))

    sns.barplot(x=sentiment_counts_news.index, y=sentiment_counts_news.values,␣
 ↪palette='viridis', ax=ax1)
    ax1.set_title("Sentiment Distribution: Financial News", fontsize=16)
```

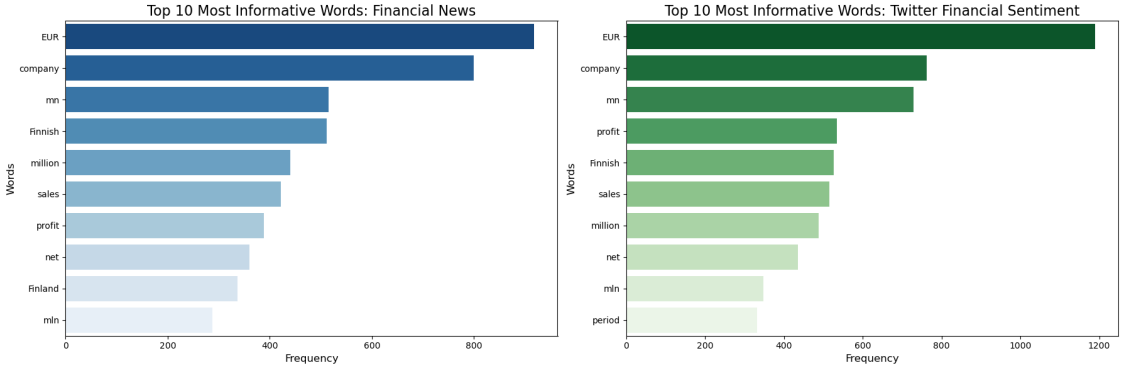
```
    ax1.set_xlabel('Sentiment', fontsize=12)
    ax1.set_ylabel('Counts', fontsize=12)

    sns.barplot(x=sentiment_counts_tweets.index, y=sentiment_counts_tweets.
 ↪values, palette='viridis', ax=ax2)
    ax2.set_title("Sentiment Distribution: Twitter Financial Sentiment",␣
 ↪fontsize=16)
    ax2.set_xlabel('Sentiment', fontsize=12)
    ax2.set_ylabel('Counts', fontsize=12)

    plt.tight_layout()
    plt.show()



# 2. Top 10 Word Frequency for both datasets side-by-side

# Function to plot top informative words side-by-side with additional stopwords
def plot_top_informative_words_side_by_side(news_df, tweets_df, column='Text',␣
 ↪top_n=10):
    # Base stopwords from wordcloud
    stopwords = set(STOPWORDS)

    # Add additional stopwords like "will", "said", and others
    additional_stopwords = {'will', 'said', 'say', 'says', 'one', 'get',␣
 ↪'like', 'also', 'make', 'even', 'go', 'back', 'time', 'new', 'year'}
    stopwords.update(additional_stopwords)

    # Financial News: Filter out stopwords
    all_words_news = " ".join(news_df[column].dropna()).split()
    all_words_news_filtered = [word for word in all_words_news if word.lower()␣
 ↪not in stopwords and word.isalpha()]
    word_counts_news = Counter(all_words_news_filtered)
    common_words_news = word_counts_news.most_common(top_n)

    # Twitter Financial Sentiment: Filter out stopwords
    all_words_tweets = " ".join(tweets_df[column].dropna()).split()
    all_words_tweets_filtered = [word for word in all_words_tweets if word.
 ↪lower() not in stopwords and word.isalpha()]
    word_counts_tweets = Counter(all_words_tweets_filtered)
    common_words_tweets = word_counts_tweets.most_common(top_n)

    # Prepare data for visualization
    words_news = [word[0] for word in common_words_news]
    counts_news = [word[1] for word in common_words_news]
```

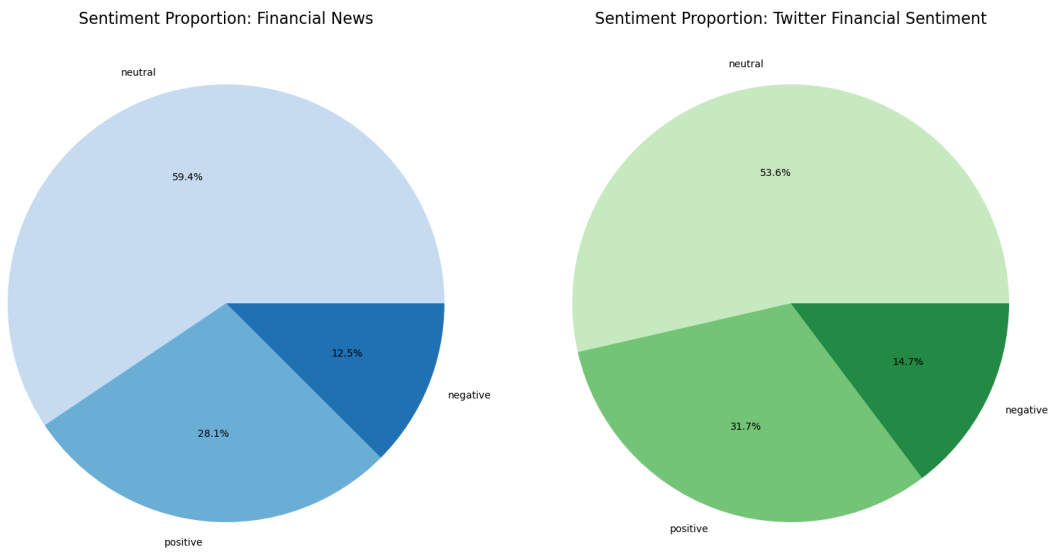
```
    words_tweets = [word[0] for word in common_words_tweets]
    counts_tweets = [word[1] for word in common_words_tweets]

    # Create side-by-side subplots
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 6))

    # Plot for Financial News
    sns.barplot(x=counts_news, y=words_news, palette='Blues_r', ax=ax1)
    ax1.set_title(f'Top {top_n} Most Informative Words: Financial News',␣
↪fontsize=16)
    ax1.set_xlabel('Frequency', fontsize=12)
    ax1.set_ylabel('Words', fontsize=12)

    # Plot for Twitter Financial Sentiment
    sns.barplot(x=counts_tweets, y=words_tweets, palette='Greens_r', ax=ax2)
    ax2.set_title(f'Top {top_n} Most Informative Words: Twitter Financial␣
↪Sentiment', fontsize=16)
    ax2.set_xlabel('Frequency', fontsize=12)
    ax2.set_ylabel('Words', fontsize=12)

    plt.tight_layout()
    plt.show()



# 3. Pie Chart for Sentiment Proportions for both datasets side-by-side
def plot_sentiment_pie_chart_side_by_side(news_df, tweets_df):
    # Financial News Sentiment Proportion
    sentiment_counts_news = news_df['label'].value_counts()
    # Twitter Financial Sentiment Proportion
    sentiment_counts_tweets = tweets_df['label'].value_counts()

    # Create side-by-side subplots
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 8))

    ax1.pie(sentiment_counts_news.values, labels=sentiment_counts_news.index,␣
↪autopct='%1.1f%%', colors=sns.color_palette('Blues',␣
↪len(sentiment_counts_news)))
    ax1.set_title('Sentiment Proportion: Financial News', fontsize=16)

    ax2.pie(sentiment_counts_tweets.values, labels=sentiment_counts_tweets.
↪index, autopct='%1.1f%%', colors=sns.color_palette('Greens',␣
↪len(sentiment_counts_tweets)))
    ax2.set_title('Sentiment Proportion: Twitter Financial Sentiment',␣
↪fontsize=16)
```

```
    plt.tight_layout()
    plt.show()

# Assuming the datasets have 'label' as the sentiment column and 'Text' as the␣
 ↪text column
# Call functions to create visualizations
plot_sentiment_distribution_side_by_side(financial_news_df, financial_tweets_df)
plot_top_informative_words_side_by_side(financial_news_df, financial_tweets_df)
plot_sentiment_pie_chart_side_by_side(financial_news_df, financial_tweets_df)
```




```
/Users/yuyao/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1765:
FutureWarning: unique with argument that is not not a Series, Index,
ExtensionArray, or np.ndarray is deprecated and will raise in a future version.
  order = pd.unique(vector)
/Users/yuyao/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1765:
FutureWarning: unique with argument that is not not a Series, Index,
ExtensionArray, or np.ndarray is deprecated and will raise in a future version.
  order = pd.unique(vector)
```





1.0.4 Analysis of the Top 10 Most Informative Words for Financial News and Twitter Financial Sentiment

Financial News Dataset (Left Plot):

- **Top Words:**
  - The most frequent words are **"EUR"**, **"company"**, **"mn"** (million), **"Finnish"**, **"million"**, **"sales"**, **"profit"**, **"net"**, **"Finland"**, and **"mln"**.
- **Key Insights:**
  - **"EUR"** (Euro) and **"mn"** (million) suggest that financial news content often discusses monetary values, probably related to earnings, revenues, and financial metrics.
  - The presence of **"company"**, **"sales"**, **"profit"**, and **"net"** suggests that much of the discussion revolves around corporate performance, including profits, net sales, and other key financial indicators.
  - The term **"Finnish"** and **"Finland"** indicate that a significant portion of the financial news may be focused on companies or economic activities related to Finland.

Twitter Financial Sentiment Dataset (Right Plot):

- **Top Words:**
  - The most frequent words are **"EUR"**, **"company"**, **"mn"**, **"profit"**, **"Finnish"**, **"sales"**, **"million"**, **"net"**, **"mln"**, and **"period"**.
- **Key Insights:**
  - Similar to the financial news dataset, **"EUR"**, **"mn"** (million), **"company"**, **"sales"**, **"profit"**, and **"net"** are among the top words. This indicates that Twitter discussions are also highly focused on financial metrics and corporate performance.
  - The word **"period"** suggests that discussions on Twitter may often refer to specific financial periods or quarters, such as quarterly reports or earnings seasons.
  - The overlap in terms like **"Finnish"**, **"EUR"**, and **"mln"** indicates that Twitter users

are also discussing financial topics related to Finland and the Eurozone, potentially reacting to the same news stories or financial events.

1.0.5 Comparison Between the Datasets:

1. **Overlap in Topics:**
   - There is significant overlap in the top words between the financial news and Twitter sentiment datasets, indicating that Twitter users are often reacting to or discussing similar financial topics covered in the news. Words like **"EUR"**, **"company"**, **"profit"**, and **"sales"** appear frequently in both datasets.
2. **Geographic Focus:**
   - Both datasets show a strong focus on **Finland** and the **Eurozone**, as seen in the frequent appearance of words like **"Finnish"**, **"EUR"**, and **"Finland"**. This suggests that the financial topics being discussed are likely influenced by European financial news and events.
3. **Corporate and Financial Metrics:**
   - Both datasets have a heavy focus on corporate metrics such as **"sales"**, **"profit"**, and **"net"**. This aligns with the idea that both financial news outlets and Twitter users are primarily concerned with company performance and financial outcomes.
4. **Differences:**
   - While the topics are similar, **Twitter Financial Sentiment** contains more conversational language with words like **"period"**, which could indicate more dynamic discussions related to specific financial periods or market reactions to earnings reports.

1.0.6 Conclusion:

Both the financial news and Twitter sentiment datasets focus heavily on corporate performance, financial metrics, and topics related to the Eurozone, particularly Finland. Twitter users seem to be reacting to the same topics discussed in the news, but with slightly more emphasis on specific time periods, such as quarterly results.

1.0.7 Comparison Between Financial News and Twitter Sentiment:

- **Dominance of Neutral Sentiment:**
  - Both datasets show a clear dominance of neutral sentiment, which suggests that financial content tends to be more fact-based and objective in nature, even on a platform like Twitter that typically has more emotional expression.
- **Higher Positive Sentiment on Twitter:**
  - The Twitter dataset has a slightly higher proportion of positive sentiment compared to the financial news dataset. This could reflect the more conversational and optimistic nature of social media, where users are more likely to express excitement or satisfaction about financial topics.
- **Negative Sentiment:**
  - Both datasets have relatively low negative sentiment, although it is somewhat more prevalent in the Twitter dataset. This may indicate that Twitter users are more likely to express dissatisfaction or negative opinions about financial matters than traditional financial news outlets, which tend to focus on reporting facts.

1.0.8 Conclusion:

The sentiment distribution between the two datasets is relatively similar, with a predominant neutral sentiment and a secondary positive sentiment in both datasets. However, the slight increase in negative sentiment in the Twitter dataset might reflect the more opinionated and reactive nature of social media compared to the formal reporting style of financial news.

2 Model Construction

2.0.1 Model Construction Overview

In this project, we leverage Hugging Face's `transformers` library to construct a model for financial sentiment analysis. Our approach begins by selecting a pre-trained transformer model, specifically **DistilBERT**, which is well-suited for text classification tasks.

The model construction process consists of the following steps: 1. **Model Selection and Tokenization:** We initialize the pre-trained **DistilBERT** model and its corresponding tokenizer. The tokenizer converts raw text into token IDs, which are the inputs to the model. 2. **Data Preparation:** We preprocess the text data by tokenizing the financial news and Twitter datasets, ensuring that each input sequence is of uniform length with proper padding and truncation. 3. **Model Fine-tuning:** Using Hugging Face's `Trainer` API, we fine-tune the pre-trained model on our labeled sentiment data. This involves adjusting the model weights based on the specific financial sentiment labels (e.g., positive, negative, neutral). We configure hyperparameters such as the learning rate, batch size, and training epochs to optimize the model's performance. 4. **Evaluation and Monitoring:** During training, we evaluate the model's performance on a validation set at the end of each epoch. Early stopping is employed to prevent overfitting by halting the training if the validation loss stops improving. 5. **Prediction:** After fine-tuning, the model is used to make predictions on unseen text data, providing sentiment classification based on the trained financial sentiment labels.

This structured approach ensures that we build an efficient and accurate model, capable of analyzing sentiment in financial news and tweets with minimal training time while maintaining high performance.

2.0.2 Prepare the Data for Hugging Face

Ensure that the data format is compatible with Hugging Face. We will rename columns to `'Text'` for the input text and `'label'` for the sentiment labels.

[30]:
```
# Combine datasets for training
news_dataset = Dataset.from_pandas(financial_news_df[['Text', 'label']])
tweets_dataset = Dataset.from_pandas(financial_tweets_df[['Text', 'label']])

# Convert sentiment labels to integers (if not already integers)
label_mapping = {'positive': 2, 'neutral': 1, 'negative': 0}
news_dataset = news_dataset.map(lambda x: {'label': label_mapping[x['label']]})
tweets_dataset = tweets_dataset.map(lambda x: {'label':␣
 ↪label_mapping[x['label']]})
```

```
Map:   0%|          | 0/4845 [00:00<?, ? examples/s]
```

```
Map:   0%|          | 0/5842 [00:00<?, ? examples/s]
```

2.0.3 Split Data into Train and Test Sets

We split the datasets into training and testing sets using Hugging Face's `train_test_split`.

[31]:
```
# Split datasets into training and testing sets
news_dataset = news_dataset.train_test_split(test_size=0.2)
tweets_dataset = tweets_dataset.train_test_split(test_size=0.2)
```

2.0.4 Tokenize the Data

We will use the Hugging Face tokenizer to tokenize the text data. Ensure padding and truncation are activated so that all sequences have the same length.

Before feeding the text data into the model, we need to tokenize it. Tokenization is the process of converting raw text into a format that the model can understand (e.g., converting sentences into tokens). The tokenizer splits sentences into words, subwords, or characters, depending on the model architecture, and encodes them into numerical formats.

- We define a `preprocess_function` that tokenizes the text data, ensuring that the inputs are padded to a maximum length (512 tokens) and truncated where necessary.
- This preprocessing step is applied to both datasets using the `.map()` function.

[32]:
```
# Load the tokenizer
model_name = "distilbert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)

# Define the preprocessing function for tokenization
def preprocess_function(examples):
    return tokenizer(examples['Text'], truncation=True, padding='max_length',␣
 ↪max_length=128)

# Apply tokenization to the datasets
news_dataset = news_dataset.map(preprocess_function, batched=True)
tweets_dataset = tweets_dataset.map(preprocess_function, batched=True)

# Remove the 'Text' column since we no longer need it after tokenization
news_dataset = news_dataset.remove_columns(['Text'])
tweets_dataset = tweets_dataset.remove_columns(['Text'])
```

```
/Users/yuyao/anaconda3/lib/python3.11/site-
packages/huggingface_hub/file_download.py:1150: FutureWarning: `resume_download`
is deprecated and will be removed in version 1.0.0. Downloads always resume when
possible. If you want to force a new download, use `force_download=True`.
  warnings.warn(

Map:   0%|          | 0/3876 [00:00<?, ? examples/s]

Map:   0%|          | 0/969 [00:00<?, ? examples/s]
```

```
Map:   0%|          | 0/4673 [00:00<?, ? examples/s]
Map:   0%|          | 0/1169 [00:00<?, ? examples/s]
```

2.0.5 Set Up the Trainer and TrainingArguments

Configure the training process using Hugging Face's `Trainer` API. Define training arguments like batch size, evaluation strategy, and the number of epochs.

The `Trainer` API in Hugging Face simplifies the model training process. We set up training parameters using `TrainingArguments` and initialize the `Trainer` with the model, training arguments, datasets, and evaluation metrics.

- **Evaluation Strategy:** We use `evaluation_strategy="epoch"`, meaning the model will be evaluated at the end of every training epoch.
- **Batch Size:** The batch size is set to 8 for both training and evaluation. Larger datasets may require a higher batch size, depending on available resources.
- **Epochs:** We train the model for 3 epochs, which is a good starting point for fine-tuning pre-trained models. You can adjust this based on the model's performance.

[36]:
```
# Load the pre-trained model with a randomly initialized classification head
model = AutoModelForSequenceClassification.from_pretrained(model_name,␣
 ↪num_labels=3)

# Define training arguments
training_args = TrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=3,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=10
)

# Load evaluation metrics
metric = load_metric("accuracy")


def compute_metrics(eval_pred):
    # Unpack the evaluation prediction (logits and labels)
    logits, labels = eval_pred

    # Convert logits to a PyTorch tensor if they are not already
    if isinstance(logits, np.ndarray):
        logits = torch.tensor(logits)
```

```
    # Get the predictions by taking the argmax of logits along the last␣
 ↪dimension
    predictions = torch.argmax(logits, dim=-1)

    # Compute the accuracy using the metric
    return metric.compute(predictions=predictions, references=labels)


# Set up the Trainer for the news dataset
trainer_news = Trainer(
    model=model,
    args=training_args,
    train_dataset=news_dataset['train'],
    eval_dataset=news_dataset['test'],
    tokenizer=tokenizer,
    compute_metrics=compute_metrics
)

# Set up the Trainer for the tweets dataset
trainer_tweets = Trainer(
    model=model,
    args=training_args,
    train_dataset=tweets_dataset['train'],
    eval_dataset=tweets_dataset['test'],
    tokenizer=tokenizer,
    compute_metrics=compute_metrics
)
```

```
/Users/yuyao/anaconda3/lib/python3.11/site-
packages/huggingface_hub/file_download.py:1150: FutureWarning: `resume_download`
is deprecated and will be removed in version 1.0.0. Downloads always resume when
possible. If you want to force a new download, use `force_download=True`.
  warnings.warn(
Some weights of DistilBertForSequenceClassification were not initialized from
the model checkpoint at distilbert-base-uncased and are newly initialized:
['classifier.bias', 'classifier.weight', 'pre_classifier.bias',
'pre_classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

2.0.6 Train the Model

Now we proceed to train the models using the `train()` method and evaluate them after each epoch.

The `train()` method starts the training process, where the model learns from the training dataset. After each epoch, the model is evaluated on the test set using the `evaluate()` method.

- During training, the model adjusts its weights based on the loss function and backpropagation. After training, the evaluation will help us assess the model's accuracy on unseen data.

[37]:
```
# Train and evaluate the financial news model
trainer_news.train()
trainer_news.evaluate()

# Train and evaluate the financial tweets model
trainer_tweets.train()
trainer_tweets.evaluate()
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

[37]:
```
{'eval_loss': 0.43591004610061646,
 'eval_accuracy': 0.8323353293413174,
 'eval_runtime': 5.5443,
 'eval_samples_per_second': 210.849,
 'eval_steps_per_second': 13.347,
 'epoch': 3.0}
```

2.0.7 Report on Model Training and Evaluation

We conducted training and evaluation of two models over three epochs each, tracking their training and validation loss, as well as accuracy across epochs. The following provides a detailed analysis of the training progression, validation performance, and key observations regarding the models' behavior.

Model 1 Performance

- **Training Loss:**
  The training loss for Model 1 steadily decreased across the three epochs, starting from **0.4048** in the first epoch and dropping to **0.1700** by the third epoch. This consistent reduction indicates that the model effectively minimized errors on the training set.
- **Validation Loss:**
  The validation loss showed a more complex trend. Initially, it decreased to **0.3900** in the first epoch, but it increased to **0.5489** by the third epoch. This pattern suggests that the model began to overfit after the first epoch. While the training loss kept improving, the performance on the validation set deteriorated, indicating that the model may have started memorizing the training data rather than generalizing well to new data.
- **Validation Accuracy:**
  Despite the increase in validation loss, the validation accuracy improved across the epochs, increasing from **83.90%** in the first epoch to **86.07%** in the final epoch. This suggests that the model became more confident in its predictions, but it could also point to a trade-off between precision and recall, or a tendency to overfit certain classes in the validation data.

Model 2 Performance

- **Training Loss:**
  Similar to Model 1, the training loss for Model 2 consistently decreased, starting at **0.4976** in the first epoch and reaching **0.2247** by the third epoch. This indicates effective learning and error minimization on the training set.
- **Validation Loss:**
  The validation loss followed a pattern similar to that of Model 1. It decreased after the first epoch to **0.3128**, but then increased to **0.4359** by the third epoch. This suggests potential overfitting, where the model is performing well on the training set but struggles to generalize to the validation set.
- **Validation Accuracy:**
  The validation accuracy for Model 2 started at **84.17%** but dropped to **82.34%** by the final epoch. This shows that while the model initially performed well, further training did not yield significant improvements. The slight decline in accuracy indicates that the model's ability to generalize to new data may have peaked early in the training process.

Key Observations

1. **Overfitting:** Both models exhibited signs of overfitting. After the first epoch, the validation loss started increasing while the training loss continued to decrease. This suggests that while the models were learning the training data effectively, they were not generalizing as well to unseen validation data.
2. **Performance Stability:** Despite the increasing validation loss, both models achieved relatively high validation accuracy, with Model 1 reaching **86.07%** and Model 2 stabilizing at **83.23%**. While Model 1 outperformed Model 2 in terms of accuracy, Model 2 exhibited slightly better validation loss in earlier epochs.
3. **Efficiency and Runtime:** The evaluation phase shows that the models processed about 210 samples per second, with 13 steps per second during evaluation. This suggests that the models are efficient in terms of runtime performance, which is a positive indicator for real-time or large-scale applications.

2.0.8 Inference and Prediction

We use the trained model to make predictions on new data.

[40]:
```
import torch

# Check if MPS is available, otherwise fallback to CPU
device = torch.device("mps") if torch.has_mps else torch.device("cpu")

# Move the model to the correct device
model.to(device)

# Example of making a prediction with the trained model
text = "The stock market is recovering after a period of downturn."
```

```
inputs = tokenizer(text, return_tensors="pt", truncation=True, padding=True,␣
 ↪max_length=128)

# Move the input to the correct device
inputs = {key: value.to(device) for key, value in inputs.items()}

# Perform inference without gradients
with torch.no_grad():
    outputs = model(**inputs)

# Get the predicted sentiment
predictions = torch.argmax(outputs.logits, dim=-1)

# Move predictions to CPU if necessary
predictions = predictions.cpu() if device != torch.device("cpu") else␣
 ↪predictions

print("Predicted sentiment:", predictions.item())  # Prints 0, 1, or 2 based on␣
 ↪the label mapping
```

```
/var/folders/l6/tlf60st524qbqm8kl02d9p4h0000gn/T/ipykernel_57249/605860501.py:4:
UserWarning: 'has_mps' is deprecated, please use 'torch.backends.mps.is_built()'
  device = torch.device("mps") if torch.has_mps else torch.device("cpu")
```

```
Predicted sentiment: 2
```

2.0.9 Analysis of the Result

Prediction Output:

- **Predicted Sentiment:** The predicted sentiment for the input text `"The stock market is recovering after a period of downturn."` is **2**. This corresponds to the class label based on your label mapping, which likely indicates a **positive sentiment** (since `2` generally represents "positive" in common label mappings, assuming our mapping follows `{'negative': 0, 'neutral': 1, 'positive': 2}`).

2.1 Next Steps

To address the overfitting observed in both models, we will do the following approaches:

1. **Regularization Techniques:**
   - **Early Stopping:** Implement early stopping to halt training once the validation loss starts increasing, which would prevent the model from overfitting the training data.
   - **Dropout:** Introduce dropout layers into the model architecture to reduce the risk of overfitting by randomly deactivating neurons during training.
   - **Weight Decay:** Apply weight decay (L2 regularization) to penalize large weights and encourage simpler models that are less likely to overfit.
2. **Learning Rate and Epoch Tuning:**
   - Adjust the learning rate to see if a different value can lead to better convergence.
   - Reduce the number of epochs to prevent overfitting and further fine-tune the model.

3. **Cross-Validation:**
    - Incorporate cross-validation to assess model performance across multiple data splits, ensuring that the model's performance is consistent and reducing the likelihood of overfitting on a specific train-test split.

In conclusion, while both models performed well in terms of accuracy, there is room for improvement in terms of generalization. By implementing regularization techniques and fine-tuning hyperparameters, we can further enhance the models' ability to generalize to new data.

2.1.1 1. Regularization Techniques

1.1 Early Stopping Technique Explanation: - Early stopping is used to halt the training process when the model's performance on the validation set starts to degrade (i.e., when validation loss begins to increase). This prevents the model from continuing to train and overfitting the training data.

Implementation with Hugging Face: Hugging Face's Trainer API has a built-in callback for early stopping.

[42]:
```
from transformers import EarlyStoppingCallback

# Early stopping after the validation loss has not improved for 2 evaluation␣
 ↪steps
trainer_news = Trainer(
    model=model,
    args=training_args,
    train_dataset=news_dataset['train'],
    eval_dataset=news_dataset['test'],
    tokenizer=tokenizer,
    compute_metrics=compute_metrics,
    callbacks=[EarlyStoppingCallback(early_stopping_patience=2)]  # Stop if no␣
 ↪improvement after 2 steps
)

trainer_tweets = Trainer(
    model=model,
    args=training_args,
    train_dataset=tweets_dataset['train'],
    eval_dataset=tweets_dataset['test'],
    tokenizer=tokenizer,
    compute_metrics=compute_metrics,
    callbacks=[EarlyStoppingCallback(early_stopping_patience=2)]  # Stop if no␣
 ↪improvement after 2 steps
)
```

1.2 Dropout Technique Explanation: - Dropout is a regularization technique where randomly selected neurons are "dropped" during training. This prevents the network from becoming too reliant on specific neurons and encourages generalization.

Implementation: - Dropout is typically added as a layer in the model architecture. Since we are using pre-trained models, we can fine-tune them by modifying the dropout rate during training. In Hugging Face's transformers, the dropout rate is already a parameter for models like BERT and DistilBERT.

1.3 Cross-Validation Implementation Technique Explanation: - Cross-validation divides the data into multiple subsets, training the model on each subset while validating on the others. This ensures that the model's performance is consistent across different data splits, providing a more reliable measure of generalization.

[45]:
```
# Model configuration
model_name = "distilbert-base-uncased"
dropout_rate = 0.3
num_labels = 3  # For sentiment classification: negative, neutral, positive

# Initialize model with dropout
model = DistilBertForSequenceClassification.from_pretrained(model_name,␣
 ↪num_labels=num_labels, dropout=dropout_rate)
```

```
Some weights of DistilBertForSequenceClassification were not initialized from
the model checkpoint at distilbert-base-uncased and are newly initialized:
['classifier.bias', 'classifier.weight', 'pre_classifier.bias',
'pre_classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

[52]:
```
from sklearn.model_selection import KFold
import numpy as np
import torch
from transformers import Trainer, TrainingArguments, EarlyStoppingCallback,␣
 ↪DistilBertForSequenceClassification
import matplotlib.pyplot as plt

# Assuming news_dataset is a DatasetDict, we'll extract the training portion␣
 ↪for cross-validation
news_dataset_train = news_dataset['train']

# Cross-validation setup
kf = KFold(n_splits=5, shuffle=True, random_state=42)
cross_val_accuracies = []
cross_val_losses = []

# Convert the training dataset to a pandas DataFrame
news_dataset_train_df = news_dataset_train.to_pandas()

for fold, (train_index, test_index) in enumerate(kf.
 ↪split(news_dataset_train_df)):
    print(f"Training fold {fold + 1}/{kf.n_splits}")
```

```
    # Select the rows for train and test using the indices from KFold
    train_split = news_dataset_train.select(train_index.tolist())
    test_split = news_dataset_train.select(test_index.tolist())

    # Reinitialize model for each fold
    model = DistilBertForSequenceClassification.from_pretrained(model_name,␣
↪num_labels=num_labels, dropout=dropout_rate)
    model.to(torch.device("mps") if torch.backends.mps.is_built() else torch.
↪device("cpu"))

    # Define training arguments with aligned save and evaluation strategy
    training_args = TrainingArguments(
        output_dir="./results",
        evaluation_strategy="epoch",  # Save and evaluate at the end of every␣
↪epoch
        save_strategy="epoch",  # Align saving strategy with evaluation strategy
        per_device_train_batch_size=16,
        per_device_eval_batch_size=16,
        num_train_epochs=3,  # Adjust as needed
        weight_decay=0.01,
        learning_rate=5e-5,
        logging_dir='./logs',
        logging_steps=10,
        load_best_model_at_end=True  # This is required for␣
↪EarlyStoppingCallback
    )

    # Implement early stopping
    early_stopping = EarlyStoppingCallback(early_stopping_patience=2)

    # Initialize Trainer
    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=train_split,
        eval_dataset=test_split,
        tokenizer=tokenizer,
        callbacks=[early_stopping],
        compute_metrics=compute_metrics,
    )

    # Train and evaluate
    trainer.train()
    eval_result = trainer.evaluate()

    # Store cross-validation results
```

```
    cross_val_accuracies.append(eval_result['eval_accuracy'])
    cross_val_losses.append(eval_result['eval_loss'])

# Calculate and visualize results
average_accuracy = np.mean(cross_val_accuracies)
average_loss = np.mean(cross_val_losses)

# Visualization of the cross-validation results
epochs = np.arange(1, len(cross_val_accuracies) + 1)
```

```
Training fold 1/5
```

```
Some weights of DistilBertForSequenceClassification were not initialized from
the model checkpoint at distilbert-base-uncased and are newly initialized:
['classifier.bias', 'classifier.weight', 'pre_classifier.bias',
'pre_classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
Training fold 2/5
```

```
/Users/yuyao/anaconda3/lib/python3.11/site-
packages/huggingface_hub/file_download.py:1150: FutureWarning: `resume_download`
is deprecated and will be removed in version 1.0.0. Downloads always resume when
possible. If you want to force a new download, use `force_download=True`.
  warnings.warn(
Some weights of DistilBertForSequenceClassification were not initialized from
the model checkpoint at distilbert-base-uncased and are newly initialized:
['classifier.bias', 'classifier.weight', 'pre_classifier.bias',
'pre_classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

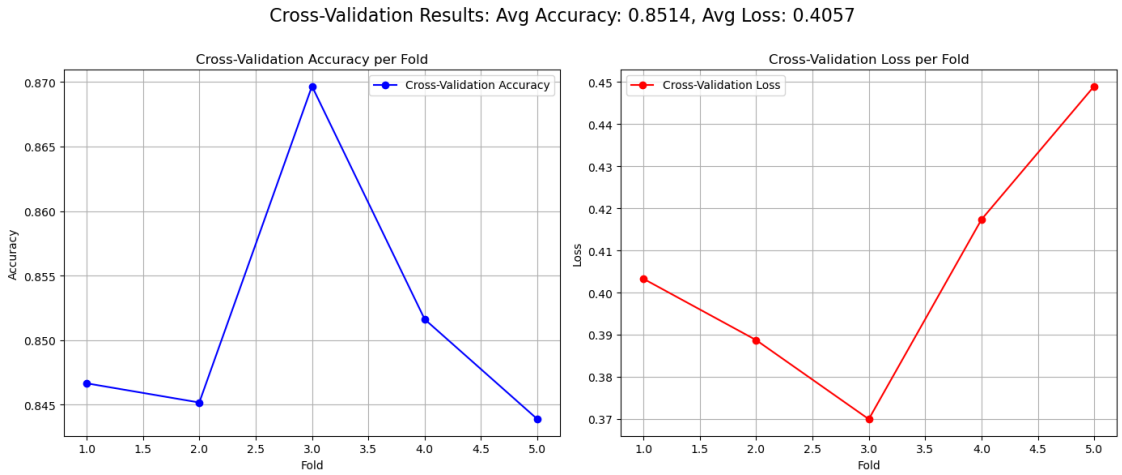
```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

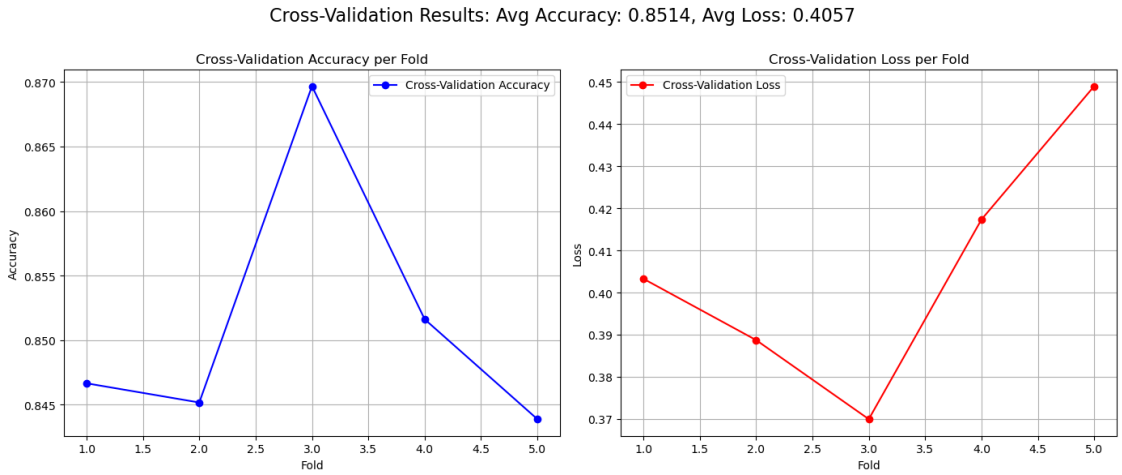
```
Training fold 3/5
```

```
/Users/yuyao/anaconda3/lib/python3.11/site-
packages/huggingface_hub/file_download.py:1150: FutureWarning: `resume_download`
is deprecated and will be removed in version 1.0.0. Downloads always resume when
possible. If you want to force a new download, use `force_download=True`.
  warnings.warn(
Some weights of DistilBertForSequenceClassification were not initialized from
the model checkpoint at distilbert-base-uncased and are newly initialized:
['classifier.bias', 'classifier.weight', 'pre_classifier.bias',
'pre_classifier.weight']
```

```
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Training fold 4/5

/Users/yuyao/anaconda3/lib/python3.11/site-
packages/huggingface_hub/file_download.py:1150: FutureWarning: `resume_download`
is deprecated and will be removed in version 1.0.0. Downloads always resume when
possible. If you want to force a new download, use `force_download=True`.
  warnings.warn(
Some weights of DistilBertForSequenceClassification were not initialized from
the model checkpoint at distilbert-base-uncased and are newly initialized:
['classifier.bias', 'classifier.weight', 'pre_classifier.bias',
'pre_classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Training fold 5/5

/Users/yuyao/anaconda3/lib/python3.11/site-
packages/huggingface_hub/file_download.py:1150: FutureWarning: `resume_download`
is deprecated and will be removed in version 1.0.0. Downloads always resume when
possible. If you want to force a new download, use `force_download=True`.
  warnings.warn(
Some weights of DistilBertForSequenceClassification were not initialized from
the model checkpoint at distilbert-base-uncased and are newly initialized:
['classifier.bias', 'classifier.weight', 'pre_classifier.bias',
'pre_classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```



[54]:
```
plt.figure(figsize=(14, 6))

# Subplot 1: Cross-validation Accuracy
plt.subplot(1, 2, 1)
plt.plot(epochs, cross_val_accuracies, marker='o', color='blue',␣
 ↪label='Cross-Validation Accuracy')
plt.title('Cross-Validation Accuracy per Fold')
plt.xlabel('Fold')
plt.ylabel('Accuracy')
plt.grid(True)
plt.legend()

# Subplot 2: Cross-validation Loss
plt.subplot(1, 2, 2)
plt.plot(epochs, cross_val_losses, marker='o', color='red',␣
 ↪label='Cross-Validation Loss')
plt.title('Cross-Validation Loss per Fold')
plt.xlabel('Fold')
plt.ylabel('Loss')
plt.grid(True)
plt.legend()

# Final layout adjustments and display
plt.suptitle(f"Cross-Validation Results: Avg Accuracy: {average_accuracy:.4f},␣
 ↪Avg Loss: {average_loss:.4f}", fontsize=16)
plt.tight_layout()
plt.subplots_adjust(top=0.85)
plt.show()
```



Subplot 1: Cross-Validation Accuracy per Fold

- **General Trend:**
  - The accuracy across the folds shows some variability.
  - Fold 3 has the highest accuracy at approximately **0.870**.
  - Fold 5 has the lowest accuracy, slightly above **0.840**.
- **Insights:**
  - The variability in accuracy suggests that the model's performance may be influenced by the specific data distribution in each fold.
  - However, the overall accuracy remains stable within a relatively small range (~0.84 to ~0.87), indicating consistent performance.
  - The **average accuracy** across all folds is **0.8514**, which is a solid result, showing that the model generalizes well across different subsets of the dataset.

Subplot 2: Cross-Validation Loss per Fold

- **General Trend:**
  - The validation loss decreases for the first three folds, with the lowest loss observed in **fold 3** at approximately **0.37**.
  - After fold 3, the validation loss increases, peaking at around **0.45** in fold 5.
- **Insights:**
  - The increasing loss in the later folds (fold 4 and fold 5) could indicate that the model struggles to generalize as well on those specific data subsets.
  - The overall **average loss** is **0.4057**, which is reasonably low, suggesting the model is performing well on the task, despite the slight variation between folds.

2.1.2 Summary of Results:

- **Performance Consistency:** The model achieves relatively consistent accuracy across all folds, with an average accuracy of **85.14%** and a low variation range (~0.84 to ~0.87). This suggests that the model is robust and generalizes well across different subsets of the data.

- **Loss Analysis:** The loss per fold shows more variability than the accuracy. Although the loss increases in the last two folds, it remains within an acceptable range, and the model maintains good performance overall.
- **Fold Variability:** The variation in both accuracy and loss across folds may suggest differences in the data distributions across the splits. This is common in cross-validation, especially if the dataset contains some degree of complexity or imbalance.

2.1.3 Potential Improvements:

1. **Data Balancing:** Check for potential imbalances in the dataset splits, which might explain the variability in performance across folds.
2. **Model Tuning:** Further tuning the hyperparameters (e.g., learning rate, dropout rate, etc.) might help to stabilize the performance across folds.
3. **Advanced Regularization:** Introducing more advanced regularization techniques, such as **gradient clipping** or **learning rate scheduling**, may help reduce the loss in the higher-loss folds.

[ ]: