

Classifying Volatility Reversals Based on Daily News Headlines

Goal:

To fine-tune an LLM that can accurately classify volatility reversals by analysing daily news headlines, helping traders and investors make more informed decisions about managing risk and timing trades.

Volatility reversals are critical events in financial markets, often signaling significant changes in market sentiment and potential shifts in asset prices. Accurately identifying these reversals enables traders and analysts to anticipate sudden shifts in market direction, improving risk management and trading strategies. By identifying potential reversals, investors can optimize entry and exit points, reduce losses, and enhance returns. Additionally, accurate classification helps in developing algorithmic trading models that adapt to changing market conditions, ultimately leading to more resilient and efficient trading systems.

Rationale:

Traders and investors are constantly being bombarded with an endless stream of news and information, some of which hold contrarian or opposing views from the current market conditions. However, traders and investors do not usually act upon 1 piece of news alone, but rather the overall market sentiment. This is where the skill of the individual comes into play; knowing exactly when the current market sentiment has shifted, and what actions to take next. I want to identify moments when negative or contrarian views, which may have been wrong for a while, start to get serious market attention and actually cause a volatility reverse. I later define what constitutes volatility reversal below. Being a data scientist, instead of relying on years of trading experience, I shall create and fine-tune a model to emulate the knowledge of experts in the finance industry.

Imports

```
In [9]: import pandas as pd
import numpy as np
# from datetime import timedelta, datetime
# from IPython.display import HTML
# import refinitiv.data as rd
import yfinance as yf
from sklearn.feature_selection import mutual_info_classif
from sklearn.model_selection import train_test_split
# from google.colab import drive
import torch
from transformers import BertTokenizer, BertForSequenceClassification, Trainer,
```

```
from sklearn.utils.class_weight import compute_class_weight
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
```

Data Collection - Financial News

Access to Refinitiv Data Platform obtained by requesting for an account via NUS Business Financial Database. <https://www.lseg.com/en/data-analytics/products/workspace>

Data is obtained between the dates 2023-11-15 to 2025-02-13. 2023-11-15 is the earliest date that headlines could be retrieved, any earlier returns NaN.

I queried for headlines relevant to S&P as the volatility index I will be using, VIX, tracks the volatility of the S&P. I also set a filter for English headlines. It is possible to translate non-English headlines using Google Translator API, however due to time constraints, this step has been forgone.

All news headlines for a given day are then aggregating into a single string. The rationale for this is explained below in the feature engineering section. However, the code is included here as the aggregation was done of the Refinitiv Data Platform itself before extracting the headlines.

```
In [ ]: '''
# Start session
rd.open_session()

headlines = pd.DataFrame()
file_num = 1
# Get headlines for past 460 days (2023-11-15 to 2025-02-13). Checked that 2023-
for i in range(460):
    date = datetime.strptime('2025-02-13', '%Y-%m-%d') + timedelta(days=-i)
    print(date)
    # Actually query for headlines here
    daily_headlines = rd.news.get_headlines("S&P AND Language:LEN", start=date,
    # Combine all headlines for the day into one string using [SEP] as separator
    daily_combined_headlines = daily_headlines.iloc[:, 0].astype(str).str.cat(se
    # Store the data in a DataFrame
    df_daily = pd.DataFrame([{"date": date, "combined_headlines": daily_combined
    headlines = pd.concat([headlines, df_daily], axis=0)
    # Save the data every 50 days since it takes a long time to get all the data
    if i % 50==0 and i!=0:
        headlines.to_csv(f"headlines_{file_num}.csv", encoding='utf-8', index=Fa
        headlines = pd.DataFrame()
        print(f"headlines_{file_num}.csv ABOVE")
        file_num += 1
# Save the remaining data
headlines.to_csv(f"headlines_{file_num}.csv", encoding='utf-8', index=False)
print(f"headlines_{file_num}.csv ABOVE")
'''
```

Data Collection - Volatility Index

Get daily VIX data from yfinance. Dates are matched to days with headlines from Refinitiv Data Platform.

```
In [2]: # Get VIX data from yfinance
vix_data = yf.download("^VIX", start='2023-11-15', end='2025-02-14')

# Save VIX data to CSV
vix_data.columns = ["Close", "High", "Low", "Open", "Volume"] # Rename columns
vix_data.to_csv("../data/VIX_data.csv") # Save to CSV

# Display VIX data
pd.concat([vix_data.head(), vix_data.tail()])

[*****100%*****] 1 of 1 completed
```

```
Out[2]:
```

	Close	High	Low	Open	Volume
Date					
2023-11-15	14.180000	14.350000	13.97	14.210000	0
2023-11-16	14.320000	14.420000	13.68	14.120000	0
2023-11-17	13.800000	14.190000	13.67	14.180000	0
2023-11-20	13.410000	14.310000	13.39	14.260000	0
2023-11-21	13.350000	14.310000	13.13	13.450000	0
2025-02-07	16.540001	16.660000	14.79	15.380000	0
2025-02-10	15.810000	16.610001	15.70	16.580000	0
2025-02-11	16.020000	16.420000	15.75	16.129999	0
2025-02-12	15.890000	17.180000	15.64	15.910000	0
2025-02-13	15.100000	16.330000	14.98	15.970000	0

Feature Engineering - Financial News

All news headlines for a given day are then aggregating into a single string. Doing this has a number of benefits over classifying each individual headline.

1. Market Sentiment Context: Financial markets react to the overall sentiment of the day's news rather than individual headlines. A single news article might not capture the full picture, but aggregating all headlines provides a more comprehensive view of market sentiment.
2. Noise Reduction: Individual headlines can be misleading or lack sufficient context for accurate classification, especially contrarian views that turn out to be wrong. By aggregating headlines, outliers are smoothed out, reducing the impact of sensational or less relevant news. This provides a more robust and stable representation of the day's sentiment compared to classifying and averaging individual headlines, which may introduce unnecessary variance.

When aggregating the headlines, I used the separator [SEP] to preserve each headlines' boundaries, preventing them from merging into an indistinguishable block of text. This helps models recognize distinct pieces of information.

It takes a long time to query all the data and the connection might time out, causing me to lose data. I split up the output file by saving every 50 queries allowing me to continue from when the connection was lost. Restarting the full query also faced the problem of hitting the API request limit.

However, I realised that the length of the combined daily headlines likely exceeds the token limit for FinBERT. In order to circumvent this, I used a chunking approach, splitting the daily combined headlines into 100 concatenated headlines at once. I chose chunking over classifying each individual headline in order to preserve context across multiple headlines. Additionally, each chunk within a day likely follow similar distributions. This chunking is done under the Feature Engineering - Merging section.

Feature Engineering - Volatility Index

This heuristic classifies daily movements in the VIX (CBOE Volatility Index) into **Positive Reversal**, **Negative Reversal**, or **No Reversal** based on specific conditions.

This classification will later be used as the labels for the the news headlines.

1. Compute Daily Changes

Let **Close** be the closing price of VIX on a given day.

Define **PrevClose** as the previous day's closing price.

Define **PrevClose_2** as the closing price two days prior.

Compute the daily percentage change in VIX:

$$\text{Change} = (\text{Close} - \text{PrevClose}) / \text{PrevClose}$$

Compute the previous day's percentage change (**PrevChange**).

2. Define Reversal Conditions

Positive Reversal:

The previous day's closing price was higher than two days ago (**PrevClose > PrevClose_2**).

The current closing price is lower than the previous day (**Close < PrevClose**).

The previous day's change (**PrevChange**) was greater than +5% (**PrevChange > 0.05**).

Negative Reversal:

The previous day's closing price was lower than two days ago (`PrevClose < PrevClose_2`).

The current closing price is higher than the previous day (`Close > PrevClose`).

The previous day's change (`PrevChange`) was less than -5% (`PrevChange < -0.05`).

3. Final Classification

If Positive Reversal conditions are met → Label as " `Positive` ".

If Negative Reversal conditions are met → Label as " `Negative` ".

Otherwise → Label as " `No Reversal` ".

This approach ensures that only significant trend reversals (where the previous day's movement exceeded $\pm 5\%$) are captured, filtering out minor fluctuations. The results are saved as a CSV file for further analysis.

```
In [3]: # Load data
vix_data = pd.read_csv("../data/VIX_data.csv") # Ensure dataset has 'Date' and

# Compute daily metrics
vix_data['PrevClose'] = vix_data['Close'].shift(1)
vix_data['PrevClose_2'] = vix_data['Close'].shift(2)
vix_data['Change'] = vix_data['Close'].pct_change() # Daily percentage change
vix_data['PrevChange'] = vix_data['Change'].shift(1)

# Define Reversal Conditions
vix_data['Positive_Reversal'] = (vix_data['PrevClose'] > vix_data['PrevClose_2']
vix_data['Negative_Reversal'] = (vix_data['PrevClose'] < vix_data['PrevClose_2']

# Label days
vix_data['Reversal_Label'] = 'No Reversal' # Default Label
vix_data.loc[vix_data['Positive_Reversal'], 'Reversal_Label'] = 'Positive'
vix_data.loc[vix_data['Negative_Reversal'], 'Reversal_Label'] = 'Negative'

# Save results
vix_data = vix_data[['Date', 'Close', 'Reversal_Label']]
vix_data.to_csv("../data/VIX_reversals.csv", index=False)
pd.concat([vix_data.head(), vix_data.tail()], axis=0)
```

Out[3]:

	Date	Close	Reversal_Label
0	2023-11-15	14.180000	No Reversal
1	2023-11-16	14.320000	No Reversal
2	2023-11-17	13.800000	No Reversal
3	2023-11-20	13.410000	No Reversal
4	2023-11-21	13.350000	No Reversal
307	2025-02-07	16.540001	No Reversal
308	2025-02-10	15.810000	Positive
309	2025-02-11	16.020000	No Reversal
310	2025-02-12	15.890000	No Reversal
311	2025-02-13	15.100000	No Reversal

Depending on how quickly traders react to news, I need to decide if I should lag the news headlines.

The `mutual_info_classif` function from `sklearn.feature_selection` estimates the mutual information (MI) between each feature and the target variable.

Mutual information measures the amount of information one variable provides about another. Higher values indicate stronger relationships.

MI is non-negative, meaning it always ranges from 0 to positive values.

A value of 0 means the feature and target are independent (no predictive power).

Higher MI means the feature contains more information about the target.

```
In [4]: # Read in daily combined headlines
headlines = pd.DataFrame()
for i in range(1, 11):
    headlines = pd.concat([headlines, pd.read_csv(f'../data/headlines_{i}.csv')])
headlines = headlines.dropna()

# Read in VIX reversals
vix_reversals = pd.read_csv('../data/VIX_reversals.csv')

# Merge headlines with VIX reversals (test with inner join for now, will forward
df_lag_test = headlines.merge(vix_reversals, left_on='date', right_on='Date', ho

# Convert labels to numeric (for correlation analysis)
label_map = {'No Reversal': 0, 'Positive': 1, 'Negative': 2}
df_lag_test['Reversal_Label_Num'] = df_lag_test['Reversal_Label'].map(label_map)

# Shift labels to simulate different lags
df_lag_test['Label_1DayLag'] = df_lag_test['Reversal_Label_Num'].shift(-1)
df_lag_test['Label_2DayLag'] = df_lag_test['Reversal_Label_Num'].shift(-2)
df_lag_test['Label_3DayLag'] = df_lag_test['Reversal_Label_Num'].shift(-3)
df_lag_test['Label_4DayLag'] = df_lag_test['Reversal_Label_Num'].shift(-4)
df_lag_test['Label_5DayLag'] = df_lag_test['Reversal_Label_Num'].shift(-5)
```

```

df_lag_test['Label_6DayLag'] = df_lag_test['Reversal_Label_Num'].shift(-6)
df_lag_test['Label_7DayLag'] = df_lag_test['Reversal_Label_Num'].shift(-7)

# Compute Mutual Information
X = df_lag_test[['Reversal_Label_Num', 'Label_1DayLag', 'Label_2DayLag', 'Label_
y = df_lag_test['Reversal_Label_Num'].iloc[:len(X)] # Align y with X

mutual_info = mutual_info_classif(X, y, discrete_features=True)
print(f"Mutual Info for no lag: {mutual_info[0]:.4f}")
print(f"Mutual Info for 1-day lag: {mutual_info[1]:.4f}")
print(f"Mutual Info for 2-day lag: {mutual_info[2]:.4f}")
print(f"Mutual Info for 3-day lag: {mutual_info[3]:.4f}")
print(f"Mutual Info for 4-day lag: {mutual_info[4]:.4f}")
print(f"Mutual Info for 5-day lag: {mutual_info[5]:.4f}")
print(f"Mutual Info for 6-day lag: {mutual_info[6]:.4f}")
print(f"Mutual Info for 7-day lag: {mutual_info[7]:.4f}")

```

```

Mutual Info for no lag: 0.5686
Mutual Info for 1-day lag: 0.0303
Mutual Info for 2-day lag: 0.0144
Mutual Info for 3-day lag: 0.0122
Mutual Info for 4-day lag: 0.0108
Mutual Info for 5-day lag: 0.0120
Mutual Info for 6-day lag: 0.0058
Mutual Info for 7-day lag: 0.0021

```

No lag (0 days) has the highest MI (0.5686), meaning today's news headlines have the strongest relationship with the VIX reversal label.

As the lag increases (1 day, 2 days, etc.), the MI drops sharply, meaning older headlines are less useful in predicting reversals.

By 7 days, MI is nearly 0.0021, indicating almost no relationship.

This aligns well with my chunking approach as traders can test new chunks at a time and identify if they are presently undergoing a volatility reversal.

Feature Engineering - Merge Datasets

VIX only has data on training days. compile the news over non trading days to classify for upcoming trading day.

As previously mentioned, the chunking of concatenated news headlines is done here.

```

In [8]: # Read in daily combined headlines
headlines = pd.DataFrame()
for i in range(1, 11):
    headlines = pd.concat([headlines, pd.read_csv(f'../data/headlines_{i}.csv')])
headlines = headlines.dropna()

# Read in VIX reversals
vix_reversals = pd.read_csv('../data/VIX_reversals.csv')

# Merge daily headlines with VIX reversals
labelled_news = pd.merge(headlines, vix_reversals, left_on='date', right_on='Date')
# VIX reversals are only available for trading days, so we back fill the NaN Dat

```

```

labelled_news['Date'] = labelled_news['Date'].bfill()
labelled_news = labelled_news.groupby('Date').agg({'combined_headlines': '[SEP]')

labelled_news.to_csv('../data/labelled_news.csv', index=False)

# I realised that the combined_headlines column likely exceeds FinBERT's token l
def split_by_sep(row):
    # Split combined_headlines by '[SEP]'
    segments = row['combined_headlines'].split('[SEP]')
    # Group into chunks of 100
    chunks = [segments[i:i + 100] for i in range(0, len(segments), 100)]
    return [(row['Date'], row['Reversal_Label'], ' [SEP] '.join(chunk)) for chunk
expanded_rows = [item for sublist in labelled_news.apply(split_by_sep, axis=1) f
chunked_news = pd.DataFrame(expanded_rows, columns=['Date', 'Reversal_Label', 'c
chunked_news.to_csv('../data/chunked_news.csv', index=False)
chunked_news

```

Out[8]:

	Date	Reversal_Label	combined_headlines
0	2023-11-15	No Reversal	WIPO Publishes Patent of DIRECTA PLUS S.P.A. w...
1	2023-11-15	No Reversal	Invesco S&P 500 Downside Hedged (PHDG:\$32.61) ...
2	2023-11-15	No Reversal	Invesco S&P SmallCap Momentum (XSMO:\$49.95) fa...
3	2023-11-15	No Reversal	Lyxor S&P 500 UCITS - Daily Hedged D-EUR (SP5H...
4	2023-11-15	No Reversal	iShares Core S&P BSE SENSEX India (2836:HKD35...
...
2975	2025-02-13	No Reversal	Top Performers Past Week: Palantir Technolog...
2976	2025-02-13	No Reversal	S&P 500 Consumer Staples (Sector): The Top Fiv...
2977	2025-02-13	No Reversal	General Mills offers 46th lowest Price Earning...
2978	2025-02-13	No Reversal	SPDR S&P MidCap 400 (MDY:\$579.11) in 2nd conse...
2979	2025-02-13	No Reversal	Consolidated Planning Corp Acquires 4,066 Shar...

2980 rows × 3 columns

Dataset Imbalance

In [7]:

```

# Load the dataset
chunked_news = pd.read_csv('../data/chunked_news.csv')

# Display the counts and percentages
pd.DataFrame({
    'Count': chunked_news['Reversal_Label'].value_counts(),
    'Percentage': chunked_news['Reversal_Label'].value_counts(normalize=True) *
})

```


Out[7]:

	Count	Percentage
Reversal_Label		
No Reversal	2522	84.630872
Positive	298	10.000000
Negative	160	5.369128

As I am using textual data, instead of oversampling or undersampling, I will adjust the loss function of the model below to give higher weights to the minority classes.

FinBERT Setup

```
In [ ]: # Read in processed data
df = pd.read_csv('../data/chunked_news.csv')

# Preprocess the data
label_map = {'No Reversal': 0, 'Positive': 1, 'Negative': 2}
df['label'] = df['Reversal_Label'].map(label_map)

# Split the data into training, validation, and test sets
train_texts, temp_texts, train_labels, temp_labels = train_test_split(df['combined_text'], df['label'],
                               train_size=0.8, random_state=42)
val_texts, test_texts, val_labels, test_labels = train_test_split(temp_texts, temp_labels,
                           train_size=0.5, random_state=42)

# Load the tokenizer
tokenizer = BertTokenizer.from_pretrained('yiyanghkust/finbert-tone')

# Tokenize the text data
train_encodings = tokenizer(train_texts.tolist(), truncation=True, padding=True, max_length=128)
val_encodings = tokenizer(val_texts.tolist(), truncation=True, padding=True, max_length=128)
test_encodings = tokenizer(test_texts.tolist(), truncation=True, padding=True, max_length=128)

# Convert the data into torch tensors
class NewsDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {'key': torch.tensor(val[idx]) for key, val in self.encodings.items}
        item['label_ids'] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.labels)

train_dataset = NewsDataset(train_encodings, train_labels.tolist())
val_dataset = NewsDataset(val_encodings, val_labels.tolist())
test_dataset = NewsDataset(test_encodings, test_labels.tolist())
```

FinBERT Fine-Tuning

Tokenization:

1. Label Encoding To facilitate supervised learning, the categorical labels ("No Reversal," "Positive," and "Negative") are mapped to numerical values: 0: No Reversal, 1: Positive Reversal, 2: Negative Reversal
2. Dataset Splitting The dataset is divided into training (70%), validation (15%), and test (15%) subsets using `train_test_split`. This ensures that the model is trained on one portion of the data while another portion is held out for validation and testing.
3. Tokenization Using FinBERT The FinBERT tokenizer (`yyanghkust/finbert-tone`) is used to preprocess text data. The tokenizer converts input text into tokenized sequences with attention masks, ensuring a maximum length of 512 tokens to fit BERT's input requirements.

Preprocessing into Dataset class:

1. Custom PyTorch Dataset A PyTorch Dataset class is defined to handle tokenized data and corresponding labels efficiently. This class:
 - Stores the tokenized encodings (`input_ids`, `attention_mask`).
 - Converts these encodings into PyTorch tensors.
 - Provides a structured way to access samples and labels for training.
2. Conversion to PyTorch Dataset Format The processed data is encapsulated within the `NewsDataset` class, which allows efficient batching and loading using PyTorch's `Dataloader` for model training and evaluation.

This preprocessing ensures that the input data is structured, tokenized, and formatted correctly for training a FinBERT model on volatility reversal classification.

```
In [ ]: # Convert Label to numbers
label_map = {'No Reversal': 0, 'Positive': 1, 'Negative': 2} # Use 0, 1, 2 as re
df['label'] = df['Reversal_Label'].map(label_map)

# Splitting the dataset into training (70%), validation (15%), and test (15%) se
train_texts, temp_texts, train_labels, temp_labels = train_test_split(df['combin
val_texts, test_texts, val_labels, test_labels = train_test_split(temp_texts, te

# Loading the FinBERT tokenizer for text preprocessing
tokenizer = BertTokenizer.from_pretrained('yyanghkust/finbert-tone')

# Tokenizing the text data to convert raw text into token IDs and attention mask
train_encodings = tokenizer(train_texts.tolist(), truncation=True, padding=True,
val_encodings = tokenizer(val_texts.tolist(), truncation=True, padding=True, max
test_encodings = tokenizer(test_texts.tolist(), truncation=True, padding=True, m

# Creating a PyTorch dataset class for handling tokenized data
class NewsDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        # Dictionary containing input_ids and attention_mask
        self.encodings = encodings
        # Corresponding labels for each input
        self.labels = labels
```

```

def __getitem__(self, idx):
    # Extract the encoded values and convert them into PyTorch tensors
    item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items}
    # Label tensor
    item['label_ids'] = torch.tensor(self.labels[idx])
    return item

def __len__(self):
    # Return dataset size
    return len(self.labels)

# Converting tokenized data and labels into dataset objects
train_dataset = NewsDataset(train_encodings, train_labels.tolist())
val_dataset = NewsDataset(val_encodings, val_labels.tolist())
test_dataset = NewsDataset(test_encodings, test_labels.tolist())

```

Model Selection:

A pre-trained FinBERT model (yiyanghkust/finbert-tone) is loaded for sequence classification with three output labels. This model is optimized for financial sentiment analysis, making it well-suited for volatility reversal classification.

Training Configuration:

- 10 epochs of training
- Batch size of 32 for training and evaluation
- Cosine learning rate scheduler for gradual decay
- Mixed precision (FP16) to speed up training
- Early stopping after 3 epochs with no improvement
- Model checkpointing at each epoch for best model retrieval

Handling Imbalanced Data:

Since most data points belong to the No Reversal class, class weights are computed using `compute_class_weight` to assign higher weights to minority classes (Positive and Negative Reversals). These weights are passed to `CrossEntropyLoss` during training. A custom `Trainer` class is implemented to modify the loss function by incorporating class weights. This prevents the model from being biased toward the dominant class.

Model Training and Evaluation:

- The model is trained using the weighted loss function.
- After each epoch, it is evaluated on the validation set.
- The best-performing model is saved and tested on the test dataset to assess final performance.

```

In [ ]: # Load the pre-trained FinBERT model for sequence classification with 3 labels (
model = BertForSequenceClassification.from_pretrained('yiyanghkust/finbert-tone')

# Enable CuDNN optimization for faster training on compatible hardware
torch.backends.cudnn.benchmark = True

```

```

# I ran this on Colab, so I mounted my Google Drive to save the model checkpoint
# drive.mount('/content/gdrive')

# Define the training arguments for model fine-tuning
training_args = TrainingArguments(
    output_dir='/content/drive/MyDrive/finbert_checkpoints', # Directory to save
    num_train_epochs=10, # Train the model for 10 epochs
    per_device_train_batch_size=32, # Batch size for training
    per_device_eval_batch_size=32, # Batch size for evaluation
    warmup_steps=500, # Number of warmup steps for learning rate scheduler
    max_steps=1000, # Maximum number of training steps (early stopping may halt
    weight_decay=0.01, # L2 regularization for better generalization
    logging_dir='./logs', # Directory for logging training metrics
    logging_steps=10, # Log training metrics every 10 steps
    eval_strategy="epoch", # Evaluate the model at the end of every epoch
    save_strategy="epoch", # Save the model at the end of every epoch
    report_to="none", # Disable automatic reporting (e.g., to Weights & Biases)
    lr_scheduler_type="cosine", # Use a cosine learning rate scheduler
    learning_rate=1e-6, # Initial learning rate
    fp16 = True, # Enable mixed-precision training for better performance
    load_best_model_at_end=True # Load the best model at the end of training
)

# Compute class weights to handle imbalanced data
class_weights = compute_class_weight(
    class_weight="balanced", # Compute inverse class frequencies to balance the
    classes=np.array([0,1,2]), # Labels: No Reversal (0), Positive (1), Negative
    y=train_labels.to_numpy() # Extract Labels from training data
)

# Convert class weights to a tensor and move to GPU for training
class_weights = torch.tensor(class_weights, dtype=torch.float32).to("cuda")

# Define a custom Trainer class with weighted Loss computation
class WeightedTrainer(Trainer):
    def compute_loss(self, model, inputs, return_outputs=False, **kwargs):
        labels = inputs.get("labels") # Extract labels from inputs
        outputs = model(**inputs) # Forward pass through the model
        logits = outputs.get("logits") # Extract Logits (raw model predictions)

        # Apply CrossEntropyLoss with computed class weights
        loss_fct = torch.nn.CrossEntropyLoss(weight=class_weights)
        loss = loss_fct(logits, labels)


        return (loss, outputs) if return_outputs else loss # Return Loss (and o

# Create the Trainer with the customized Loss function
trainer = WeightedTrainer(
    model=model, # Use the pre-trained FinBERT model
    args=training_args, # Training configurations
    train_dataset=train_dataset, # Training dataset
    eval_dataset=val_dataset, # Validation dataset
    callbacks=[EarlyStoppingCallback(early_stopping_patience=3)] # Stop trainin
)

# Train the model using the training dataset
trainer.train()
# Evaluate the model using the validation dataset
trainer.evaluate()

```

```
# Evaluate the model using the test dataset and print results
print("Evaluating on the test set...")
test_results = trainer.evaluate(test_dataset)
print(f"Test results: {test_results}")
```

No description has been provided for this image

Training was ran on Colab, this is a screenshot of the results.

Metrics & Results Analysis

Accuracy: The percentage of correctly classified instances.

Precision: Measures how many of the predicted positive cases were actually correct.

Recall: Measures how many of the actual positive cases were correctly identified.


F1-Score: The harmonic mean of precision and recall, balancing both.

```
In [ ]: def compute_metrics(eval_pred):
        logits, labels = eval_pred
        predictions = logits.argmax(axis=-1) # Get predicted class
        precision, recall, f1, _ = precision_recall_fscore_support(labels, predictions)
        acc = accuracy_score(labels, predictions)
        return {
            "accuracy": acc,
            "precision": precision,
            "recall": recall,
            "f1": f1
        }

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    compute_metrics=compute_metrics
)

print("Evaluating on the test set...")
test_results = trainer.evaluate(test_dataset)

# Print evaluation metrics
print(f"Accuracy: {test_results['eval_accuracy']:.4f}")
print(f"Precision: {test_results['eval_precision']:.4f}")
print(f"Recall: {test_results['eval_recall']:.4f}")
print(f"F1 Score: {test_results['eval_f1']:.4f}")
```

No description has been provided for this image

Metrics was calculated on Colab. This is a screenshot of the results.

The fine-tuned FinBERT model demonstrates a moderate performance in classifying volatility reversals based on daily news headlines.

Accuracy (34.68%): This metric indicates that the model correctly predicts the reversal labels for about one-third of the test set. While this is a modest result, it provides a baseline for further improvements.

Precision (72.44%): Precision measures the proportion of true positive predictions among all positive predictions. A precision score of 72.44% suggests that when the model predicts a reversal, it is correct approximately 72% of the time. This is important for minimizing false positives, which can lead to unnecessary trading actions.

Recall (34.68%): Recall measures the proportion of actual reversals that the model correctly identifies. A recall score of 34.68% indicates that the model successfully identifies about 35% of all actual reversals, highlighting the need for better recall to capture more significant market shifts.

F1 Score (42.59%): The F1 score balances precision and recall, providing a single metric that reflects the model's overall performance. A score of 42.59% indicates a moderate performance, balancing the trade-off between precision and recall.

Conclusion

By leveraging FinBERT and fine-tuning it on this specific task, we have created a model that can assist traders and investors in making more informed decisions by anticipating significant shifts in market volatility.

The moderate accuracy of the FinBERT model in predicting volatility reversals suggests that while the model has some predictive power, there is room for improvement. Leveraging natural language processing (NLP) techniques in financial analysis remains valuable, as the model's high precision indicates that it can reliably identify true reversals when it makes a prediction.

Future Work

Future work could explore further improvements, such as

1. Experimenting with different model architectures such as LSTM to capture temporal dependencies as previous days' volatility and news may affect future volatility.
2. Incorporating additional features such as other financial metrics like trading volume, interest rates, forex. This will likely further improve the model's predictive capabilities.

Strengths

Being proactive to seek out the Refinitiv Data Platform gave me access to trustable and relatively clean sources of news.

I also have previously read up on financial concepts and I believe it has helped me contextualise the project well, especially helping me to carry out exploratory data analysis. It was really interesting to finally marry finance concepts with actual working data science practices.

Using FinBERT probably helped the model perform better as it is specialised to financial knowledge. I considered using Longformers allowing me to pass in their entire concatenated daily headlines. However, I prioritised FinBERT for it's specialised knowledge. Perhaps more testing on this could be done.

I used the chunking approach as I believe aggregating the news is more useful than having the model learn from individual headlines. From personal experience, headlines confuse me more than it helps me because I'm stuck thinking if that piece is a contrarian view, or if the rest of the market will react in the same way. By aggregating the headlines, the model can learn the other market sentiment, factoring in the presence of contrarian views.

Weaknesses

Computing power, especially the lack of access to a local GPU, handicapped the training process. I was constrained by Colab's GPU maximum run times, which after tweaking and re-training the model, quickly reached the limit.

Refinitiv Data Platform also limited the amount of news I could extract. Additionally, their dataset only went back to 2023-11-15. Perhaps with more historical data to train on, the model may yield better results.

Using the Refinitiv Data Platform was rather challenging as information and instructions are harder to come by given it's a paywalled and typically B2B service. It was difficult to find answers for questions I had on navigating and using the platform to obtain the necessary and relevant data. One challenge was figuring out that the API through VSCode does not work, and that I have to access the API via the Refinitiv Workspace on their Codeblocks notebook app. Another challenge was querying the data as the API had documentation, however the parameter was just "query" but no instructions on how to actually write the query. I believe with better querying to obtain more relevant news might yield better results.