9. DistilBERTSentimentAnalysis

September 6, 2023

[ ]:
```python
import pandas as pd
import os
from transformers import pipeline
```

[ ]:
```python
# Load CSV Data

# Stocks :- AAPL, MSFT, AMZN, NVDA, TSLA, GOOGL
# Sector Indices :- SSINFT (^SP500-45)


ticker = "AAPL"
method = "DistilBERT"
```

[ ]:
```python
# Load the news artcile file
#df = pd.read_csv(f"PreProcessedContextArticles/{ticker}_news_data.csv")

# Load sentiment analysis transformer
sentiment_pipeline = pipeline("text-classification",␣
↪model="distilbert-base-uncased-finetuned-sst-2-english")
```

[ ]:
```python
# Due to imput size limitaiton, we need oto feed each article into the␣
↪transformer and ged the sentiment score
# 3. Sentiment Analysis
def analyze_sentiment(text):
    response = sentiment_pipeline(str(text))
    if response[0]['label'] == 'NEGATIVE':
        return -response[0]['score']
    elif response[0]['label'] == 'POSITIVE':
        return response[0]['score']
    else:
        return 0

#df['sentiment_score'] = df['Headline'].apply(analyze_sentiment)
```

[ ]:
```python
# 5. Output Sentiment Results
df.to_csv(f"SentimentAnalysis/{method}/{ticker}sentiment_output.csv",␣
↪index=False)
```

[ ]:
```python
# Load the HB Sentiment Results
df = pd.read_csv(f"SentimentAnalysis/{method}/{ticker}sentiment_output.csv")
```

[ ]:
```python
df
```

[ ]:
```python
# 6. Aggregate the sentiment score on a given day and calculate the overall␣
 ↪sentiment by taking each days positive and negative score sum and dividing␣
 ↪by total number of articels on that day
# Fill NaN values in the Summary column
df['Summary'].fillna("", inplace=True)

# Convert all values in 'Headline' and 'Summary' to strings
df['Headline'] = df['Headline'].astype(str)
df['Summary'] = df['Summary'].astype(str)

# Define aggregation functions
aggregations = {
    'Headline': ' '.join,
    'Summary': ' '.join,
    'sentiment_score': 'sum',
}

# Group by Date and aggregate
agg_df = df.groupby('Date').agg(aggregations).reset_index()

# Compute the average sentiment score
agg_df['polarity'] = agg_df['sentiment_score'] / df['Date'].value_counts().
 ↪sort_index().values
```

[ ]:
```python
agg_df
```

[ ]:
```python
# Convert the 'Date' column to datetime dtype (if it's not already)
agg_df['Date'] = pd.to_datetime(agg_df['Date'], format='%Y-%m-%d')
```

[ ]:
```python
# Sort the DataFrame by the 'Date' column
agg_df = agg_df.sort_values(by='Date')
```

[ ]:
```python
stock_df = pd.read_excel(f"PreProcessedStocks/{ticker}_stock_data.xlsx")

# Convert the 'Date' column to datetime dtype
stock_df['Date'] = pd.to_datetime(stock_df['Date'], format='%d/%m/%Y')
```

[ ]:
```python
# 7. Compare the the sentiment value to the following days price trend and get␣
 ↪the accuracy
merged_df = pd.merge(agg_df,stock_df,  on="Date", how='inner')
```

```
[ ]: # Use next day price trend to check the effect of news sentiment
merged_df['next_day_price_trend'] = merged_df['price_trend'].shift(-1)
```

```
[ ]: # Remove days with nuetral value for sentiment_label to simulate not trading on␣
↪those days since no clear directional sentiment was found.
merged_df = merged_df[~merged_df['next_day_price_trend'].isin(['neutral',␣
↪'None'])]

# Drop all rows without a "price_trend" value (removing non trading days)
merged_df = merged_df.dropna(subset=["price_trend", "next_day_price_trend"])
```

```
[ ]: # Convert sentiments to binary
merged_df['price_trend'] = merged_df['price_trend'].replace({'positive': 1,␣
↪'negative': 0})
merged_df['next_day_price_trend'] = merged_df['next_day_price_trend'].
↪replace({'positive': 1, 'negative': 0})
```

```
[ ]: merged_df
```

```
[ ]: # 5. Output Sentiment Results with stock price trend
merged_df.to_csv(f"SentimentAnalysis/{method}/
↪{ticker}sentiment_agg_stock_trend_output.csv", index=False)
```

```
[ ]: # Load Sentiment Results with stock price trend
# df = pd.read_csv(f"SentimentAnalysis/{method}/
↪{ticker}sentiment_agg_stock_trend_output.csv")
```

```
[ ]: ### Linear Discriminant Analysis (LDA) Model
#----------------------------------------------------------
```

```
[ ]: import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# Columns to keep
keep_columns = ['Open', 'High', 'Low', 'Volume', 'polarity', 'price_trend']
# keep_columns = ['Open', 'High', 'Low', 'Close', 'Volume', 'polarity',␣
↪'next_day_price_trend']
model_df1 = merged_df[keep_columns]
print(model_df1)
```

```
[ ]: # Creating the feature dataset
x = np.array(model_df1.drop(columns=['price_trend']))
# x = np.array(merged_df.drop(columns=['next_day_price_trend']))
# Creating the target dataset
y = np.array(model_df1['price_trend'])
```

```
# y = np.array(merged_df['next_day_price_trend'])

# Splitting the data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
↪random_state=0)

# Creating and training the model
model = LinearDiscriminantAnalysis().fit(x_train, y_train)

# Model's predictions
predictions = model.predict(x_test)
print(predictions)

print(y_test)

# Model metrics
print(classification_report(y_test, predictions))
```

[ ]:
```
### LSTM ( Long Short-Term Memory) Model
#--------------------------------------------------------
```

[ ]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Bidirectional, Dropout,␣
↪GlobalAveragePooling1D, MaxPooling1D, Conv1D
from tensorflow.keras.callbacks import ReduceLROnPlateau, ModelCheckpoint,␣
↪EarlyStopping
```

[ ]:
```
# Normalize data since LSTMs are sensitive to the scale of input data
scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)

# Reshape input data to be 3D [samples, timesteps, features]. In this case,␣
↪considering each row as 1 timestep.
x_train = x_train.reshape((x_train.shape[0], 1, x_train.shape[1]))
x_test = x_test.reshape((x_test.shape[0], 1, x_test.shape[1]))
```

[ ]:
```
# Callbacks
# reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5,␣
↪min_lr=1e-6)
# checkpoint = ModelCheckpoint('best_model.h5', monitor='val_accuracy',␣
↪save_best_only=True, mode='max')
```

```
# early_stop = EarlyStopping(monitor='val_loss', patience=7)


#Build the Single layer LSTM model
# model = Sequential([
#     Bidirectional(LSTM(64, input_shape=(x_train.shape[1], x_train.shape[2]))),
#     Dense(64, activation='relu'),
#     Dense(1, activation='sigmoid')
# ])
# Accuracy 56%

# Build the BiDirectional LSTM model
model = Sequential([
    Bidirectional(LSTM(64, input_shape=(x_train.shape[1], x_train.shape[2]),␣
 ↪return_sequences=True)),
    Bidirectional(LSTM(32)),
    Dropout(0.5),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(1, activation='sigmoid')
])
# Accuracy 55%






model.compile(loss='binary_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
```

[ ]:
```
epochs = 200 #Accuracy 80%

# Train the LSTM model
history = model.fit(x_train, y_train, epochs= epochs, batch_size=32,␣
 ↪validation_data=(x_test, y_test), verbose=2, shuffle=False)
# history = model.fit(x_train, y_train, epochs= epochs, batch_size=32,␣
 ↪validation_data=(x_test, y_test), verbose=2, shuffle=False, ␣
 ↪callbacks=[reduce_lr, checkpoint, early_stop])

# Predictions
predictions = model.predict(x_test)
predictions = (predictions > 0.5).astype(int)

# Printing metrics
from sklearn.metrics import classification_report
print(classification_report(y_test, predictions))
```

[ ]:
```
# Print the model summary
model.summary()
```

[ ]:
```
# Evaluate the model on the testing dataset
loss, accuracy = model.evaluate(x_test, y_test)

import matplotlib.pyplot as plt

# Plot utility
def plot_graphs(history, string):
  plt.plot(history.history[string])
  plt.plot(history.history['val_'+string])
  plt.xlabel("Epochs")
  plt.ylabel(string)
  plt.legend([string, 'val_'+string])
  plt.show()

# Plot the accuracy and loss
plot_graphs(history, "accuracy")
plot_graphs(history, "loss")
```

[ ]:
```
### Gated Recurrent Unit (GRU) Model
#---------------------------------------------------------
```

[ ]:
```
from tensorflow.keras.layers import GRU
```

[ ]:
```
# Model Definition with GRU
# model = Sequential([
#     GRU(64, input_shape=(x_train.shape[1], x_train.shape[2])),
#     Dense(64, activation='relu'),
#     Dense(1, activation='sigmoid')
# ])

# Model Definition with GRU
model = Sequential([
    Bidirectional(GRU(64, input_shape=(x_train.shape[1], x_train.shape[2]))),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])

# Set the training parameters
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

[ ]:
```
epochs = 90

# Train the LSTM model
```

```
history = model.fit(x_train, y_train, epochs= epochs, batch_size=32,␣
 ↪validation_data=(x_test, y_test), verbose=2, shuffle=False)
# history = model.fit(x_train, y_train, epochs= epochs, batch_size=32,␣
 ↪validation_data=(x_test, y_test), verbose=2, shuffle=False, ␣
 ↪callbacks=[reduce_lr, checkpoint, early_stop])

# Predictions
predictions = model.predict(x_test)
predictions = (predictions > 0.5).astype(int)

# Printing metrics
from sklearn.metrics import classification_report
print(classification_report(y_test, predictions))
```

[ ]:
```
# Print the model summary
model.summary()
```

[ ]:
```
# Evaluate the model on the testing dataset
loss, accuracy = model.evaluate(x_test, y_test)

# Plot the accuracy and loss
plot_graphs(history, "accuracy")
plot_graphs(history, "loss")
```

[ ]:
```

```