Configuration Manual

MSc Research Project
Data Analytics

Girija Madireddy
Student ID: x21235929

School of Computing
National College of Ireland

Supervisor: Dr.Athanasios Staikopoulos

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Girija Madireddy
Student ID:	x21235929
Programme:	Master Of Science in Data Analytics **Year:** 2023-24
Module:	MSc Research Project
Lecturer:	Dr.Athanasios Staikopoulos
Submission Due Date:	31-01-2024
Project Title:	Configuration Manual - **Optimal Timing for Stock Trading in USA: Data Driven Signals for Buy and Sell**

Word Count: .....1743..... **Page Count:** .....11.....

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Girija Madireddy

Date: 26-01-2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Girija Madireddy
Student ID: x21235929

1 Environment

This configuration manual can be used to reproduce research work and achieve the intended outcomes. The requirements for system configuration, the procedures for collecting data, cleaning it up, and using it to apply models are all covered in the handbook. Code snippets for major steps of the models were detailed in this report.

System configuration used for research work is detailed below.

System Configuration	
Operating System	Microsoft Windows 11 Pro
RAM	16 GB
Hard Disk	512 SSD
Processor	11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz, 2304 Mhz, 8 Core(s), 16 Logical Processor(s)
GPU	NVIDIA GeForce RTX 3050 Ti Laptop GPU

2 Tools and Libraries

Tools

This section describes about Tools used for development process. Installing these is a prerequisite to reproduce this work.

Tools	Version
Anaconda Navigator	2.4.0
Jupyter notebook	6.4.12
Python	3.9.13 (main, Aug 25 2022, 23:51:50) [MSC v.1916 64 bit (AMD64)]

Artifacts & Descriptions

This section describes about documents submitted as part of artifacts. First three csv files are datasets used for sentiment analysis. IPYNB files are Jupyter notebooks with python code for model development. Datasets need to be placed in the same folder as code files or code needs to be updated to new file location if they are placed separately.

Artifact	Description
RedditNews.csv	News headlines data from Reddit News
upload_DJIA_table.csv	DJIA index Stock Time series data
Combined_News_DJIA.csv	Combined data with News Headlines and Time series Data
DJIA_ARIMA.ipynb	Python code for Statistical ARIMA Model
DJIA_LSTM.ipynb	Python code for LSTM Model
DJIA_LSTM_NEWS_SENTIMENT.ipynb	Python code for LSTM-News Sentiment Model
GCN_LSTM_Final-500Epochs.ipynb	Python code for GCN-LSTM model with 500 Epochs
GCN_LSTM_Final-200Epochs.ipynb	Python code for GCN-LSTM model with 200 Epochs
GCN_LSTM_NEWS_SENTIMENT_Final_200Epochs.ipynb	Python code for GCN_LSTM-News Sentiment with 200 epochs
GCN_LSTM_NEWS_SENTIMENT_Final_500Epochs.ipynb	Python code for GCN_LSTM-News Sentiment with 500 epochs

Python Libraries

Python Library	Usage
yfinance	Download stock data from yfinance
pandas	Data Manipulation and analysis
numpy	Numerical computing in data preprocessing
sklearn.model_selection	scikit-learn library to split dataset into training and testing sets
nltk.sentiment	Natural Language Toolkit for sentiment analysis
re	Package for regular expressions used for text data preprocessing
networkx	Creating, manipulating and studying complex networks and graphs
matplotlib	Library to create interactive visualizations
PyTorch	Deep Learning library for training neural networks
torch.nn	Creating layers for Neural networks
torch.nn.functional	Functions for activation and convolution operations
torch_geometric	PyTorch library to build Graph based neural networks
torch.optim	PyTorch library to build and train neural networks especially for optimizing algorithms
sklearn.metrics	Used to evaluate performance of classification models
textblob	Used for calculating subjectivity and polarity for news sentiment
vaderSentiment.vaderSentiment	Valence Aware Dictionary and Sentiment Reasoner - Sentiment analysis dictionary
seaborn	Data Visualization library for statistical graphs
datetime	Manipulating and formatting date and time
sklearn.preprocessing	Scale, Transform and preprocess data
tensorflow.keras.models	Building and training neural networks
tensorflow.keras.layers	building blocks in neural network architecture
tensorflow.keras.optimizers	Optimization algorithms for training neural networks
tensorflow.random	Used in neural network training
warnings	Used for development and error investigation

3 Model Development and Execution

This section is detailed on execution of GCN-LSTM_News Sentiment model. Other models can be run by following same steps and referencing code files from artifacts.

3.1 Libraries to import

All libraries must be installed before running jupyter notebook. Jupyter notebook needs to be restarted and run from the beginning if any new libraries are installed at run time. Usage of each Python library is explained in section 2.

```
import yfinance as yf
import pandas as pd
import numpy as np
import torch
import torch.nn.functional as F
from torch_geometric.data import Data
from torch_geometric.nn import GCNConv
from sklearn.model_selection import train_test_split
import pandas as pd
from nltk.sentiment import SentimentIntensityAnalyzer
import re
import networkx as nx
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from sklearn.metrics import roc_curve, auc

import warnings
warnings.simplefilter("ignore")
```

3.2 Data Preprocessing and Graph Creation

Historical Stock market data is downloaded from yahoofinance using yfiance python library. Tickers are different companies in this study. Data is downloaded for 29 companies which are from DJIA index. This data will be downloaded dynamically when code is run on jupyter notebook. News headlines data is taken from RedditNews.csv

```
# Downloading stock data
tickers = ['CSCO', 'TRV', 'IBM', 'PFE', 'JNJ', 'AXP', 'GS', 'GOOGL', 'GE', 'KO', 'VZ', 'BA', 'NKE', 'CVX', 'AMZN', 'UNH', 'WM
data = {ticker: yf.download(ticker, start="2012-01-01", end="2014-07-01") for ticker in tickers}
```

```
import pandas as pd
from nltk.sentiment import SentimentIntensityAnalyzer

# Load news data
news_df = pd.read_csv('RedditNews.csv')
```

A function ‘clean_text’ is defined to clean and format news headlines data. It converts text to lowercase, use regular expressions to remove special characters, numbers and remove leading, trailing spaces to create clean text. Clean news headlines data is then stored in new data frame called news_df. Sentiment scores were calculated using SentimentIntensityAnalyzer for each day.

```
import re

def clean_text(text):
    """
    Function to clean the news text.
    - Lowercases the text.
    - Removes special characters and numbers.
    - Strips extra spaces.
    """
    # Lowercase the text
    text = text.lower()
    # Remove special characters and numbers
    text = re.sub(r'[^a-z\s]', '', text)
    # Remove extra spaces
    text = re.sub(r'\s+', ' ', text).strip()
    return text

# Clean the 'News' column
news_df['cleaned_news'] = news_df['News'].apply(clean_text)

# Display the first few rows of the dataframe after cleaning
news_df.head()
```

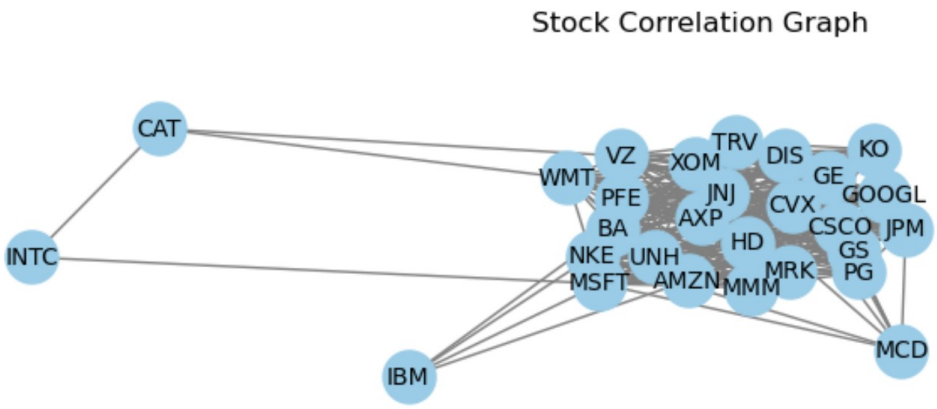
Date format is updated in accordance with timeseries data. Sentiment scores and stock price data are integrated to create combined features for each company and each day. Missing sentiment values are filled with 0. For each stock represented by data dictionary, ‘close’ price is normalized. Normalization scales the values to a range from 0 to 1. Each company’s stock price will be at a different range of values, hence normalizing all of them to a single scale is crucial preprocessing step. Features are created after normalization. Labels are generated based on increase or decrease in stock price. If the price is increased then label is 1, otherwise 0. Features and Labels are converted to NumPy arrays for further processing.

```
# Preprocessing
features = []
labels = []
for ticker, df in data.items():
    # Normalize close prices
    normalized_close = (df['Close'] - df['Close'].min()) / (df['Close'].max() - df['Close'].min())

    # Map sentiment scores to the stock data dates
    df_index_as_str = df.index.strftime('%Y-%m-%d')
    df['Sentiment'] = df_index_as_str.map(sentiment_dict)

    # Normalize sentiment scores
    normalized_sentiment = (df['Sentiment'] - df['Sentiment'].min()) / (df['Sentiment'].max() - df['Sentiment'].min())

    # Combine normalized close prices and sentiment scores
    combined_features = np.column_stack((normalized_close, normalized_sentiment))

    # Append to features list
    features.append(combined_features)

    # Generate labels (1 if price increase, 0 otherwise)
    labels.append(df['Close'].diff().apply(lambda x: 1 if x > 0 else 0).values)

# Convert to numpy array
features = np.array(features)
labels = np.array(labels)

# Print shapes for verification
print("Shape of features:", features.shape)
print("Shape of labels:", labels.shape)
```

```
Shape of features: (29, 626, 2)
Shape of labels: (29, 626)
```

Scatter plots were created for close price and sentiment scores to verify if there is any possible correlation between them. Stock close price is considered to calculate correlation matrix. Each element in the matrix represents the correlation between Close price of two stocks. 0.5 is defined as threshold to create an edge. If correlation between two stocks is above threshold, then we consider them as related and create an edge between them. A network graph is constructed based on these edges between different companies. The graph represents the spatial relationships between different companies. The list of edges in graph is converted to a PyTorch tensor. ‘networkx’ library is used to show the network in graphical representation.

```
# Correlation matrix and edge creation
close_prices = pd.DataFrame({ticker: df['Close'] for ticker, df in data.items()})
correlation_matrix = close_prices.corr()
print(correlation_matrix)
correlation_threshold = 0.5
edges = []
num_nodes = len(tickers)
for i in range(num_nodes):
    for j in range(i + 1, num_nodes):
        if abs(correlation_matrix.iloc[i, j]) > correlation_threshold:
            edges.append((i, j))
            edges.append((j, i))
edge_index = torch.tensor(edges, dtype=torch.long).t().contiguous()
```



This code is to set up the device (either CUDA GPU or CPU) for PyTorch computations. If a CUDA-compatible GPU is available, it will use the GPU for faster computations; otherwise, it falls back to the CPU. PyTorch Geometric is used to create graph data object. Features and Labels are converted to PyTorch tensors. x represents node features, edge_index represents edges and y represents labels. Graph_data object is created by combining x, edge_index and y which will act as input to GCN model.

```
import torch
from torch_geometric.data import Data

# Device setup
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Convert the features NumPy array to a PyTorch tensor
# The shape of features is (29, 626, 2)
x = torch.tensor(features, dtype=torch.float).to(device)

# Convert the NumPy array of labels to a PyTorch tensor
# The shape of labels is (29, 626)
y = torch.tensor(labels, dtype=torch.long).to(device)


# to match the new size of the feature matrix. Ensure edge_index is on the same device
edge_index = edge_index.to(device)

# Create the Data object with features, edge index, and labels
# The shape of 'x' and 'y' tensors should be compatible with model's architecture
graph_data = Data(x=x, edge_index=edge_index, y=y)

print(x, y, edge_index)
print(graph_data)
```

3.3 GCN-LSTM Model definition

PyTorch neural network model named GCNLSTM is defined with below input parameters.

num_features -> Number of features in input data.
gcn_hidden -> Number of hidden units in GCN layer.
lstm_hidden -> Number of hidden units in LSTM layer.
num_classes -> Number of output classes
num_nodes -> Number of nodes in the graph (number of companies)

GCNConv is a graph convolutional layer from PyTorch Geometric library. It takes num_features as input and outputs gcn_hidden features. It performs graph convolution using the given edge_indices and node features.

nn.LSTM is a Long Short Term Memory layer in PyTorch Geometric library. It takes output of GCNConv layer and process it through LSTM.

nn.Linear is fully connected layer that maps LSTM hidden states to the output classes.

The 'forward' method defines the forward pass of the model. Input x is reshaped to combine all days and nodes for GCN processing. After GCN, result is reshaped back to original structure. Reshaped data is processed through LSTM layer. LSTM output is reshaped for classification. Fully connected layer produces the final output of classification label for each company.

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch_geometric.nn import GCNConv

class GCNLSTM(nn.Module):
    def __init__(self, num_features, gcn_hidden, lstm_hidden, num_classes, num_nodes):
        super(GCNLSTM, self).__init__()
        self.num_nodes = num_nodes
        self.lstm_hidden = lstm_hidden  # Ensure this attribute is correctly set
        self.gcn = GCNConv(num_features, gcn_hidden)
        self.lstm = nn.LSTM(gcn_hidden, lstm_hidden, batch_first=True)
        self.fc = nn.Linear(lstm_hidden, num_classes)

    def forward(self, x, edge_index):
        # Process each day for all stocks through GCN
        # Reshape x to (-1, num_features) to combine all days and nodes for GCN processing
        x = x.view(-1, x.size(-1))
        x = F.relu(self.gcn(x, edge_index))

        # Reshape back to original structure: (num_nodes, num_days, gcn_hidden)
        x = x.view(self.num_nodes, -1, x.size(-1))

        # Process each stock's time series through LSTM
        lstm_out, (h_n, c_n) = self.lstm(x)

        # Reshape LSTM output for classification: (num_nodes * num_days, lstm_hidden)
        lstm_out = lstm_out.contiguous().view(-1, self.lstm_hidden)

        # Classifier on the output of LSTM
        out = self.fc(lstm_out)

        return out
```

3.4 GCN-LSTM Model Training

This snippet shows model training for 200 epochs. Number of features are defined as 2 since we are using close price and sentiment score for each stock. Number of classes are 2 since this is a classification task. Number of nodes is 29 , as we have 29 companies data. Number of days is 626 as we have 626 days data for training. GCNLSTM model is initialzed with these parameters. Adam optimizer is defined to update mdoel parameters while training. CrossEntrophyLoss function is defined as this is a classification task. Model is set up to train for 200 epochs. The loss is printed for each epoch to monitor the trainign process. The traniing loop is desinged for a single graph representing multiple nodes and time seires data for each node.

```
# Device setup
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Initialize the model with the correct number of arguments
num_features = 2  # Number of features: normalized close price and sentiment score
gcn_hidden = 64   # Size of GCN hidden layer
lstm_hidden = 32  # Size of LSTM hidden layer
num_classes = 2   # Number of classes for prediction (e.g., up or down)
num_nodes = 29    # Number of stocks
num_days = 626    # Number of days

model = GCNLSTM(num_features, gcn_hidden, lstm_hidden, num_classes, num_nodes).to(device)

# Define the optimizer
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Define the loss function
criterion = nn.CrossEntropyLoss()

# Training Loop
for epoch in range(200):
    model.train()
    optimizer.zero_grad()

    # Move your data to the same device as your model
    x = x.to(device)
    edge_index = edge_index.to(device)
    y = y.to(device)

    # Forward pass for the entire graph
    out = model(x, edge_index)  # Assuming you have a single graph

    # Reshape output and labels to be compatible with CrossEntropyLoss
    out = out.view(num_nodes * num_days, num_classes)
    y = y.view(-1)

    # Compute loss on the entire graph
    loss = criterion(out, y)
    loss.backward()
    optimizer.step()

    print(f'Epoch {epoch}, Loss: {loss.item()}')
```

3.5 GCN-LSTM Model Evaluation

Remiaining Test Data is processed same as trainign data and input tensors are created for validation. Model is set to evaluation mode. Validation loss is computed using loss function. The predicted class for each sample is obtained by selecting the class with highest probability. The number of correctly predicted samples and the validation accuracy is calculated.

```
# Validation Phase
model.eval()
with torch.no_grad():
    # Move your validation data to the same device as your model
    x_val = x_val.to(device)
    edge_index = edge_index.to(device)
    y_val = y_val.to(device)

    # Forward pass for the validation data
    val_out = model(x_val, edge_index)

    # Reshape val_out to match the shape of y_val for loss calculation
    val_out = val_out.view(-1, num_classes)

    # Compute the validation loss
    val_loss = criterion(val_out, y_val)

    # Get the predicted class (with highest probability)
    _, val_pred = torch.max(val_out, dim=1)

    # Calculate accuracy
    val_correct = int(val_pred.eq(y_val).sum().item())
    val_accuracy = val_correct / y_val.size(0)

print(f'Validation Loss: {val_loss.item()}, Validation Accuracy: {val_accuracy}')
```

3.6 DJIA Aggregation

Retrieve the model predictions for all stocks on current day. Aggregated label for DJIA is determined by a majority vote. If the sum of predicted labels is greater than half of num_nodes, the aggregated label is set to 1, otherwise set to 0. Daily aggregated labels are calculated for DJIA to compare against actual values. Accuracy, precision and other metrics are calculated using sklearn.metrics library.

```
daily_aggregated_predictions = []
num_days=505

for day in range(num_days):
    # Get predictions for all stocks for this day
    daily_predictions = val_out[day * num_nodes: (day + 1) * num_nodes]

    # Convert these predictions to labels (0 or 1)
    daily_labels = torch.argmax(daily_predictions, dim=1)

    # Aggregate these labels using a majority vote
    # If the sum of daily labels is greater than half of num_nodes, the aggregated label is 1, else 0
    aggregated_label = 1 if torch.sum(daily_labels).item() > num_nodes / 2 else 0

    daily_aggregated_predictions.append(aggregated_label)
```

3.7 Libraries to import for LSTM and ARIMA models

ARIMA Model Libraries

```
import warnings
warnings.filterwarnings("ignore")
from pylab import rcParams
import statsmodels.api as sm
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
%matplotlib inline
import numpy as np
import pandas as pd
import datetime as dt
import chart_studio.plotly as py
import plotly.graph_objs as go
import plotly.figure_factory as ff
import cufflinks as cf
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
init_notebook_mode(connected=True)
from pandasql import sqldf
pysqldf = lambda q: sqldf(q, globals())
np.set_printoptions(suppress=True)
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from IPython.display import Image
import yfinance as yf
import itertools
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error, mean_absolute_error
import matplotlib.pyplot as plt
```

LSTM Model Libraries

```
import pandas as pd
import numpy as np
from textblob import TextBlob
import re
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix, roc_auc_score, roc_curve
import matplotlib.pyplot as plt
```

```
# Importing the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
import yfinance as yf

from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, GRU
from tensorflow.keras.optimizers import SGD
from tensorflow.random import set_seed

from pandas_datareader.data import DataReader
from sklearn.metrics import mean_absolute_error, mean_squared_error

import warnings
warnings.simplefilter("ignore")

set_seed(455)
np.random.seed(455)
```

All import statements needs to be executed successfully for notebook to run without any issues.