National College of Ireland

Configuration Manual

MSc Research Project
Msc Data Analytics

Saif Ali Khan
Student ID: X22123296

School of Computing
National College of Ireland

Supervisor: **Cristina Hava Muntean**

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……SAIF ALI KHAN……

Student ID: ……x22123296@student.ncirl.ie……

Programme: ……Data Analytics…… **Year:** ……2023……..

Module: ……MSC Research Project……

Lecturer: …… Cristina Hava Muntean ……

Submission Due Date: ……14/12/2023……

Project Title: …… Investigating the Application of Natural Language Processing in Analysing Sentiment within Financial News ……

Word Count: 1300…… **Page Count:** ……12……

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.

Saif Ali Khan

Signature: ……

Date: ……14/12/2023……

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

Saif Ali Khan
Student ID: x22123296

1 Introduction

This document gives a clear demonstration of the hardware and software requirements along with the various steps taken to successfully implement the research

Project- Investigating the Application of Natural Language Processing in Analysing Sentiment within Financial News

2 System Configuration

2.1 Hardware Requirement

2.1.1 System OS: Windows 10 & Windows 8

2.1.2 Processor: i5

2.1.3 Ram : 8GB

2.2 Software Requirements

The project is implemented in Jupyter Notebook version 6.4.5 with the Python kernel version 3.9.7. Jupyter notebook can be installed using the Anaconda environment (fig. 1) which by default installs the python kernel as well

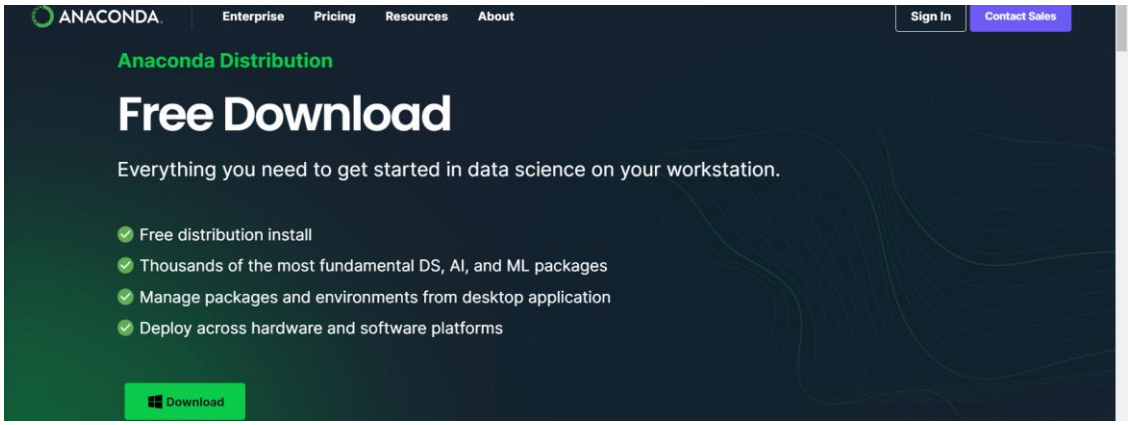

Fig 1 : Anaconda Environment

3 Project Implementation

The first step will be importing all the necessary libraries for the project

```
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
import re  # Add this import statement
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.utils import resample
```

```
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
```

Fig 2 : Installing Libraries

3.1 Data Collection:

The first dataset, obtained from Kaggle, contains 4,846 rows of data, each with a "Sentiment" label and accompanying "News Headline." This dataset, dubbed "FinancialPhraseBank," was created expressly to assess the attitudes conveyed in financial news headlines,. The second dataset, which has 5,834 rows of data, includes a "Sentiment" label as well as an associated "News Headline." This dataset was compiled in order to advance research in the field of financial sentiment analysis. It merges two distinct datasets, "FiQA" and "Financial PhraseBank," into a single, easy-to-use CSV file. This dataset, like the first, includes financial sentences and their related sentiment classifications.
Below is the dataset link

https://www.kaggle.com/datasets/ankurzing/sentiment-analysis-for-financial-news
https://www.kaggle.com/datasets/sbhatti/financial-sentiment-analysis

3.2 Pre –Processing:

pre-processing is a critical stage in the analysis of financial sentiment data. It comprises a variety of key tasks aimed at improving text data quality and utility. In the context of financial sentiment analysis, a systematic technique is utilized to ensure that the data is well-structured and suitable for future analysis. The pre-processing approach for both datasets used for financial sentiment analysis is the same. The purpose of the data cleaning procedure is to improve data quality and make it more suited for sentiment analysis. The following pre-processing stages are critical:

```
import re
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# Assuming df3 is your DataFrame with a 'Sentence' column
def preprocess_text(text):
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    text = text.lower()
    text = re.sub(r'\d+', '', text)
    text = ' '.join([word for word in text.split() if len(word) > 1])
    stop_words = set(stopwords.words('english'))
    text = ' '.join([word for word in word_tokenize(text) if word.lower() not in stop_words])
    tokens = word_tokenize(text)
    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(word) for word in tokens]
    return ' '.join(tokens)

# Apply the preprocessing function to the 'Sentence' column
df3['processed_text'] = df3['Sentence'].apply(preprocess_text)
```

Fig 3: Pre Processing

In fig 3 its shows, Pre-processing for financial sentiment analysis entails removing special characters, reducing text to lowercase, deleting numeric values, dealing with short sentences, and removing stop words.
Tokenization, lemmatization, expanding abbreviations/acronyms, employing sentiment lexicons, and correcting domain-specific data mistakes are additional processes for financial datasets.
Imbalanced data difficulties are addressed by oversampling or undersampling to ensure that sentiment classifications are distributed fairly.
The careful pre-processing ensures that the financial sentiment data is cleansed, formatted, and analytically ready, which is critical for effective analysis.
Python is the ideal programming language for these jobs, which increases productivity and consistency. After deleting duplicates in both the "Sentiment" and "Sentence" columns, the dataset with 10,688 entries is reduced to 6,051.

3.2.1 Removing Duplicates:

```
duplicates = df3.duplicated()

# Display rows that are duplicates
duplicate_rows = df3[duplicates]
print("Duplicate Rows except first occurrence:")
print(duplicate_rows)

# Count the number of duplicates
num_duplicates = duplicates.sum()
print(f"Number of duplicate rows: {num_duplicates}")
df3.drop_duplicates(subset=['Sentence'],keep='first',inplace=True)
df3.info()
```

fig 4: ensuring all duplicates are dropped

3.2.2 Imbalanced Data Handling:

```
from imblearn.over_sampling import SMOTE
```

```
# Step 1: Handling Imbalanced Data
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_tfidf, df3['Sentiment'])
```

Fig 5 : smote(Synthetic Minority Over-sampling Technique)

The fig 5 shows how to handle imbalanced data using SMOTE (Synthetic Minority Over-sampling Technique). SMOTE is a popular oversampling technique that creates synthetic samples of the minority class to balance the dataset.The code snippet first imports the SMOTE class from the imblearn library. Then, it creates a SMOTE object with the random_state parameter set to 42. This ensures that the synthetic samples are generated reproducibly.The X_tfidf variable contains the text data features, and the df3['Sentiment'] variable contains the sentiment labels.

3.2.3 Word Representations and Vectorization :

Word2 vec

```
from gensim.models import Word2Vec
```

```
sentences = [text.split() for text in df3['processed_text']]
word2vec_model = Word2Vec(sentences, window=5, min_count=1, workers=4)
word_vectors = word2vec_model.wv

similar_words = word_vectors.most_similar('finance', topn=5)
print(similar_words)

similar_wordssales = word_vectors.most_similar('profit', topn=5)
print(similar_wordssales)
```

Fig 6: word2Vec

3.2.4 TFIDF

```
#tfidf
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
tfidf_vectorizer = TfidfVectorizer()
X_tfidf = tfidf_vectorizer.fit_transform(df3['processed_text'])
```

```
print(X_tfidf)
```

Fig 8 TFIDF

In these figure libraries are shown to transform the text data into a TF-IDF matrix. The TF-IDF matrix is a numerical representation of the text data that can be used by machine learning models.

3.2.5 Dimensionality Reduction with PCA

```
from sklearn.decomposition import PCA

# Reduce dimensions for visualization
pca = PCA(n_components=2)
X_tfidf_pca = pca.fit_transform(X_tfidf.toarray())

# Plot the reduced-dimensional data
plt.figure(figsize=(10, 8))
scatter = plt.scatter(X_tfidf_pca[:, 0], X_tfidf_pca[:, 1], c='blue', marker='o', alpha=0.5)
```

Fig 9 PCA

In Fig 9, PCA is a dimensionality reduction technique that can be used to reduce the number of features in a dataset while preserving as much information as possible. This can be useful for machine learning tasks, such as classification and regression, as it can help to improve the performance of models by reducing the risk of overfitting.

Fig 9

3.2.5.1 Word Cloud

```
pip install seaborn wordcloud
```

```
import matplotlib.pyplot as plt
import seaborn as sns

# Visualization 1: Bar plot for Topic Distribution
plt.figure(figsize=(10, 6))
sns.barplot(x=topic_distribution.index, y=topic_distribution.values, palette='viridis')
plt.title('Topic Distribution')
plt.xlabel('Topic')
plt.ylabel('Number of Documents')
plt.show()

# Visualization 2: Word Clouds for Top Words in Each Topic
from wordcloud import WordCloud

def display_wordcloud_for_topic(topic_words, topic_idx):
    wordcloud = WordCloud(width=800, height=400, background_color='white').generate(' '.join(topic_words))
    plt.figure(figsize=(10, 6))
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')
    plt.title(f'Topic {topic_idx + 1} - Top Words')
    plt.show()
```

Fig 10. WordCloud

In Fig 10, A topic model is a statistical model that can be used to identify the latent topics or themes in a collection of text documents. The topic distribution for a topic model shows the distribution of topics across the documents in the collection.

3.2.6 Advanced Techniques and Model Development:

3.2.6.1 Random forest

```
#Random forest

from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, precision_score, recall_score, f1_score

# Step 1: Handling Imbalanced Data
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_tfidf, df3['Sentiment'])

# Step 2: Split the Resampled Data
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)

# Step 3: Model Selection and Training
rf_classifier = RandomForestClassifier()
rf_classifier.fit(X_train, y_train)

# Step 4: Model Evaluation
y_pred = rf_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# Print the results
print(f"Accuracy: {accuracy:.4f}")
print("Confusion Matrix:")
print(conf_matrix)
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
```

Fig 11 Random Forest.

In Fig 11, A powerful ensemble machine learning technique that employs numerous decision trees for learning. Its value in sentiment analysis stems from its interpretability and the potential to enhance performance through hyperparameter optimization.

3.2.6.2 SVM:

```
#SVM

from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, precision_score, recall_score, f1_score

# Assuming you have already preprocessed your text and have X_tfidf and df3['Sentiment']

# Step 1: Handling Imbalanced Data
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_tfidf, df3['Sentiment'])

# Step 2: Split the Resampled Data
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)

# Step 3: Model Selection and Training (SVM Classifier)
svm_classifier = SVC(kernel='linear', random_state=42)
svm_classifier.fit(X_train, y_train)

# Step 4: Model Evaluation
y_pred = svm_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# Print the results
print(f"Accuracy: {accuracy:.4f}")
print("Confusion Matrix:")
print(conf_matrix)
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

Fig 12. SVM

In Fig 12, A robust algorithm employed in sentiment analysis, SVM effectively distinguishes sentiment types by constructing a hyperplane that optimally separates sentiment classes. Its ability to generalize sentiment patterns contributes to its effectiveness in this domain.

3.2.6.3 LSTM:

```
#lstm
from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Assuming X_tfidf is your preprocessed and vectorized text data
X = df3['processed_text']
y = df3['Sentiment']

# Encode labels to numerical values
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# Tokenize the text
max_words = 5000  # Set the maximum number of words to consider
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(X)

# Convert text data to sequences
X_sequences = tokenizer.texts_to_sequences(X)

# Pad sequences to ensure consistent length for input to the LSTM
X_padded = pad_sequences(X_sequences)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_padded, y_encoded, test_size=0.2, random_state=42)

# Define the LSTM model
embedding_size = 128
lstm_units = 100

model = Sequential()
model.add(Embedding(input_dim=max_words, output_dim=embedding_size, input_length=X_padded.shape[1]))
model.add(LSTM(units=lstm_units))
model.add(Dense(1, activation='sigmoid'))  # Assuming binary classification, change for multiclass
```

Fig 13 LSTM

In Fig 13, A recurrent neural network architecture particularly adept at processing sequential data, Long Short-Term Memory (LSTM) shines in capturing long-range dependencies and subtle sentiment nuances, making it a valuable tool for sentiment analysis.

3.2.6.4 LDA :

```
from sklearn.decomposition import LatentDirichletAllocation
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd
from gensim.models import CoherenceModel
from gensim.corpora import Dictionary

# Assuming df3['processed_text'] contains your preprocessed text data

# Step 1: Create a document-term matrix using CountVectorizer
vectorizer = CountVectorizer(max_df=0.95, min_df=2, stop_words='english')
dtm = vectorizer.fit_transform(df3['processed_text'])

# Step 2: Apply LDA
num_topics = 5  # You can adjust the number of topics
lda = LatentDirichletAllocation(n_components=num_topics, random_state=42)
lda_output = lda.fit_transform(dtm)

# Step 3: Display the topics and their top words
feature_names = vectorizer.get_feature_names_out()

# Helper function to display top words for each topic
def display_topics(model, feature_names, no_top_words):
    topic_list = []
    for topic_idx, topic in enumerate(model.components_):
        topic_words = [feature_names[i] for i in topic.argsort()[:-no_top_words - 1:-1]]
        topic_list.append((topic_idx, topic_words))
    return topic_list

# Display the topics with the top 10 words
topics = display_topics(lda, feature_names, 10)
for topic in topics:
    print(f"Topic {topic[0]}: {' '.join(topic[1])}")

# Step 4: Assign topics to documents
df3['topic'] = lda_output.argmax(axis=1)

# Step 5: Analyze the distribution of topics
topic_distribution = df3['topic'].value_counts().sort_index()
print("\nTopic Distribution:")
```

Fig 14 LDA

In Fig 14, A statistical technique for identifying latent topics or themes within a collection of documents. Its ability to uncover these underlying structures is evaluated using metrics like perplexity and coherence score.

3.2.7 Model Evaluation

Precise, recall, and accuracy metrics serve as benchmarks for evaluating the performance of evaluation

4 References

• Baker, M. &. (2007). Investor sentiment and stock returns. Journal of Finance, 62(3), 1643-1681.

Ahmad, K. (2006). *Multi-lingual sentiment analysis of financial news streams*. Retrieved from https://www.researchgate.net/profile/Khurshid-Ahmad-5/publication/322956319_Multi-lingual_sentiment_analysis_of_financial_news_streams/links/5ab783b0aca2722b97ce7461/Multi-lingual-sentiment-analysis-of-financial-news-streams.pdf

Alessia, D. F. (2015). *Approaches, tools and applications for sentiment analysis implementation. International Journal of Computer Applications,*. Retrieved from

https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=d709747c589bca62d9ee85752fb3a8ec899cac20

Chan, S. a. (2017). *Sentiment analysis in financial texts. Decision Support Systems,* . Retrieved from https://www.sciencedirect.com/science/article/pii/S0167923616301828?casa_token=Y-yYJNHqZTAAAAAA:ql_OTlQjwvlqkj8SHAGSba54Es2mRljOALNKZYKET9gKu0mSMe-4NQ9Hx8uMTuo8U10u4ZWxAA

Day, M. Y. (2016). *Deep learning for financial sentiment analysis on finance news providers*. Retrieved from https://ieeexplore.ieee.org/abstract/document/7752381

Kilimci, Z. (2020). *Financial sentiment analysis with Deep Ensemble Models (DEMs) for stock market prediction. Journal of the Faculty of Engineering and Architecture of Gazi University, 35(2)*. Retrieved from https://www.researchgate.net/profile/Zeynep-Kilimci/publication/338124910_Borsa_tahmini_icin_Derin_Topluluk_Modellleri_DTM_ile_finansal_duygu_analizi/links/5e0b12ba4585159aa4a70b0f/Borsa-tahmini-icin-Derin-Topluluk-Modellleri-DTM-ile-finansal-duygu-analiz

Qian, C. M. (2022). *Understanding public opinions on social media for financial sentiment analysis using AI-based techniques.* . Retrieved from https://www.sciencedirect.com/science/article/pii/S0306457322001996?casa_token=dqvXENLI15wAAAAA:Yyap6ws2PVzpqSdrSzA2OcGeMcDZqDINrBG3gOaKXL_J77IJM70FrPmO8y8UnmUTdEOo_H4j6w

Sohangir, S. W. (2018). Big Data: Deep Learning for financial sentiment analysis. Journal of Big Data, 5(1),. 25.

Xu, Y. a. (2019). *Stock prediction using deep learning and sentiment analysis .In 2019 IEEE international conference on big data (big data)*. Retrieved from https://ieeexplore.ieee.org/abstract/document/9006342