Pipeline

```
1) Removing HTML tagas and URLs, Punctiation*, Replacing emoticons*.
2) Tokenization
3) Removing Stop Words
4) Splitting data: Training, Validation, Test
5) TF-IDF Calculation
```

Defining working environment.

In [1]:

```python
# Multilayer perceptron working environment.
# Getting ready the work environment. Importing libraries and modules:
import time
import pandas as pd
import re
import nltk
import torch
import torch.nn as nn
import numpy as np
import string
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_recall_fscore_support, classification_report, roc_curve, roc_auc_scor
from collections import Counter
from bs4 import BeautifulSoup
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

#===========          Extra tools for the statistic analysis            ======================
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()
#----------------------------------------------------------------------

stop_words = stopwords.words('english')
tfidf_vectorizer = TfidfVectorizer()
vectorizer = CountVectorizer()
```

In [2]:

```python
# Getting ready the work environment. Importing libraries and modules:
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import re
import nltk
import string

from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
from sklearn.metrics import classification_report, roc_curve, roc_auc_score, confusion_matrix
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from collections import Counter
from bs4 import BeautifulSoup


#===========          Extra tools for the statistic analysis            ======================
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()
#----------------------------------------------------------------------

stop_words = stopwords.words('english')
tfidf_vectorizer = TfidfVectorizer(stop_words='english')
vectorizer = CountVectorizer()
```

Load dataset

In [3]:

```
# Importing dataset from the hugging face
from datasets import load_dataset

dataset1 = load_dataset('financial_phrasebank', 'sentences_50agree')
dataset2 = load_dataset('financial_phrasebank', 'sentences_66agree')
dataset3 = load_dataset('financial_phrasebank', 'sentences_75agree')
```

```
Found cached dataset financial_phrasebank (C:/Users/nonox/.cache/huggingface/datasets/financial_phrasebank/sen
tences_50agree/1.0.0/550bde12e6c30e2674da973a55f57edde5181d53f5a5a34c1531c53f93b7e141)

  0%|          | 0/1 [00:00<?, ?it/s]

Found cached dataset financial_phrasebank (C:/Users/nonox/.cache/huggingface/datasets/financial_phrasebank/sen
tences_66agree/1.0.0/550bde12e6c30e2674da973a55f57edde5181d53f5a5a34c1531c53f93b7e141)

  0%|          | 0/1 [00:00<?, ?it/s]

Found cached dataset financial_phrasebank (C:/Users/nonox/.cache/huggingface/datasets/financial_phrasebank/sen
tences_75agree/1.0.0/550bde12e6c30e2674da973a55f57edde5181d53f5a5a34c1531c53f93b7e141)

  0%|          | 0/1 [00:00<?, ?it/s]
```

In [4]:

```
# Checking dataset
print(dataset1)
```

```
DatasetDict({
    train: Dataset({
        features: ['sentence', 'label'],
        num_rows: 4846
    })
})
```

In [5]:

```
# Transforming the data set into a more friendly frame (tables)
df50 = pd.DataFrame(dataset1['train'])
df66 = pd.DataFrame(dataset2['train'])
df75 = pd.DataFrame(dataset3['train'])
```

In [6]:

```python
# Checking data
print(df50)
print("\n")
print(df66)
print("\n")
print(df75)
```

```
                                               sentence  label
0     According to Gran , the company has no plans t...      1
1     Technopolis plans to develop in stages an area...      1
2     The international electronic industry company ...      0
3     With the new production plant the company woul...      2
4     According to the company 's updated strategy f...      2
...                                                 ...    ...
4841  LONDON MarketWatch -- Share prices ended lower...      0
4842  Rinkuskiai 's beer sales fell by 6.5 per cent ...      1
4843  Operating profit fell to EUR 35.4 mn from EUR ...      0
4844  Net sales of the Paper segment decreased to EU...      0
4845  Sales in Finland decreased by 10.5 % in Januar...      0

[4846 rows x 2 columns]


                                               sentence  label
0     According to Gran , the company has no plans t...      1
1     Technopolis plans to develop in stages an area...      1
2     With the new production plant the company woul...      2
3     According to the company 's updated strategy f...      2
4     For the last quarter of 2010 , Componenta 's n...      2
...                                                 ...    ...
4212  HELSINKI Thomson Financial - Shares in Cargote...      0
4213  LONDON MarketWatch -- Share prices ended lower...      0
4214  Rinkuskiai 's beer sales fell by 6.5 per cent ...      1
4215  Operating profit fell to EUR 35.4 mn from EUR ...      0
4216  Sales in Finland decreased by 10.5 % in Januar...      0

[4217 rows x 2 columns]


                                               sentence  label
0     According to Gran , the company has no plans t...      1
1     With the new production plant the company woul...      2
2     For the last quarter of 2010 , Componenta 's n...      2
3     In the third quarter of 2010 , net sales incre...      2
4     Operating profit rose to EUR 13.1 mn from EUR ...      2
...                                                 ...    ...
3448  Operating result for the 12-month period decre...      0
3449  HELSINKI Thomson Financial - Shares in Cargote...      0
3450  LONDON MarketWatch -- Share prices ended lower...      0
3451  Operating profit fell to EUR 35.4 mn from EUR ...      0
3452  Sales in Finland decreased by 10.5 % in Januar...      0

[3453 rows x 2 columns]
```

In [7]:

```python
# Checking the data we will preproccess
print(df50['sentence'])
```

```
0       According to Gran , the company has no plans t...
1       Technopolis plans to develop in stages an area...
2       The international electronic industry company ...
3       With the new production plant the company woul...
4       According to the company 's updated strategy f...
                              ...
4841    LONDON MarketWatch -- Share prices ended lower...
4842    Rinkuskiai 's beer sales fell by 6.5 per cent ...
4843    Operating profit fell to EUR 35.4 mn from EUR ...
4844    Net sales of the Paper segment decreased to EU...
4845    Sales in Finland decreased by 10.5 % in Januar...
Name: sentence, Length: 4846, dtype: object
```

In [8]:

```python
# Checking data balance
sentiment_counts = df50['label'].value_counts()
print('Sentiment distribution: 2-Positive, 1-Neutral, 0-Negative, ')
print(sentiment_counts)
```

```
Sentiment distribution: 2-Positive, 1-Neutral, 0-Negative,
1    2879
2    1363
0     604
Name: label, dtype: int64
```

) Removing HTML tags and URLs, punctuction, lowercasing

In [9]:
```
# Function to remove HTML tags:
def remove_html(text):
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

#Sources:
#https://stackoverflow.com/questions/328356/extracting-text-from-html-file-using-python?newreg=aa9f4dc4aea34
#https://beautiful-soup-4.readthedocs.io/en/latest/
#https://www.datacamp.com/tutorial/web-scraping-using-python
#https://www.geeksforgeeks.org/how-to-write-the-output-to-html-file-with-python-beautifulsoup/
```

In [10]:
```
def remove_urls(text):
    return re.sub(r'https?://\S+|www\.\S+', '', text)

#Source: https://www.geeksforgeeks.org/remove-urls-from-string-in-python/
```

In [11]:
```
 def remove_punctuation(text):
    return text.translate(str.maketrans('', '', string.punctuation))

#Source: https://stackoverflow.com/questions/34293875/how-to-remove-punctuation-marks-from-a-string-in-pytho
```

In [12]:
```
# Function to remove HTML tags:
def remove_html(text):
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

#Sources:
#https://stackoverflow.com/questions/328356/extracting-text-from-html-file-using-python?newreg=aa9f4dc4aea34
#https://beautiful-soup-4.readthedocs.io/en/latest/
#https://www.datacamp.com/tutorial/web-scraping-using-python
#https://www.geeksforgeeks.org/how-to-write-the-output-to-html-file-with-python-beautifulsoup/
```

In [13]:
```
# Function to put together all the previous functions:
def preprocess_1(text):
    text = remove_html(text)
    text = remove_urls(text)
    text = remove_punctuation(text)
    text = text.lower()
    return text

df50['sentence_preprocessed_1'] = df50['sentence'].apply(preprocess_1)
```

```
C:\Users\nonox\AppData\Local\Temp\ipykernel_14764\2872886666.py:3: MarkupResemblesLocatorWarning: The input lo
oks more like a filename than markup. You may want to open this file and pass the filehandle into Beautiful So
up.
  soup = BeautifulSoup(text, "html.parser")
```

) Tokenization

In [14]:
```
# Function to tokenize and convert to lower case the text in review column
def tokenize(text):
    tokens = re.findall(r'\b\w+\b', text)
    return tokens

#Tokenization
df50['token'] = df50['sentence_preprocessed_1'].apply(tokenize)
```

) Removing Stop Words

In [15]:
```
# Function to remove stop words from the tokenized review column
def remove_stopwords(tokens):
    filtered_tokens = [word for word in tokens if word not in stop_words]
    return filtered_tokens

#Remove stopwords
df50['token'] = df50['token'].apply(remove_stopwords)
```

) Some statistics

In [16]:

```python
# Calculating the total tokens for each review
df50['token_count'] = df50['token'].apply(lambda x: len(x) if isinstance(x, list) else 0)

# Dispersion and central tendency measurements
statistics = df50.groupby('label')['token_count'].agg(['min', 'max', 'mean', 'var', 'std'])

# Avg words per review:
avg_words = df50['token'].apply(len).mean()

#Print the statistics
print("Statistics by Label: ")
print('\n')
print(statistics)
print('\n')
print('\n')
print('Average Words: ', f"{avg_words:.0f}")

#Resources:
#https://www.geeksforgeeks.org/pandas-groupby-one-column-and-get-mean-min-and-max-values/
#https://www.kaggle.com/code/akshaysehgal/ultimate-guide-to-pandas-groupby-aggregate
```

```
Statistics by Label:


       min  max       mean        var       std
label
0        2   34  13.877483  37.746159  6.143790
1        0   46  12.516151  37.739055  6.143212
2        2   35  14.318415  40.969022  6.400705




Average Words:  13
```

In [17]:

```python
# Iterating through the list of lists(each row) to create a new list with all the tokens
def word_freq(list_of_list):
    single_list = [item for sublist in list_of_list for item in sublist]
    token_freq = Counter(single_list)
    return token_freq

# Counting the frequency for each word.
word_frequency = word_freq(df50['token'])
print(word_frequency)

#Sources: https://www.datacamp.com/tutorial/pandas-apply
```

```
Counter({'eur': 1015, 'company': 848, 'said': 544, 'mn': 515, 'finnish': 512, 'sales': 453, 'million': 440,
'net': 412, 'profit': 409, 'finland': 337, 'group': 320, 'operating': 299, '2009': 297, 'mln': 288, '2008':
283, 'year': 273, 'new': 267, 'business': 265, 'period': 264, '2007': 243, 'oyj': 241, 'quarter': 238, '201
0': 238, 'share': 237, 'also': 224, 'services': 223, 'market': 217, 'shares': 198, 'first': 193, '2006': 17
3, 'euro': 164, 'helsinki': 163, 'loss': 153, 'compared': 149, 'today': 149, 'operations': 149, 'contract':
142, 'nokia': 139, 'total': 137, 'financial': 134, 'mobile': 134, 'production': 130, 'products': 130, 'per':
129, 'corporation': 129, 'bank': 126, 'according': 123, 'percent': 123, 'companies': 122, 'hel': 121, 'techn
ology': 120, 'corresponding': 119, 'plant': 118, 'solutions': 117, 'service': 116, 'increased': 109, 'constr
uction': 109, 'capital': 109, 'agreement': 106, 'investment': 105, '2005': 104, 'well': 104, 'increase': 10
3, 'rose': 102, 'customers': 102, 'pct': 99, 'value': 98, 'order': 97, 'us': 97, 'oy': 97, 'stock': 94, 'wou
ld': 92, 'board': 91, 'omx': 91, 'unit': 90, 'development': 90, 'one': 90, '1': 89, 'building': 89, 'part':
88, 'management': 88, 'industry': 86, 'russia': 85, 'two': 85, 'earlier': 83, 'last': 83, 'result': 83, 'pap
er': 83, 'equipment': 83, 'second': 82, '10': 81, 'project': 81, 'expected': 79, 'decreased': 79, 'ceo': 78,
'employees': 77, 'maker': 77, 'signed': 76, 'deal': 76, 'media': 75, 'plc': 75, 'end': 75, 'software': 75,
'2011': 74, 'price': 74, '20': 73, 'third': 73, '3': 73, '5': 73, 'systems': 73, 'usd': 72, 'announced': 71,
'markets': 70, 'number': 69, 'exchange': 69, 'including': 69, 'global': 69, 'annual': 69, 'approximately': 6
9, 'area': 67, '15': 67, 'news': 67, 'data': 67, 'system': 67, 'people': 67, 'growth': 65, 'billion': 65, 'a
cquisition': 65, 'september': 64, 'line': 63, 'ltd': 63, '2': 63, 'euros': 62, 'months': 62, 'report': 61,
'4': 61, 'countries': 61, 'based': 61, 'may': 61, 'eur0': 60, '30': 60, 'network': 59, 'information': 59, 'u
```

In [18]:

```python
# Unique words
unique_words = len(word_frequency.keys())
print('Unique_words: ',f'{unique_words}')
```

```
Unique_words:  11105
```

In [19]:

```
# Repeated words in each label
positive_words = Counter()
neutral_words = Counter()
negative_words = Counter()

for index, row in df50.iterrows():
    words = row['token']
    label = row['label']
    if label == 1:
        positive_words.update(words)
    elif label == 2:
        negative_words.update(words)
    else:
        neutral_words.update(words)

#Resources:
#https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.iterrows.html
#https://www.kaggle.com/code/juicykn/imdb-movie-list-analysis-in-python-and-sql
```

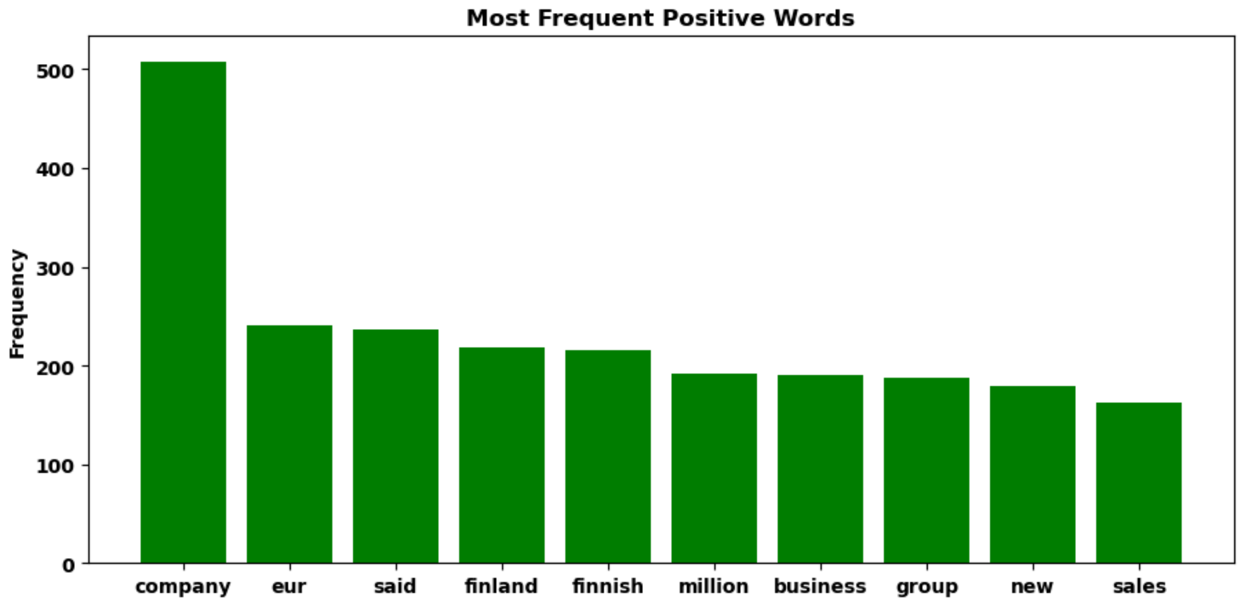
In [20]:

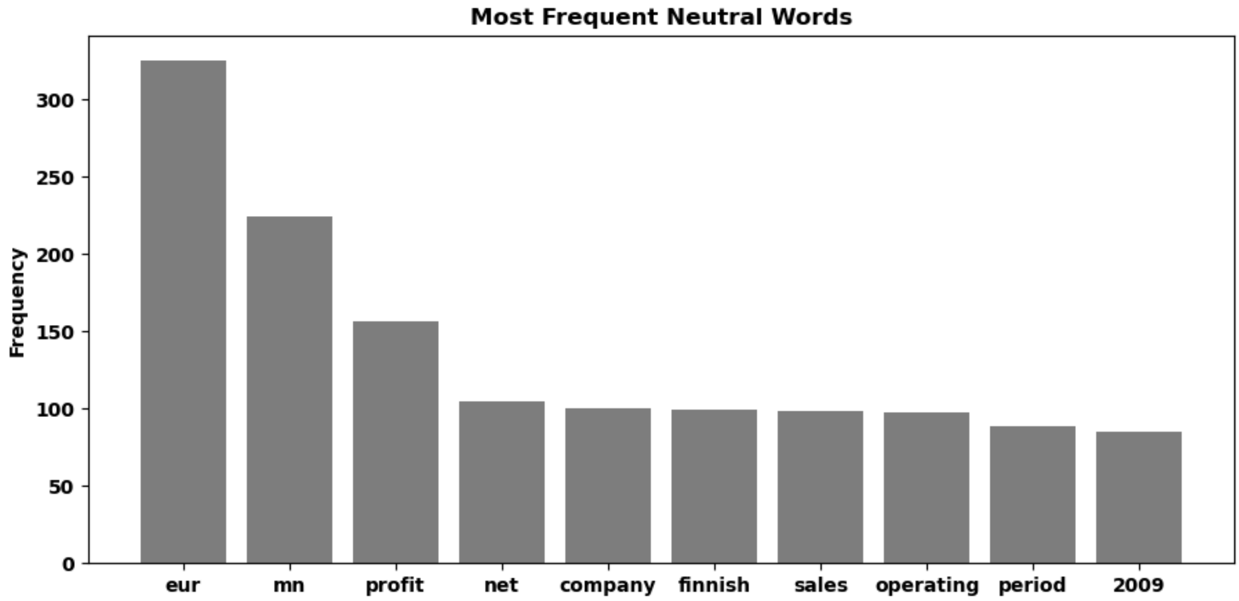
```
# Most repeated words in each label
top_positive_words = positive_words.most_common(10)
top_negative_words = negative_words.most_common(10)
top_neutral_words = neutral_words.most_common(10)

print('Positive: ', top_positive_words)
print('\n')
print('Negative: ', top_negative_words)
print('\n')
print('Neutral: ', top_neutral_words)
```

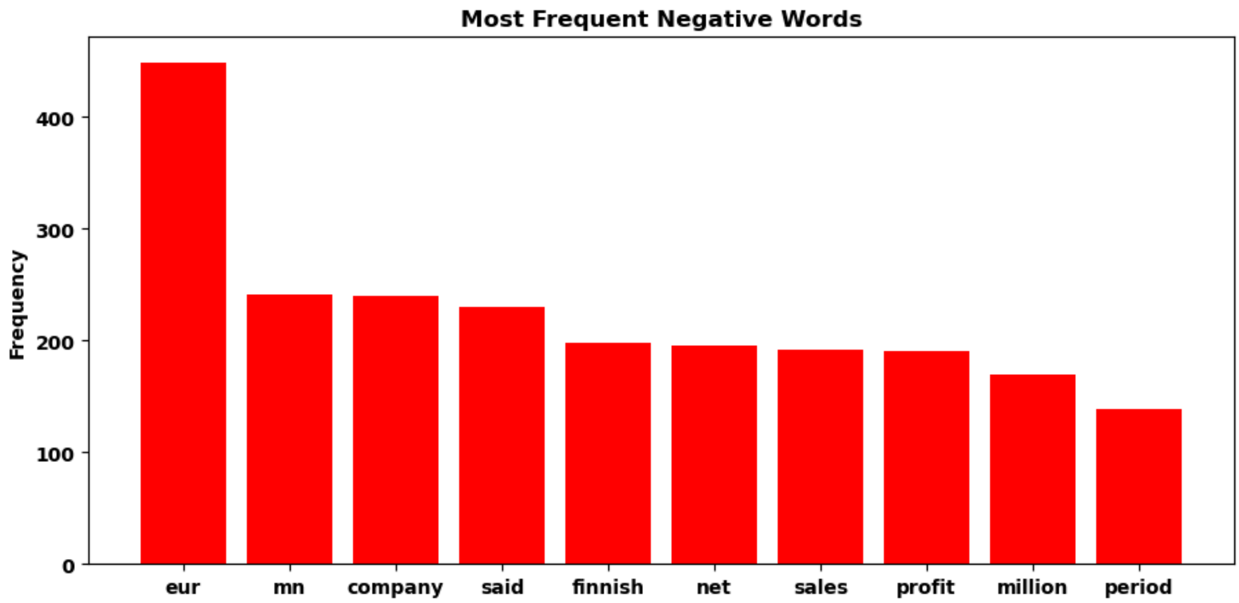
```
Positive:  [('company', 508), ('eur', 241), ('said', 237), ('finland', 219), ('finnish', 215), ('million', 19
2), ('business', 190), ('group', 187), ('new', 179), ('sales', 163)]


Negative:  [('eur', 449), ('mn', 241), ('company', 240), ('said', 230), ('finnish', 198), ('net', 196), ('sale
s', 192), ('profit', 191), ('million', 170), ('period', 139)]


Neutral:  [('eur', 325), ('mn', 224), ('profit', 156), ('net', 104), ('company', 100), ('finnish', 99), ('sale
s', 98), ('operating', 97), ('period', 88), ('2009', 85)]
```

In [21]:

```
# Spliting the tupple we got earlier
positive_words, positive_counts = zip(*top_positive_words)
negative_words, negative_counts = zip(*top_negative_words)
neutral_words, neutral_counts = zip(*top_neutral_words)

# Charts----------------------------------------------------
fig, axs = plt.subplots(3,1,figsize=(10,15))

# Positive words plot
axs[0].bar(positive_words, positive_counts, color='green')
axs[0].set_title('Most Frequent Positive Words')
axs[0].set_ylabel('Frequency')

# Negative words plot
axs[1].bar(neutral_words, neutral_counts, color='grey')
axs[1].set_title('Most Frequent Neutral Words')
axs[1].set_ylabel('Frequency')

# Neutral words plot
axs[2].bar(negative_words, negative_counts, color='red')
axs[2].set_title('Most Frequent Negative Words')
axs[2].set_ylabel('Frequency')

# Space between charts
plt.tight_layout(pad=4.0)
plt.show()

#Resources:
# https://realpython.com/python-zip-function/#using-zip-in-python
# https://matplotlib.org/stable/index.html
```


Most Frequent Positive Words
Frequency
0
100
200
300
400
500
company
eur
said
finland
finnish
million
business
group
new
sales


Most Frequent Neutral Words
Frequency
0
50
100
150
200
250
300
eur
mn
profit
net
company
finnish
sales
operating
period
2009


Most Frequent Negative Words
Frequency
0
100
200
300
400
eur
mn
company
said
finnish
net
sales
profit
million
period

In [22]:
```
# ) Spliting data
```

In [23]:
```
# Splitting data into train 70%, validation 15%, test 15%

X_train_val, X_test, y_train_val, y_test = train_test_split(df50['sentence_preprocessed_1'], df50['label'],

X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.15, random_state=42
```

In [ ]:
```

```

In [24]:
```
# Feature extraction: Transforming data into TF-IDF features.
X_train = tfidf_vectorizer.fit_transform(X_train)
X_val = tfidf_vectorizer.transform(X_val)
X_test = tfidf_vectorizer.transform(X_test)
```

In [25]:
```
print(X_train.shape)  # Should output (number_of_samples, 33154)
print(X_test.shape)   # Should also output (number_of_samples, 33154)
print(X_val.shape)
```

```
(3501, 9185)
(727, 9185)
(618, 9185)
```

In [26]:
```
#Turning sparse matrix into dense
X_train = X_train.toarray()
X_val = X_val.toarray()
X_test = X_test.toarray()

#Turning into PyTorch tensors
X_train = torch.tensor(X_train, dtype=torch.float32)
X_val = torch.tensor(X_val, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_train = torch.tensor(y_train.values, dtype=torch.float32)
y_val = torch.tensor(y_val.values, dtype=torch.float32)
y_test = torch.tensor(y_test.values, dtype=torch.float32)

#Resources:
# https://pytorch.org/docs/stable/tensors.html
```

In [38]:
```
class SimpleMLPmodel(nn.Module):
    def __init__(self):
        super(SimpleMLPmodel, self).__init__()
        self.fc = nn.Linear(9185, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.fc(x)
        x = self.sigmoid(x)
        return x

# Model set on ev mode.
model.eval()

# Single forward pass
with torch.no_grad():
    outputs = model(X_test)

# threshold to classify pb
threshold = 0.5
predicted_labels = (outputs > threshold).float()  # Convert probabilities to 0 or 1 based on the threshold

# Number of correct predictions
correct_predictions = (predicted_labels.squeeze() == y_test).float().sum()

# Accuracy
accuracy = correct_predictions / y_test.shape[0]
print(f'Accuracy: {accuracy.item():.2f}')
```

```
Accuracy: 0.48
```

In [ ]:

```python
# Multilayer perceptron working environment.
# Getting ready the work environment. Importing libraries and modules:
import time
import pandas as pd
import re
import nltk
import torch
import torch.nn as nn
import numpy as np
import string
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_recall_fscore_support, classification_report, roc_curve, roc_auc_scor
from collections import Counter
from bs4 import BeautifulSoup
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

#===========         Extra tools for the statistic analysis             ======================
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()
#----------------------------------------------------------------------

stop_words = stopwords.words('english')
tfidf_vectorizer = TfidfVectorizer()
vectorizer = CountVectorizer()
```

In [ ]:

```python
# Getting ready the work environment. Importing libraries and modules:
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import re
import nltk
import string

from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
from sklearn.metrics import classification_report, roc_curve, roc_auc_score, confusion_matrix
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from collections import Counter
from bs4 import BeautifulSoup


#===========         Extra tools for the statistic analysis             ======================
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()
#----------------------------------------------------------------------

stop_words = stopwords.words('english')
tfidf_vectorizer = TfidfVectorizer(stop_words='english')
vectorizer = CountVectorizer()
```

In [ ]:

```python
# Importing dataset from the hugging face
from datasets import load_dataset

dataset1 = load_dataset('financial_phrasebank', 'sentences_50agree')
dataset2 = load_dataset('financial_phrasebank', 'sentences_66agree')
dataset3 = load_dataset('financial_phrasebank', 'sentences_75agree')
```

In [ ]:

```python
# Checking dataset
print(dataset1)
```

In [ ]:

```python
# Transforming the data set into a more friendly frame (tables)
df50 = pd.DataFrame(dataset1['train'])
df66 = pd.DataFrame(dataset2['train'])
df75 = pd.DataFrame(dataset3['train'])
```

In [ ]:
```
# Checking data
print(df50)
print("\n")
print(df66)
print("\n")
print(df75)
```

In [ ]:
```
# Checking the data we will preproccess
print(df50['sentence'])
```

In [ ]:
```
# Checking data balance
sentiment_counts = df50['label'].value_counts()
print('Sentiment distribution: 2-Positive, 1-Neutral, 0-Negative, ')
print(sentiment_counts)
```

In [ ]:
```
# Function to remove HTML tags:
def remove_html(text):
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

#Sources:
#https://stackoverflow.com/questions/328356/extracting-text-from-html-file-using-python?newreg=aa9f4dc4aea3
#https://beautiful-soup-4.readthedocs.io/en/latest/
#https://www.datacamp.com/tutorial/web-scraping-using-python
#https://www.geeksforgeeks.org/how-to-write-the-output-to-html-file-with-python-beautifulsoup/
```

In [ ]:
```
def remove_urls(text):
    return re.sub(r'https?://\S+|www\.\S+', '', text)

#Source: https://www.geeksforgeeks.org/remove-urls-from-string-in-python
```

In [ ]:
```
 def remove_punctuation(text):
    return text.translate(str.maketrans('', '', string.punctuation))

#Source: https://stackoverflow.com/questions/34293875/how-to-remove-punctuation-marks-from-a-string-in-pyth
```

In [ ]:
```
# Function to remove HTML tags:
def remove_html(text):
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

#Sources:
#https://stackoverflow.com/questions/328356/extracting-text-from-html-file-using-python?newreg=aa9f4dc4aea3
#https://beautiful-soup-4.readthedocs.io/en/latest/
#https://www.datacamp.com/tutorial/web-scraping-using-python
#https://www.geeksforgeeks.org/how-to-write-the-output-to-html-file-with-python-beautifulsoup/
```

In [ ]:
```
# Function to put together all the previous functions:
def preprocess_1(text):
    text = remove_html(text)
    text = remove_urls(text)
    text = remove_punctuation(text)
    text = text.lower()
    return text

df50['sentence_preprocessed_1'] = df50['sentence'].apply(preprocess_1)
```

In [ ]:
```
# Function to tokenize and convert to lower case the text in review column
def tokenize(text):
    tokens = re.findall(r'\b\w+\b', text)
    return tokens

#Tokenization
df50['token'] = df50['sentence_preprocessed_1'].apply(tokenize)
```

In [ ]:

```
# Function to remove stop words from the tokenized review column
def remove_stopwords(tokens):
    filtered_tokens = [word for word in tokens if word not in stop_words]
    return filtered_tokens

#Remove stopwords
df50['token'] = df50['token'].apply(remove_stopwords)
```

In [ ]:

```
# Calculating the total tokens for each review
df50['token_count'] = df50['token'].apply(lambda x: len(x) if isinstance(x, list) else 0)

# Dispersion and central tendency measurements
statistics = df50.groupby('label')['token_count'].agg(['min', 'max', 'mean', 'var', 'std'])

# Avg words per review:
avg_words = df50['token'].apply(len).mean()

#Print the statistics
print("Statistics by Label: ")
print('\n')
print(statistics)
print('\n')
print('\n')
print('Average Words: ', f"{avg_words:.0f}")

#Resources:
#https://www.geeksforgeeks.org/pandas-groupby-one-column-and-get-mean-min-and-max-values/
#https://www.kaggle.com/code/akshaysehgal/ultimate-guide-to-pandas-groupby-aggregate
```

In [ ]:

```
# Iterating through the list of lists(each row) to create a new list with all the tokens
def word_freq(list_of_list):
    single_list = [item for sublist in list_of_list for item in sublist]
    token_freq = Counter(single_list)
    return token_freq

# Counting the frequency for each word.
word_frequency = word_freq(df50['token'])
print(word_frequency)

#Sources: https://www.datacamp.com/tutorial/pandas-apply
```

In [ ]:

```
# Unique words
unique_words = len(word_frequency.keys())
print('Unique_words: ',f'{unique_words}')
```

In [ ]:

```
# Repeated words in each label
positive_words = Counter()
neutral_words = Counter()
negative_words = Counter()

for index, row in df50.iterrows():
    words = row['token']
    label = row['label']
    if label == 1:
        positive_words.update(words)
    elif label == 2:
        negative_words.update(words)
    else:
        neutral_words.update(words)

#Resources:
#https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.iterrows.html
#https://www.kaggle.com/code/juicykn/imdb-movie-list-analysis-in-python-and-sql
```

In [ ]:

```
# Most repeated words in each label
top_positive_words = positive_words.most_common(10)
top_negative_words = negative_words.most_common(10)
top_neutral_words = neutral_words.most_common(10)

print('Positive: ', top_positive_words)
print('\n')
print('Negative: ', top_negative_words)
print('\n')
print('Neutral: ', top_neutral_words)
```

In [ ]:

```
# Spliting the tupple we got earlier
positive_words, positive_counts = zip(*top_positive_words)
negative_words, negative_counts = zip(*top_negative_words)
neutral_words, neutral_counts = zip(*top_neutral_words)

# Charts----------------------------------------------------
fig, axs = plt.subplots(3,1,figsize=(10,15))

# Positive words plot
axs[0].bar(positive_words, positive_counts, color='green')
axs[0].set_title('Most Frequent Positive Words')
axs[0].set_ylabel('Frequency')

# Negative words plot
axs[1].bar(neutral_words, neutral_counts, color='grey')
axs[1].set_title('Most Frequent Neutral Words')
axs[1].set_ylabel('Frequency')

# Neutral words plot
axs[2].bar(negative_words, negative_counts, color='red')
axs[2].set_title('Most Frequent Negative Words')
axs[2].set_ylabel('Frequency')

# Space between charts
plt.tight_layout(pad=4.0)
plt.show()

#Resources:
# https://realpython.com/python-zip-function/#using-zip-in-python
# https://matplotlib.org/stable/index.html
```

In [ ]:

```
# Splitting data into train 70%, validation 15%, test 15%

X_train_val, X_test, y_train_val, y_test = train_test_split(df50['sentence_preprocessed_1'], df50['label'],

X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.15, random_state=42
```

In [ ]:

```
# Feature extraction: Transforming data into TF-IDF features.
X_train = tfidf_vectorizer.fit_transform(X_train)
X_val = tfidf_vectorizer.transform(X_val)
X_test = tfidf_vectorizer.transform(X_test)
```

In [ ]:

```
print(X_train.shape)  # Should output (number_of_samples, 33154)
print(X_test.shape)   # Should also output (number_of_samples, 33154)
print(X_val.shape)
```

In [ ]:

```
#Turning sparse matrix into dense
X_train = X_train.toarray()
X_val = X_val.toarray()
X_test = X_test.toarray()

#Turning into PyTorch tensors
X_train = torch.tensor(X_train, dtype=torch.float32)
X_val = torch.tensor(X_val, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_train = torch.tensor(y_train.values, dtype=torch.float32)
y_val = torch.tensor(y_val.values, dtype=torch.float32)
y_test = torch.tensor(y_test.values, dtype=torch.float32)

#Resources:
# https://pytorch.org/docs/stable/tensors.html
```

In [ ]:

```python
# Time consumed (starts)
start_time = time.time()

# Building the Multilayer Perceptron model with back propagation.
class MLPmodel(nn.Module):
    def __init__(self):
        super(MLPmodel, self).__init__()
        self.fc1 = nn.Linear(9185,610)
        self.fc2 = nn.Linear(610,377)
        self.fc3 = nn.Linear(377,23)
        self.fc4 = nn.Linear(23,1)
        self.sigmoid = nn.Sigmoid()
        self.relu = nn.ReLU()

    def forward(self, x):
        hidden = self.relu(self.fc1(x))
        hidden = self.relu(self.fc2(hidden))
        hidden = self.relu(self.fc3(hidden))
        output = self.sigmoid(self.fc4(hidden))
        return output
```

In [ ]:

```python
# Model environment
model = MLPmodel() # Define the model
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.0001, weight_decay=0.00001)  # Using Adam optimizer

#Intersting combinations: 0.1/0.00001
#0.1/ 0
#Is there correlation? what is the relationship?

#Note: SDG was used earlier in during the experimentation, however, the performance was way worst than the
#Different arguments were used with the different parameters and anything changed barely.
```

In [ ]:

```python
# Training with early stoping
epochs = 5000
patience = 10 # Here we define how many epochs wait until we stop
best_val_loss = float('inf') #To save the best model/early stop
patience_counter = 0 #This one starts a counter to track number of epochs without improvement

for epoch in range(epochs):
    model.train()
    optimizer.zero_grad()
  #Forward Pass
    outputs = model(X_train)
    loss = criterion(outputs.squeeze(), y_train)
  #Backward and Optimize
    loss.backward()
    optimizer.step()

  #Validation
    model.eval()
    with torch.no_grad():
        val_outputs = model(X_val)
        val_loss = criterion(val_outputs.squeeze(), y_val)

  #Early stop
    if val_loss < best_val_loss:
        best_val_loss = val_loss
        best_model_state = model.state_dict()  # Save the best model state
        patience_counter=0
    else:
        patience_counter += 1

  #Print the early stopping
    if patience_counter > patience:
        print(f'Stopping early at epoch {epoch+1}.')
        break

    if (epoch+1) % 5 == 0:
        print(f'Epoch {epoch+1}/{epochs}, Loss: {loss.item()}, Val Loss: {val_loss.item()}')

if best_model_state:
    model.load_state_dict(best_model_state)

#Resources:
#https://pythonguides.com/pytorch-early-stopping/
#https://discuss.pytorch.org/t/can-i-deepcopy-a-model/52192
#https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early
#https://stackoverflow.com/questions/71998978/early-stopping-in-pytorch
#https://github.com/Bjarten/early-stopping-pytorch
#https://debuggercafe.com/using-learning-rate-scheduler-and-early-stopping-with-pytorch/
```

In [ ]:

```python
#Accuracy on validation set
model.eval()
with torch.no_grad():
    val_outputs = model(X_val)
    val_predicted_bin = (val_outputs.squeeze() > 0.5).int()

    #validation accuracy
    val_accuracy = accuracy_score(y_val.numpy(), val_predicted_bin.numpy())
    print(f'Validation set accuracy: {val_accuracy}')

# Resources:
# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
# https://pytorch.org/docs/stable/generated/torch.Tensor.numpy.html
# https://www.youtube.com/watch?v=E35CVhVKISA&t=135s
```

In [ ]:

```python
#Accuracy on test set
model.eval()

with torch.no_grad():
    test_outputs = model(X_test)
    predicted_bin = (test_outputs.squeeze() > 0.5).int()

#accuracy
test_accuracy = accuracy_score(y_test.numpy(), predicted_bin.numpy())
print(f'Accuracy on test set: {test_accuracy:.2f}')

#classification report
print(classification_report(y_test.numpy(), predicted_bin.numpy()))
```

In [ ]:

```
# Total Time Consumed
end_time = time.time()
execution_time = end_time - start_time
print(f"Total Execution Time: {execution_time} seconds")
```

In [ ]:

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Calculate the confusion matrix
cm = confusion_matrix(y_test.numpy(), predicted_bin.numpy())

# Plot the confusion matrix using seaborn heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='seismic', cbar=False,
            xticklabels=['Negative', 'Neutral', 'Positive'],
            yticklabels=['Negative', 'Neutral', 'Positive'])
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.tight_layout()
plt.show()
```

In [ ]:

```
# Getting ready the work environment. Importing libraries and modules:

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import nltk
import re
import seaborn as sns
import string
import time

from bs4 import BeautifulSoup
from collections import Counter
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
from sklearn.metrics import classification_report, roc_curve, roc_auc_score, confusion_matrix

#===========         Extra tools for the statistic analysis              ======================
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()
#----------------------------------------------------------------------

stop_words = stopwords.words('english')
tfidf_vectorizer = TfidfVectorizer(stop_words='english')
vectorizer = CountVectorizer()
```

In [ ]:

```
# Time consumed (starts)
start_time = time.time()

# Hyperparameters
param_grid = {'alpha': np.logspace(10, 15, 5),
              'fit_prior' : [True],
              'class_prior':[None, [0.01, 0.15, 0.6, 1]],
              }

# Training the model and looking for the best combination
grid_search = GridSearchCV(MultinomialNB(), param_grid, cv=3, verbose=2)
grid_search.fit(X_train, y_train)

# Getting the best estimator################################################
Best_NBmodel = grid_search.best_estimator_
Best_score = grid_search.best_score_

# Print results
print(f'Best Model: {Best_NBmodel}\n')
#print('\n')
print(f'Best CV Score: {Best_score}')
```

In [ ]:
```python
# Evaluation on the Validation
y_pred_val = Best_NBmodel.predict(X_val)
val_accuracy = accuracy_score(y_val, y_pred_val)

# Print validation results
print(f'Validation Accuracy: {val_accuracy}')
```

In [ ]:
```python
# Evaluation on the test set
y_pred_test = Best_NBmodel.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred_test)

# Print test results
print(f'Accuracy on test set: {test_accuracy}')
print(classification_report(y_test, y_pred_test))
```

In [ ]:
```python
# Predicting labels for the test
y_pred_test = Best_NBmodel.predict(X_test)

# Confusion matrix
cm = confusion_matrix(y_test, y_pred_test)

# Plot the confusion matrix using seaborn heatmap
plt.figure(figsize=(8, 6))

sns.heatmap(cm,
            annot=True,
            fmt='d',
            cmap='seismic',
            cbar=False,
            xticklabels=['Negative', 'Neutral', 'Positive'],
            yticklabels=['Negative', 'Neutral', 'Positive'])

plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')

plt.tight_layout()

plt.show()
```

In [ ]:
```python
# Total Time Consumed
end_time = time.time()
execution_time = end_time - start_time
print(f"Total Execution Time: {execution_time} seconds")
```

In [ ]:
```python
data into train 70%, validation 15%, test 15%

, X_test, y_train_val, y_test = train_test_split(df50['sentence_preprocessed_1'], df50['label'], test_size=0.15,

val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.15, random_state=42)
```

In [ ]:
```python
# Feature extraction: Transforming data into TF-IDF features.
X_train = tfidf_vectorizer.fit_transform(X_train)
X_val = tfidf_vectorizer.transform(X_val)
X_test = tfidf_vectorizer.transform(X_test)
```

In [ ]:
```python
# Checking shape
print(X_train.shape)
print(X_test.shape)
print(X_val.shape)
```

In [ ]:

```
# Time consumed (starts)
start_time = time.time()

# Multinomial Naive Bayes
model = MultinomialNB()

# Fit the model to the training data
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

# Print the accuracy
print(f"Accuracy: {accuracy:.2f}")

# Total Time Consumed
end_time = time.time()
execution_time = end_time - start_time
print(f"Total Execution Time: {execution_time} seconds")
```

In [ ]:

```
# Total Time Consumed
end_time = time.time()
execution_time = end_time - start_time
print(f"Total Execution Time: {execution_time} seconds")
```

In [ ]:

```
# Support Vector Machine working environment.
# Getting ready the work environment. Importing libraries and modules:

import time
import pandas as pd
import re
import nltk
import torch
import torch.nn as nn
import numpy as np
import string
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.svm import SVC
from sklearn.metrics import roc_curve, roc_auc_score, confusion_matrix
from sklearn.metrics import accuracy_score, precision_recall_fscore_support, classification_report
from sklearn.model_selection import train_test_split, GridSearchCV
from collections import Counter
from bs4 import BeautifulSoup
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

#===========          Extra tools for the statistic analysis              ======================
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()
#----------------------------------------------------------------------

stop_words = stopwords.words('english')
tfidf_vectorizer = TfidfVectorizer(stop_words='english')
vectorizer = CountVectorizer()
```

In [ ]:

```
data into train 70%, validation 15%, test 15%

, X_test, y_train_val, y_test = train_test_split(df50['sentence_preprocessed_1'], df50['label'], test_size=0.15,

val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.15, random_state=42)
```

In [ ]:

```
# Feature extraction: Transforming data into TF-IDF features.
X_train = tfidf_vectorizer.fit_transform(X_train)
X_val = tfidf_vectorizer.transform(X_val)
X_test = tfidf_vectorizer.transform(X_test)
```

In [ ]:

```python
# Checking shape
print(X_train.shape)
print(X_test.shape)
print(X_val.shape)
```

In [ ]:

```python
#Turning sparse matrix into dense
X_train = X_train.toarray()
X_val = X_val.toarray()
X_test = X_test.toarray()

#Turning into PyTorch tensors
X_train = torch.tensor(X_train, dtype=torch.float32)
X_val = torch.tensor(X_val, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_train = torch.tensor(y_train.values, dtype=torch.float32)
y_val = torch.tensor(y_val.values, dtype=torch.float32)
y_test = torch.tensor(y_test.values, dtype=torch.float32)

#Resources:
# https://pytorch.org/docs/stable/tensors.html
```

In [ ]:

```python
# Time Consumed (starts)
start_time = time.time()

# Definition of the SVM model and hyperparameter for tuning on the training set
param_grid = {'C': np.logspace(-15, 15, 2), 'kernel': ['rbf'], 'tol' : [1e-3]}

# Hyperparameters tuning, cross-validation using training set.
grid_search = GridSearchCV(SVC(), param_grid, cv=2, scoring='accuracy', verbose=2)
grid_search.fit(X_train, y_train)

# Getting the best estimator
Best_SVMmodel = grid_search.best_estimator_
Best_score = grid_search.best_score_

# Print results
print(f'Best Model: {Best_SVMmodel}')
print('\n')
print(f'Best CV Score: {Best_score}')
```

In [ ]:

```python
# Evaluation of the model on the validation set witht he best parameters
y_pred = Best_SVMmodel.predict(X_val)
val_accuracy = accuracy_score(y_val, y_pred)

# Print results
print(f'Validation set accuracy: {val_accuracy}')
```

In [ ]:

```python
# Evaluation of the final model using the test set
y_pred = Best_SVMmodel.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred)

#Print results
print(f'Accuracy on test set: {test_accuracy}')
print(classification_report(y_test, y_pred))
```

In [ ]:

```python
# Predicting labels for the test
y_pred_test = Best_SVMmodel.predict(X_test)

# Confusion matrix
cm = confusion_matrix(y_test, y_pred_test)

# Plot the confusion matrix using seaborn heatmap
plt.figure(figsize=(8, 6))

sns.heatmap(cm,
            annot=True,
            fmt='d',
            cmap='seismic',
            cbar=False,
            xticklabels=['Negative', 'Neutral', 'Positive'],
            yticklabels=['Negative', 'Neutral', 'Positive'])

plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')


plt.show()
```

In [ ]:

```python
# Total Time Consumed
end_time = time.time()
execution_time = end_time - start_time
print(f"Total Execution Time: {execution_time} seconds")
```

In [ ]:

```python
# Support Vector Machine working environment.
# Getting ready the work environment. Importing libraries and modules:

import time
import pandas as pd
import re
import nltk
import torch
import torch.nn as nn
import numpy as np
import string
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.svm import SVC
from sklearn.metrics import roc_curve, roc_auc_score, confusion_matrix
from sklearn.metrics import accuracy_score, precision_recall_fscore_support, classification_report
from sklearn.model_selection import train_test_split, GridSearchCV
from collections import Counter
from bs4 import BeautifulSoup
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

#===========          Extra tools for the statistic analysis              ======================
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()
#----------------------------------------------------------------------

stop_words = stopwords.words('english')
tfidf_vectorizer = TfidfVectorizer(stop_words='english')
vectorizer = CountVectorizer()
```

In [ ]:

```python
data into train 70%, validation 15%, test 15%

, X_test, y_train_val, y_test = train_test_split(df50['sentence_preprocessed_1'], df50['label'], test_size=0.15,

val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.15, random_state=42)
```

In [ ]:

```python
# Feature extraction: Transforming data into TF-IDF features.
X_train = tfidf_vectorizer.fit_transform(X_train)
X_val = tfidf_vectorizer.transform(X_val)
X_test = tfidf_vectorizer.transform(X_test)
```

In [ ]:

```
# Checking shape
print(X_train.shape)
print(X_test.shape)
print(X_val.shape)
```

In [ ]:

```
#Turning sparse matrix into dense
X_train = X_train.toarray()
X_val = X_val.toarray()
X_test = X_test.toarray()

#Turning into PyTorch tensors
X_train = torch.tensor(X_train, dtype=torch.float32)
X_val = torch.tensor(X_val, dtype=torch.float32)
X_test = torch.tensor(X_test, dtype=torch.float32)
y_train = torch.tensor(y_train.values, dtype=torch.float32)
y_val = torch.tensor(y_val.values, dtype=torch.float32)
y_test = torch.tensor(y_test.values, dtype=torch.float32)

#Resources:
# https://pytorch.org/docs/stable/tensors.html
```

In [ ]:

```
# Time Consumed (starts)
start_time = time.time()

# Initialize the Support Vector Machine classifier with default parameters
svm_model = SVC()

# Fit the model
svm_model.fit(X_train, y_train)

# Predict the labels
y_pred = svm_model.predict(X_test)

# Calculate the accuracy
accuracy = accuracy_score(y_test, y_pred)

# Print the accuracy
print(f"Accuracy: {accuracy:.2f}")


# Total Time Consumed
end_time = time.time()
execution_time = end_time - start_time
print(f"Total Execution Time: {execution_time} seconds")
```

In [ ]:

```
#!pip install transformers[torch]
#!pip install torch transformers datasets
#!pip install accelerate -U

# Getting ready the work environment. Importing libraries and modules:

import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import torch
import time
import pandas as pd

from datasets import load_dataset
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from transformers import BertTokenizer, BertForSequenceClassification, Trainer, TrainingArguments
```

In [ ]:

```
# Time consumed (starts)
start_time = time.time()

# Loading the pre-trained FinBERT model and tokenizer
model_ = "ProsusAI/finbert"
tokenizer = BertTokenizer.from_pretrained(model_)
model = BertForSequenceClassification.from_pretrained(model_)

# We directly load the Financial PhraseBank dataset from The Hugging Face
dataset = load_dataset("financial_phrasebank", "sentences_allagree") # https://huggingface.co/datasets/finan
#"sentences_50agree"
#"sentences_66agree"
#"sentences_75agree"
```

In [ ]:

```
# Preprocess the dataset
def preprocess_function(examples):
    return tokenizer(examples["sentence"],
                     padding="longest",
                     max_length= 5,
                     truncation = True)

dataset = dataset.map(preprocess_function, batched=True)

# Split the dataset into training and validation sets
train_test_split = dataset['train'].train_test_split(test_size=0.2, seed=42)
train_dataset = train_test_split['train']
eval_dataset = train_test_split['test']


```

In [ ]:

```
# Training arguments
training_args = TrainingArguments(
    output_dir = "./results",
    evaluation_strategy = "epoch",
    learning_rate = 1,
    per_device_train_batch_size = 10,
    per_device_eval_batch_size = 10,
    num_train_epochs = 1,
    warmup_steps = 200,
    save_strategy = "epoch"
)

# Define the trainer
trainer = Trainer(
    model = model,
    args = training_args,
    train_dataset = train_dataset,
    eval_dataset = eval_dataset,
)

# Fine-tune the model
trainer.train()
```

In [ ]:

```
# Evaluate the fine-tuned model
predictions = trainer.predict(eval_dataset)
preds = np.argmax(predictions.predictions, axis=1)
labels = eval_dataset["label"]
accuracy = accuracy_score(labels, preds)
report = classification_report(labels, preds)

print(f"Validation Accuracy: {accuracy:.2f}")
print(f"Classification Report:\n{report}")
```

In [ ]:

```
# Confusion matrix
cm =  confusion_matrix(labels, preds)

# Plot the confusion matrix using seaborn heatmap
plt.figure(figsize=(8, 6))

sns.heatmap(cm,
            annot=True,
            fmt='d',
            cmap='seismic',
            cbar=False,
            xticklabels=['Negative', 'Neutral', 'Positive'],
            yticklabels=['Negative', 'Neutral', 'Positive'])

plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')


plt.show()
```

In [ ]:

```
# Load the test dataset
test_dataset = load_dataset("financial_phrasebank", "sentences_75agree")
test_dataset = test_dataset.map(preprocess_function, batched=True)

# Evaluate the fine-tuned model on the test set
test_predictions = trainer.predict(test_dataset['train'])
test_preds = np.argmax(test_predictions.predictions, axis=1)
test_labels = test_dataset['train']["label"]
test_accuracy = accuracy_score(test_labels, test_preds)
test_report = classification_report(test_labels, test_preds)

```

In [ ]:

```
print(f"Test Accuracy: {test_accuracy:.2f}")
print(f"Test Classification Report:\n{test_report}")
```

In [ ]:

```
# Confusion matrix
test_cm = confusion_matrix(test_labels, test_preds)

# Plot the confusion matrix using seaborn heatmap
plt.figure(figsize=(8, 6))

sns.heatmap(test_cm,
            annot=True,
            fmt='d',
            cmap='seismic',
            cbar=False,
            xticklabels=['Negative', 'Neutral', 'Positive'],
            yticklabels=['Negative', 'Neutral', 'Positive'])

plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')


plt.show()
```

In [ ]:

```
# Total Time Consumed
end_time = time.time()
execution_time = end_time - start_time
print(f"Total Execution Time: {execution_time} seconds")
```

In [ ]:

```
#!pip install transformers[torch]
#!pip install torch transformers datasets
#!pip install accelerate -U

# Getting ready the work environment. Importing libraries and modules:

import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import torch
import time
import pandas as pd

from datasets import load_dataset
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from transformers import BertTokenizer, BertForSequenceClassification, Trainer, TrainingArguments
```

In [ ]:

```
# Time consumed (starts)
start_time = time.time()

# Loading the pre-trained FinBERT model and tokenizer
model_ = "ProsusAI/finbert"
tokenizer = BertTokenizer.from_pretrained(model_)
model = BertForSequenceClassification.from_pretrained(model_)

# We directly load the Financial PhraseBank dataset from The Hugging Face
dataset = load_dataset("financial_phrasebank", "sentences_allagree") # https://huggingface.co/datasets/fina
#"sentences_50agree"
#"sentences_66agree"
#"sentences_75agree"
```

In [ ]:

```python
# Preprocess the dataset
def preprocess_function(examples):
    return tokenizer(examples["sentence"],
                     padding="longest",
                     max_length= 45,
                     truncation = True)

dataset = dataset.map(preprocess_function, batched=True)

# Split the dataset into training and validation sets
train_test_split = dataset['train'].train_test_split(test_size=0.2, seed=42)
train_dataset = train_test_split['train']
eval_dataset = train_test_split['test']
```

In [ ]:

```python
# Training arguments
training_args = TrainingArguments(
    output_dir = "./results",
    evaluation_strategy = "epoch",
    learning_rate = 0.0001,
    per_device_train_batch_size = 25,
    per_device_eval_batch_size = 25,
    num_train_epochs = 2,
    warmup_steps = 500,
    save_strategy = "epoch"
)

# Define the trainer
trainer = Trainer(
    model = model,
    args = training_args,
    train_dataset = train_dataset,
    eval_dataset = eval_dataset,
)

# Fine-tune the model
trainer.train()
```

In [ ]:

```python
# Evaluate the fine-tuned model
predictions = trainer.predict(eval_dataset)
preds = np.argmax(predictions.predictions, axis=1)
labels = eval_dataset["label"]
accuracy = accuracy_score(labels, preds)
report = classification_report(labels, preds)

print(f"Validation Accuracy: {accuracy:.2f}")
print(f"Classification Report:\n{report}")
```

In [ ]:

```python
# Confusion matrix
cm =  confusion_matrix(labels, preds)

# Plot the confusion matrix using seaborn heatmap
plt.figure(figsize=(8, 6))

sns.heatmap(cm,
            annot=True,
            fmt='d',
            cmap='seismic',
            cbar=False,
            xticklabels=['Negative', 'Neutral', 'Positive'],
            yticklabels=['Negative', 'Neutral', 'Positive'])

plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')


plt.show()
```

In [ ]:
```python
# Load the test dataset
test_dataset = load_dataset("financial_phrasebank", "sentences_75agree")
test_dataset = test_dataset.map(preprocess_function, batched=True)

# Evaluate the fine-tuned model on the test set
test_predictions = trainer.predict(test_dataset['train'])
test_preds = np.argmax(test_predictions.predictions, axis=1)
test_labels = test_dataset['train']["label"]
test_accuracy = accuracy_score(test_labels, test_preds)
test_report = classification_report(test_labels, test_preds)

```

In [ ]:
```python
print(f"Test Accuracy: {test_accuracy:.2f}")
print(f"Test Classification Report:\n{test_report}")
```

In [ ]:
```python
# Confusion matrix
test_cm = confusion_matrix(test_labels, test_preds)

# Plot the confusion matrix using seaborn heatmap
plt.figure(figsize=(8, 6))

sns.heatmap(test_cm,
            annot=True,
            fmt='d',
            cmap='seismic',
            cbar=False,
            xticklabels=['Negative', 'Neutral', 'Positive'],
            yticklabels=['Negative', 'Neutral', 'Positive'])

plt.title('Confusion Matrix')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')


plt.show()
```

In [ ]:
```python
# Total Time Consumed
end_time = time.time()
execution_time = end_time - start_time
print(f"Total Execution Time: {execution_time} seconds")
```