

## 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]: 1 # Multilayer perceptron working environment.
2 # Getting ready the work environment. Importing Libraries and modules:
3 import time
4 import pandas as pd
5 import re
6 import nltk
7 import torch
8 import torch.nn as nn
9 import numpy as np
10 import string
11 import matplotlib.pyplot as plt
12 import seaborn as sns
13
14 from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
15 from sklearn.model_selection import train_test_split
16 from sklearn.metrics import precision_recall_fscore_support, classification_report, roc_curve, roc_auc_score
17 from collections import Counter
18 from bs4 import BeautifulSoup
19 from nltk.corpus import stopwords
20 from nltk.tokenize import word_tokenize
21
22 #===== Extra tools for the statistic analysis =====
23 from nltk.stem import WordNetLemmatizer
24
25 lemmatizer = WordNetLemmatizer()
26 #-----
27
28 stop_words = stopwords.words('english')
29 tfidf_vectorizer = TfidfVectorizer()
30 vectorizer = CountVectorizer()
```

```
In [2]: 1 # Getting ready the work environment. Importing Libraries and modules:
2 import seaborn as sns
3 import matplotlib.pyplot as plt
4 import pandas as pd
5 import re
6 import nltk
7 import string
8
9 from sklearn.naive_bayes import MultinomialNB
10 from sklearn.model_selection import train_test_split, GridSearchCV
11 from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
12 from sklearn.linear_model import LogisticRegression
13 from sklearn.metrics import accuracy_score, precision_recall_fscore_support
14 from sklearn.metrics import classification_report, roc_curve, roc_auc_score, confusion_matrix
15 from nltk.corpus import stopwords
16 from nltk.tokenize import word_tokenize
17 from collections import Counter
18 from bs4 import BeautifulSoup
19
20
21 #===== Extra tools for the statistic analysis =====
22 from nltk.stem import WordNetLemmatizer
23
24 lemmatizer = WordNetLemmatizer()
25 #-----
26
27 stop_words = stopwords.words('english')
28 tfidf_vectorizer = TfidfVectorizer(stop_words='english')
29 vectorizer = CountVectorizer()
```

## Load dataset

In [3]:

```
1 # Importing dataset from the hugging face
2 from datasets import load_dataset
3
4 dataset1 = load_dataset('financial_phrasebank', 'sentences_50agree')
5 dataset2 = load_dataset('financial_phrasebank', 'sentences_66agree')
6 dataset3 = load_dataset('financial_phrasebank', 'sentences_75agree')
```

Found cached dataset financial\_phrasebank (C:/Users/nonox/.cache/huggingface/datasets/financial\_phrasebank/sentences\_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/sentences\_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/sentences\_75agree/1.0.0/550bde12e6c30e2674da973a55f57edde5181d53f5a5a34c1531c53f93b7e141)

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

In [4]:

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

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

In [5]:

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

In [6]:

```
1 # Checking data
2 print(df50)
3 print("\n")
4 print(df66)
5 print("\n")
6 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...
```

[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...
```

[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...
```

[3453 rows x 2 columns]

In [7]:

```
1 # Checking the data we will preprocess
2 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]:

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

```
In [9]: 1 # Function to remove HTML tags:
2 def remove_html(text):
3     soup = BeautifulSoup(text, "html.parser")
4     return soup.get_text()
5
6 #Sources:
7 #https://stackoverflow.com/questions/328356/extracting-text-from-html-file-using-python?newreg=aa9f4dc4aea34
8 #https://beautiful-soup-4.readthedocs.io/en/latest/
9 #https://www.datacamp.com/tutorial/web-scraping-using-python
10 #https://www.geeksforgeeks.org/how-to-write-the-output-to-html-file-with-python-beautifulsoup/

In [10]: 1 def remove_urls(text):
2     return re.sub(r'https?://\S+|www\.\S+', '', text)
3
4 #Source: https://www.geeksforgeeks.org/remove-urls-from-string-in-python/

In [11]: 1 def remove_punctuation(text):
2     return text.translate(str.maketrans('', '', string.punctuation))
3
4 #Source: https://stackoverflow.com/questions/34293875/how-to-remove-punctuation-marks-from-a-string-in-python

In [12]: 1 # Function to remove HTML tags:
2 def remove_html(text):
3     soup = BeautifulSoup(text, "html.parser")
4     return soup.get_text()
5
6 #Sources:
7 #https://stackoverflow.com/questions/328356/extracting-text-from-html-file-using-python?newreg=aa9f4dc4aea34
8 #https://beautiful-soup-4.readthedocs.io/en/latest/
9 #https://www.datacamp.com/tutorial/web-scraping-using-python
10 #https://www.geeksforgeeks.org/how-to-write-the-output-to-html-file-with-python-beautifulsoup/

In [13]: 1 # Function to put together all the previous functions:
2 def preprocess_1(text):
3     text = remove_html(text)
4     text = remove_urls(text)
5     text = remove_punctuation(text)
6     text = text.lower()
7     return text
8
9 df50['sentence_preprocessed_1'] = df50['sentence'].apply(preprocess_1)
```

C:\Users\nonox\AppData\Local\Temp\ipykernel\_14764\2872886666.py:3: MarkupResemblesLocatorWarning: The input looks more like a filename than markup. You may want to open this file and pass the filehandle into BeautifulSoup Soup.

```
soup = BeautifulSoup(text, "html.parser")
```

## ) Tokenization

```
In [14]: 1 # Function to tokenize and convert to lower case the text in review column
2 def tokenize(text):
3     tokens = re.findall(r'\b\w+\b', text)
4     return tokens
5
6 #Tokenization
7 df50['token'] = df50['sentence_preprocessed_1'].apply(tokenize)
```

## ) Removing Stop Words

```
In [15]: 1 # Function to remove stop words from the tokenized review column
2 def remove_stopwords(tokens):
3     filtered_tokens = [word for word in tokens if word not in stop_words]
4     return filtered_tokens
5
6 #Remove stopwords
7 df50['token'] = df50['token'].apply(remove_stopwords)
```

## ) Some statistics

```
In [16]: 1 # Calculating the total tokens for each review
2 df50['token_count'] = df50['token'].apply(lambda x: len(x) if isinstance(x, list) else 0)
3
4 # Dispersion and central tendency measurements
5 statistics = df50.groupby('label')['token_count'].agg(['min', 'max', 'mean', 'var', 'std'])
6
7 # Avg words per review:
8 avg_words = df50['token'].apply(len).mean()
9
10 #Print the statistics
11 print("Statistics by Label: ")
12 print('\n')
13 print(statistics)
14 print('\n')
15 print('\n')
16 print('Average Words: ', f"{avg_words:.0f}")
17
18 #Resources:
19 #https://www.geeksforgeeks.org/pandas-groupby-one-column-and-get-mean-min-and-max-values/
20 #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]: 1 # Iterating through the List of Lists(each row) to create a new List with all the tokens
2 def word_freq(list_of_list):
3     single_list = [item for sublist in list_of_list for item in sublist]
4     token_freq = Counter(single_list)
5     return token_freq
6
7 # Counting the frequency for each word.
8 word_frequency = word_freq(df50['token'])
9 print(word_frequency)
10
11 #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, '2010': 238, 'share': 237, 'also': 224, 'services': 223, 'market': 217, 'shares': 198, 'first': 193, '2006': 173, '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, 'technology': 120, 'corresponding': 119, 'plant': 118, 'solutions': 117, 'service': 116, 'increased': 109, 'construction': 109, 'capital': 109, 'agreement': 106, 'investment': 105, '2005': 104, 'well': 104, 'increase': 103, 'rose': 102, 'customers': 102, 'pct': 99, 'value': 98, 'order': 97, 'us': 97, 'oy': 97, 'stock': 94, 'would': 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, 'paper': 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': 69, 'area': 67, '15': 67, 'news': 67, 'data': 67, 'system': 67, 'people': 67, 'growth': 65, 'billion': 65, 'acquisition': 65, 'september': 64, 'line': 63, 'ltd': 63, '2': 63, 'euros': 62, 'months': 62, 'report': 61, '4': 61, 'countries': 61, 'based': 61, 'may': 61, 'euro': 60, '30': 60, 'network': 59, 'information': 59, 'u': 59, '2004': 59, '2003': 59, '2002': 59, '2001': 59, '2000': 59, '1999': 59, '1998': 59, '1997': 59, '1996': 59, '1995': 59, '1994': 59, '1993': 59, '1992': 59, '1991': 59, '1990': 59, '1989': 59, '1988': 59, '1987': 59, '1986': 59, '1985': 59, '1984': 59, '1983': 59, '1982': 59, '1981': 59, '1980': 59, '1979': 59, '1978': 59, '1977': 59, '1976': 59, '1975': 59, '1974': 59, '1973': 59, '1972': 59, '1971': 59, '1970': 59, '1969': 59, '1968': 59, '1967': 59, '1966': 59, '1965': 59, '1964': 59, '1963': 59, '1962': 59, '1961': 59, '1960': 59, '1959': 59, '1958': 59, '1957': 59, '1956': 59, '1955': 59, '1954': 59, '1953': 59, '1952': 59, '1951': 59, '1950': 59, '1949': 59, '1948': 59, '1947': 59, '1946': 59, '1945': 59, '1944': 59, '1943': 59, '1942': 59, '1941': 59, '1940': 59, '1939': 59, '1938': 59, '1937': 59, '1936': 59, '1935': 59, '1934': 59, '1933': 59, '1932': 59, '1931': 59, '1930': 59, '1929': 59, '1928': 59, '1927': 59, '1926': 59, '1925': 59, '1924': 59, '1923': 59, '1922': 59, '1921': 59, '1920': 59, '1919': 59, '1918': 59, '1917': 59, '1916': 59, '1915': 59, '1914': 59, '1913': 59, '1912': 59, '1911': 59, '1910': 59, '1909': 59, '1908': 59, '1907': 59, '1906': 59, '1905': 59, '1904': 59, '1903': 59, '1902': 59, '1901': 59, '1900': 59, '1899': 59, '1898': 59, '1897': 59, '1896': 59, '1895': 59, '1894': 59, '1893': 59, '1892': 59, '1891': 59, '1890': 59, '1889': 59, '1888': 59, '1887': 59, '1886': 59, '1885': 59, '1884': 59, '1883': 59, '1882': 59, '1881': 59, '1880': 59, '1879': 59, '1878': 59, '1877': 59, '1876': 59, '1875': 59, '1874': 59, '1873': 59, '1872': 59, '1871': 59, '1870': 59, '1869': 59, '1868': 59, '1867': 59, '1866': 59, '1865': 59, '1864': 59, '1863': 59, '1862': 59, '1861': 59, '1860': 59, '1859': 59, '1858': 59, '1857': 59, '1856': 59, '1855': 59, '1854': 59, '1853': 59, '1852': 59, '1851': 59, '1850': 59, '1849': 59, '1848': 59, '1847': 59, '1846': 59, '1845': 59, '1844': 59, '1843': 59, '1842': 59, '1841': 59, '1840': 59, '1839': 59, '1838': 59, '1837': 59, '1836': 59, '1835': 59, '1834': 59, '1833': 59, '1832': 59, '1831': 59, '1830': 59, '1829': 59, '1828': 59, '1827': 59, '1826': 59, '1825': 59, '1824': 59, '1823': 59, '1822': 59, '1821': 59, '1820': 59, '1819': 59, '1818': 59, '1817': 59, '1816': 59, '1815': 59, '1814': 59, '1813': 59, '1812': 59, '1811': 59, '1810': 59, '1809': 59, '1808': 59, '1807': 59, '1806': 59, '1805': 59, '1804': 59, '1803': 59, '1802': 59, '1801': 59, '1800': 59, '1799': 59, '1798': 59, '1797': 59, '1796': 59, '1795': 59, '1794': 59, '1793': 59, '1792': 59, '1791': 59, '1790': 59, '1789': 59, '1788': 59, '1787': 59, '1786': 59, '1785': 59, '1784': 59, '1783': 59, '1782': 59, '1781': 59, '1780': 59, '1779': 59, '1778': 59, '1777': 59, '1776': 59, '1775': 59, '1774': 59, '1773': 59, '1772': 59, '1771': 59, '1770': 59, '1769': 59, '1768': 59, '1767': 59, '1766': 59, '1765': 59, '1764': 59, '1763': 59, '1762': 59, '1761': 59, '1760': 59, '1759': 59, '1758': 59, '1757': 59, '1756': 59, '1755': 59, '1754': 59, '1753': 59, '1752': 59, '1751': 59, '1750': 59, '1749': 59, '1748': 59, '1747': 59, '1746': 59, '1745': 59, '1744': 59, '1743': 59, '1742': 59, '1741': 59, '1740': 59, '1739': 59, '1738': 59, '1737': 59, '1736': 59, '1735': 59, '1734': 59, '1733': 59, '1732': 59, '1731': 59, '1730': 59, '1729': 59, '1728': 59, '1727': 59, '1726': 59, '1725': 59, '1724': 59, '1723': 59, '1722': 59, '1721': 59, '1720': 59, '1719': 59, '1718': 59, '1717': 59, '1716': 59, '1715': 59, '1714': 59, '1713': 59, '1712': 59, '1711': 59, '1710': 59, '1709': 59, '1708': 59, '1707': 59, '1706': 59, '1705': 59, '1704': 59, '1703': 59, '1702': 59, '1701': 59, '1700': 59, '1699': 59, '1698': 59, '1697': 59, '1696': 59, '1695': 59, '1694': 59, '1693': 59, '1692': 59, '1691': 59, '1690': 59, '1689': 59, '1688': 59, '1687': 59, '1686': 59, '1685': 59, '1684': 59, '1683': 59, '1682': 59, '1681': 59, '1680': 59, '1679': 59, '1678': 59, '1677': 59, '1676': 59, '1675': 59, '1674': 59, '1673': 59, '1672': 59, '1671': 59, '1670': 59, '1669': 59, '1668': 59, '1667': 59, '1666': 59, '1665': 59, '1664': 59, '1663': 59, '1662': 59, '1661': 59, '1660': 59, '1659': 59, '1658': 59, '1657': 59, '1656': 59, '1655': 59, '1654': 59, '1653': 59, '1652': 59, '1651': 59, '1650': 59, '1649': 59, '1648': 59, '1647': 59, '1646': 59, '1645': 59, '1644': 59, '1643': 59, '1642': 59, '1641': 59, '1640': 59, '1639': 59, '1638': 59, '1637': 59, '1636': 59, '1635': 59, '1634': 59, '1633': 59, '1632': 59, '1631': 59, '1630': 59, '1629': 59, '1628': 59, '1627': 59, '1626': 59, '1625': 59, '1624': 59, '1623': 59, '1622': 59, '1621': 59, '1620': 59, '1619': 59, '1618': 59, '1617': 59, '1616': 59, '1615': 59, '1614': 59, '1613': 59, '1612': 59, '1611': 59, '1610': 59, '1609': 59, '1608': 59, '1607': 59, '1606': 59, '1605': 59, '1604': 59, '1603': 59, '1602': 59, '1601': 59, '1600': 59, '1599': 59, '1598': 59, '1597': 59, '1596': 59, '1595': 59, '1594': 59, '1593': 59, '1592': 59, '1591': 59, '1590': 59, '1589': 59, '1588': 59, '1587': 59, '1586': 59, '1585': 59, '1584': 59, '1583': 59, '1582': 59, '1581': 59, '1580': 59, '1579': 59, '1578': 59, '1577': 59, '1576': 59, '1575': 59, '1574': 59, '1573': 59, '1572': 59, '1571': 59, '1570': 59, '1569': 59, '1568': 59, '1567': 59, '1566': 59, '1565': 59, '1564': 59, '1563': 59, '1562': 59, '1561': 59, '1560': 59, '1559': 59, '1558': 59, '1557': 59, '1556': 59, '1555': 59, '1554': 59, '1553': 59, '1552': 59, '1551': 59, '1550': 59, '1549': 59, '1548': 59, '1547': 59, '1546': 59, '1545': 59, '1544': 59, '1543': 59, '1542': 59, '1541': 59, '1540': 59, '1539': 59, '1538': 59, '1537': 59, '1536': 59, '1535': 59, '1534': 59, '1533': 59, '1532': 59, '1531': 59, '1530': 59, '1529': 59, '1528': 59, '1527': 59, '1526': 59, '1525': 59, '1524': 59, '1523': 59, '1522': 59, '1521': 59, '1520': 59, '1519': 59, '1518': 59, '1517': 59, '1516': 59, '1515': 59, '1514': 59, '1513': 59, '1512': 59, '1511': 59, '1510': 59, '1509': 59, '1508': 59, '1507': 59, '1506': 59, '1505': 59, '1504': 59, '1503': 59, '1502': 59, '1501': 59, '1500': 59, '1499': 59, '1498': 59, '1497': 59, '1496': 59, '1495': 59, '1494': 59, '1493': 59, '1492': 59, '1491': 59, '1490': 59, '1489': 59, '1488': 59, '1487': 59, '1486': 59, '1485': 59, '1484': 59, '1483': 59, '1482': 59, '1481': 59, '1480': 59, '1479': 59, '1478': 59, '1477': 59, '1476': 59, '1475': 59, '1474': 59, '1473': 59, '1472': 59, '1471': 59, '1470': 59, '1469': 59, '1468': 59, '1467': 59, '1466': 59, '1465': 59, '1464': 59, '1463': 59, '1462': 59, '1461': 59, '1460': 59, '1459': 59, '1458': 59, '1457': 59, '1456': 59, '1455': 59, '1454': 59, '1453': 59, '1452': 59, '1451': 59, '1450': 59, '1449': 59, '1448': 59, '1447': 59, '1446': 59, '1445': 59, '1444': 59, '1443': 59, '1442': 59, '1441': 59, '1440': 59, '1439': 59, '1438': 59, '1437': 59, '1436': 59, '1435': 59, '1434': 59, '1433': 59, '1432': 59, '1431': 59, '1430': 59, '1429': 59, '1428': 59, '1427': 59, '1426': 59, '1425': 59, '1424': 59, '1423': 59, '1422': 59, '1421': 59, '1420': 59, '1419': 59, '1418': 59, '1417': 59, '1416': 59, '1415': 59, '1414': 59, '1413': 59, '1412': 59, '1411': 59, '1410': 59, '1409': 59, '1408': 59, '1407': 59, '1406': 59, '1405': 59, '1404': 59, '1403': 59, '1402': 59, '1401': 59, '1400': 59, '1399': 59, '1398': 59, '1397': 59, '1396': 59, '1395': 59, '1394': 59, '1393': 59, '1392': 59, '1391': 59, '1390': 59, '1389': 59, '1388': 59, '1387': 59, '1386': 59, '1385': 59, '1384': 59, '1383': 59, '1382': 59, '1381': 59, '1380': 59, '1379': 59, '1378': 59, '1377': 59, '1376': 59, '1375': 59, '1374': 59, '1373': 59, '1372': 59, '1371': 59, '1370': 59, '1369': 59, '1368': 59, '1367': 59, '1366': 59, '1365': 59, '1364': 59, '1363': 59, '1362': 59, '1361': 59, '1360': 59, '1359': 59, '1358': 59, '1357': 59, '1356': 59, '1355': 59, '1354': 59, '1353': 59, '1352': 59, '1351': 59, '1350': 59, '1349': 59, '1348': 59, '1347': 59, '1346': 59, '1345': 59, '1344': 59, '1343': 59, '1342': 59, '1341': 59, '1340': 59, '1339': 59, '1338': 59, '1337': 59, '1336': 59, '1335': 59, '1334': 59, '1333': 59, '1332': 59, '1331': 59, '1330': 59, '1329': 59, '1328': 59, '1327': 59, '1326': 59, '1325': 59, '1324': 59, '1323': 59, '1322': 59, '1321': 59, '1320': 59, '1319': 59, '1318': 59, '1317': 59, '1316': 59, '1315': 59, '1314': 59, '1313': 59, '1312': 59, '1311': 59, '1310': 59, '1309': 59, '1308': 59, '1307': 59, '1306': 59, '1305': 59, '1304': 59, '1303': 59, '1302': 59, '1301': 59, '1300': 59, '1299': 59, '1298': 59, '1297': 59, '1296': 59, '1295': 59, '1294': 59, '1293': 59, '1292': 59, '1291': 59, '1290': 59, '1289': 59, '1288': 59, '1287': 59, '1286': 59, '1285': 59, '1284': 59, '1283': 59, '1282': 59, '1281': 59, '1280': 59, '1279': 59, '1278': 59, '1277': 59, '1276': 59, '1275': 59, '1274': 59, '1273': 59, '1272': 59, '1271': 59, '1270': 59, '1269': 59, '1268': 59, '1267': 59, '1266': 59, '1265': 59, '1264': 59, '1263': 59, '1262': 59, '1261': 59, '1260': 59, '1259': 59, '1258': 59, '1257': 59, '1256': 59, '1255': 59, '1254': 59, '1253': 59, '1252': 59, '1251': 59, '1250': 59, '1249': 59, '1248': 59, '1247': 59, '1246': 59, '1245': 59, '1244': 59, '1243': 59, '1242': 59, '1241': 59, '1240': 59, '1239': 59, '1238': 59, '1237': 59, '1236': 59, '1235': 59, '1234': 59, '1233': 59, '1232': 59, '1231': 59, '1230': 59, '1229': 59, '1228': 59, '1227': 59, '1226': 59, '1225': 59, '1224': 59, '1223': 59, '1222': 59, '1221': 59, '1220': 59, '1219': 59, '1218': 59, '1217': 59, '1216': 59, '1215': 59, '1214': 59, '1213': 59, '1212': 59, '1211': 59, '1210': 59, '1209': 59, '1208': 59, '1207': 59, '1206': 59, '1205': 59, '1204': 59, '1203': 59, '1202': 59, '1201': 59, '1200': 59, '1199': 59, '1198': 59, '1197': 59, '1196': 59, '1195': 59, '1194': 59, '1193': 59, '1192': 59, '1191': 59, '1190': 59, '1189': 59, '1188': 59, '1187': 59, '1186': 59, '1185': 59, '1184': 59, '1183': 59, '1182': 59, '1181': 59, '1180': 59, '1179': 59, '1178': 59, '1177': 59, '1176': 59, '1175': 59, '1174': 59, '1173': 59, '1172': 59, '1171': 59, '1170': 59, '1169': 59, '1168': 59, '1167': 59, '1166': 59, '1165': 59, '1164': 59, '1163': 59, '1162': 59, '1161': 59, '1160': 59, '1159': 59, '1158': 59, '1157': 59, '1156': 59, '1155': 59, '1154': 59, '1153': 59, '1152': 59, '1151': 59, '1150': 59, '1149': 59, '1148': 59, '1147': 59, '1146': 59, '1145': 59, '1144': 59, '1143': 59, '1142': 59, '1141': 59, '1140': 59, '1139': 59, '1138': 59, '1137': 59, '1136': 59, '1135': 59, '1134': 59, '1133': 59, '1132': 59, '1131': 59, '1130': 59, '1129': 59, '1128': 59, '1127': 59, '1126': 59, '1125': 59, '1124': 59, '1123': 59, '1122': 59, '1121': 59, '1120': 59, '1119': 59, '1118': 59, '1117': 59, '1116': 59, '1115': 59, '1114': 59, '1113': 59, '1112': 59, '1111': 59, '1110': 59, '1109': 59, '1108': 59, '1107': 59, '1106': 59, '1105': 59, '1104': 59, '1103': 59, '1102': 59, '1101': 59, '1100': 59, '1099': 59, '1098': 59, '1097': 59, '1096': 59, '1095': 59, '1094': 59, '1093': 59, '1092': 59, '1091': 59, '1090': 59, '1089': 59, '1088': 59, '1087': 59, '1086': 59, '1085': 59, '1084': 59, '1083': 59, '1082': 59, '1081': 59, '1080': 59, '1079': 59, '1078': 59, '1077': 59, '1076': 59, '1075': 59, '1074': 59, '1073': 59, '1072': 59, '1071': 59, '1070': 59, '1069': 59, '1068': 59, '1067': 59, '1066': 59, '1065': 59, '1064': 59, '1063': 59, '1062': 59, '1061': 59, '1060': 59, '1059': 59, '1058': 59, '1057': 59, '1056': 59, '1055': 59, '1054': 59, '1053': 59, '1052': 59, '1051': 59, '1050': 59, '1049': 59, '1048': 59, '1047': 59, '1046': 59, '1045': 59, '1044': 59, '1043': 59, '1042': 59, '1041': 59, '1040': 59, '1039': 59, '1038': 59, '1037': 59, '1036': 59, '1035': 59, '1034': 59, '1033': 59, '1032': 59, '1031': 59, '1030': 59, '1029': 59, '1028': 59, '1027': 59, '1026': 59, '1025': 59, '1024': 59, '1023': 59, '1022': 59, '1021': 59, '1020': 59, '1019': 59, '1018': 59, '1017': 59, '1016': 59, '1015': 59, '1014': 59, '1013': 59, '1012': 59, '1011': 59, '1010': 59, '1009': 59, '1008': 59, '1007': 59, '1006': 59, '1005': 59, '1004': 59, '1003': 59, '1002': 59, '1001': 59, '1000': 59, '999': 59, '998': 59, '997': 59, '996': 59, '995': 59, '994': 59, '993': 59, '992': 59, '991': 59, '990': 59, '989': 59, '988': 59, '987': 59, '986': 59, '985': 59, '984': 59, '983': 59, '982': 59, '981': 59, '980': 59, '979': 59, '978': 59, '977': 59, '976': 59, '975': 59, '974': 59, '973': 59, '972': 59, '971': 59, '970': 59, '969': 59, '968': 59, '967': 59, '966': 59, '965': 59, '964': 59, '963': 59, '962': 59

In [19]:

```
1 # Repeated words in each Label
2 positive_words = Counter()
3 neutral_words = Counter()
4 negative_words = Counter()
5
6 for index, row in df50.iterrows():
7     words = row['token']
8     label = row['label']
9     if label == 1:
10         positive_words.update(words)
11     elif label == 2:
12         negative_words.update(words)
13     else:
14         neutral_words.update(words)
15
16 #Resources:
17 #https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.iterrows.html
18 #https://www.kaggle.com/code/juicykn/imdb-movie-list-analysis-in-python-and-sql
```

In [20]:

```
1 # Most repeated words in each Label
2 top_positive_words = positive_words.most_common(10)
3 top_negative_words = negative_words.most_common(10)
4 top_neutral_words = neutral_words.most_common(10)
5
6 print('Positive: ', top_positive_words)
7 print('\n')
8 print('Negative: ', top_negative_words)
9 print('\n')
10 print('Neutral: ', top_neutral_words)
```

Positive: [('company', 508), ('eur', 241), ('said', 237), ('finland', 219), ('finnish', 215), ('million', 192), ('business', 190), ('group', 187), ('new', 179), ('sales', 163)]

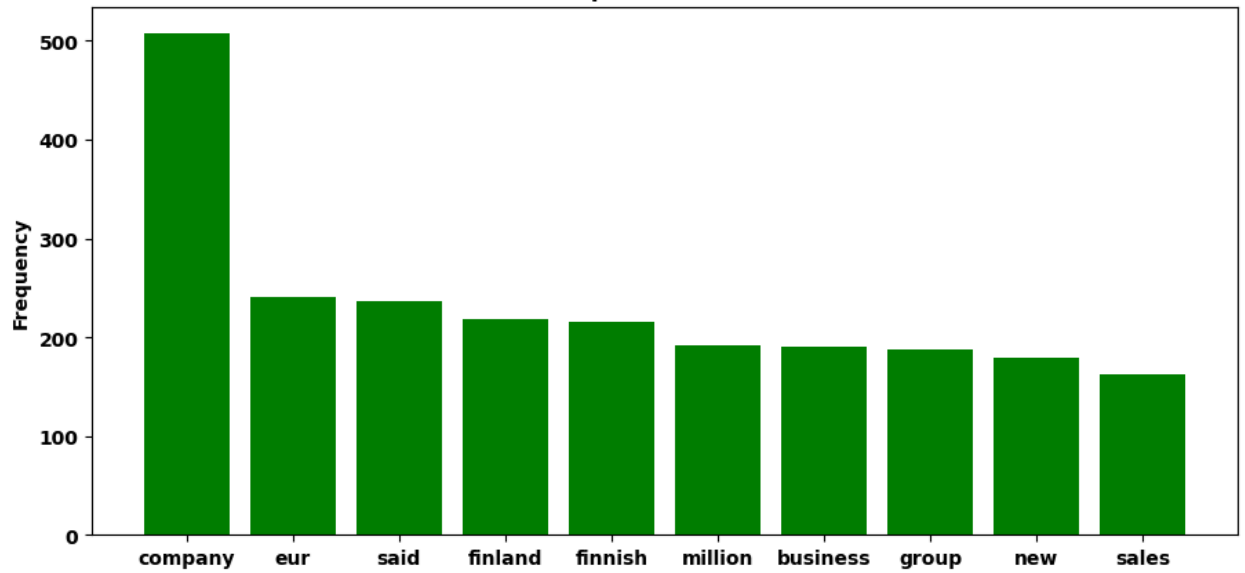
Negative: [('eur', 449), ('mn', 241), ('company', 240), ('said', 230), ('finnish', 198), ('net', 196), ('sales', 192), ('profit', 191), ('million', 170), ('period', 139)]

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

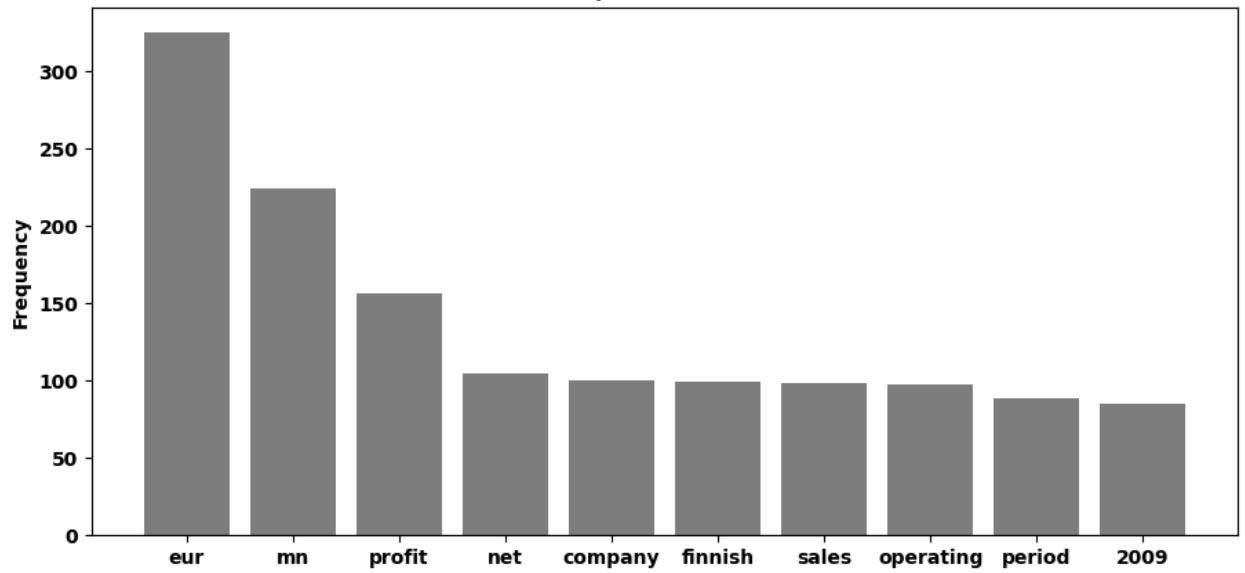
In [21]:

```
1 # Splitting the tuple we got earlier
2 positive_words, positive_counts = zip(*top_positive_words)
3 negative_words, negative_counts = zip(*top_negative_words)
4 neutral_words, neutral_counts = zip(*top_neutral_words)
5
6 # Charts-----
7 fig, axs = plt.subplots(3,1,figsize=(10,15))
8
9 # Positive words plot
10 axs[0].bar(positive_words, positive_counts, color='green')
11 axs[0].set_title('Most Frequent Positive Words')
12 axs[0].set_ylabel('Frequency')
13
14 # Negative words plot
15 axs[1].bar(negative_words, negative_counts, color='grey')
16 axs[1].set_title('Most Frequent Negative Words')
17 axs[1].set_ylabel('Frequency')
18
19 # Neutral words plot
20 axs[2].bar(neutral_words, neutral_counts, color='red')
21 axs[2].set_title('Most Frequent Neutral Words')
22 axs[2].set_ylabel('Frequency')
23
24 # Space between charts
25 plt.tight_layout(pad=4.0)
26 plt.show()
27
28 #Resources:
29 # https://realpython.com/python-zip-function/#using-zip-in-python
30 # https://matplotlib.org/stable/index.html
```

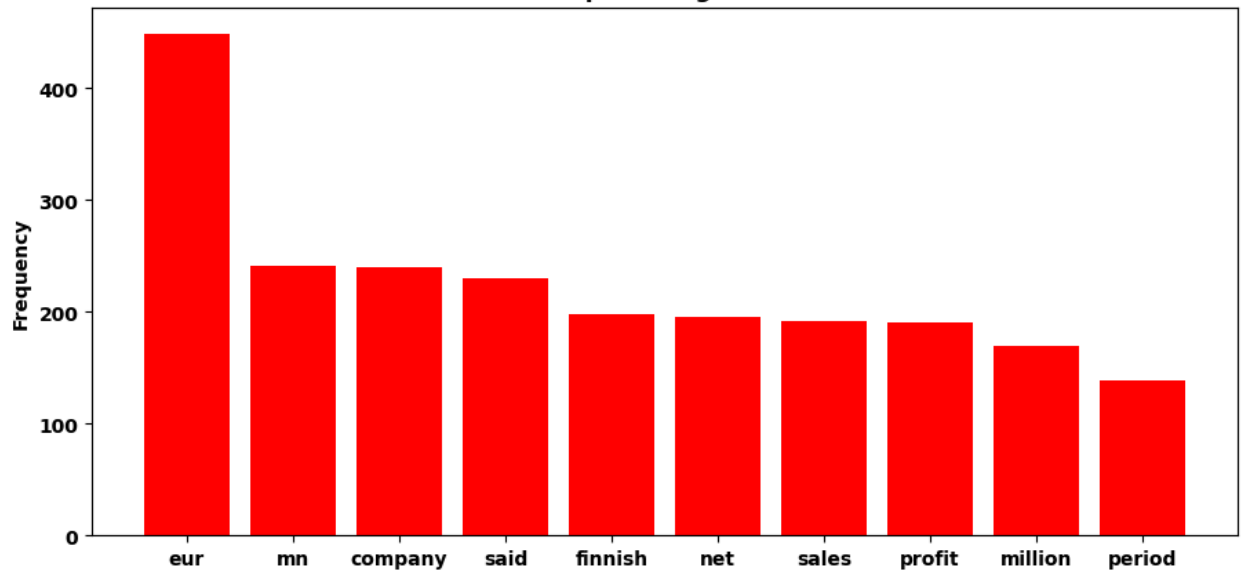
**Most Frequent Positive Words**



**Most Frequent Neutral Words**



**Most Frequent Negative Words**





In [22]: 1 # ) Splitting data

In [23]: 1 # Splitting data into train 70%, validation 15%, test 15%  
2  
3 X\_train\_val, X\_test, y\_train\_val, y\_test = train\_test\_split(df50['sentence\_preprocessed\_1'], df50['label'],  
4  
5 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 [ ]: 1

In [24]: 1 # Feature extraction: Transforming data into TF-IDF features.  
2 X\_train = tfidf\_vectorizer.fit\_transform(X\_train)  
3 X\_val = tfidf\_vectorizer.transform(X\_val)  
4 X\_test = tfidf\_vectorizer.transform(X\_test)

In [25]: 1 print(X\_train.shape) # Should output (number\_of\_samples, 33154)  
2 print(X\_test.shape) # Should also output (number\_of\_samples, 33154)  
3 print(X\_val.shape)

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

In [26]: 1 #Turning sparse matrix into dense  
2 X\_train = X\_train.toarray()  
3 X\_val = X\_val.toarray()  
4 X\_test = X\_test.toarray()  
5  
6 #Turning into PyTorch tensors  
7 X\_train = torch.tensor(X\_train, dtype=torch.float32)  
8 X\_val = torch.tensor(X\_val, dtype=torch.float32)  
9 X\_test = torch.tensor(X\_test, dtype=torch.float32)  
10 y\_train = torch.tensor(y\_train.values, dtype=torch.float32)  
11 y\_val = torch.tensor(y\_val.values, dtype=torch.float32)  
12 y\_test = torch.tensor(y\_test.values, dtype=torch.float32)  
13  
14 #Resources:  
15 # <https://pytorch.org/docs/stable/tensors.html>

In [38]: 1 class SimpleMLPModel(nn.Module):  
2 def \_\_init\_\_(self):  
3 super(SimpleMLPModel, self).\_\_init\_\_()  
4 self.fc = nn.Linear(9185, 1)  
5 self.sigmoid = nn.Sigmoid()  
6  
7 def forward(self, x):  
8 x = self.fc(x)  
9 x = self.sigmoid(x)  
10 return x  
11  
12 # Model set on ev mode.  
13 model.eval()  
14  
15 # Single forward pass  
16 with torch.no\_grad():  
17 outputs = model(X\_test)  
18  
19 # threshold to classify pb  
20 threshold = 0.5  
21 predicted\_labels = (outputs > threshold).float() # Convert probabilities to 0 or 1 based on the threshold  
22  
23 # Number of correct predictions  
24 correct\_predictions = (predicted\_labels.squeeze() == y\_test).float().sum()  
25  
26 # Accuracy  
27 accuracy = correct\_predictions / y\_test.shape[0]  
28 print(f'Accuracy: {accuracy.item():.2f}')

Accuracy: 0.48

```

In [ ]: 1 # Multilayer perceptron working environment.
2 # Getting ready the work environment. Importing Libraries and modules:
3 import time
4 import pandas as pd
5 import re
6 import nltk
7 import torch
8 import torch.nn as nn
9 import numpy as np
10 import string
11 import matplotlib.pyplot as plt
12 import seaborn as sns
13
14 from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
15 from sklearn.model_selection import train_test_split
16 from sklearn.metrics import precision_recall_fscore_support, classification_report, roc_curve, roc_auc_score
17 from collections import Counter
18 from bs4 import BeautifulSoup
19 from nltk.corpus import stopwords
20 from nltk.tokenize import word_tokenize
21
22 #===== Extra tools for the statistic analysis =====
23 from nltk.stem import WordNetLemmatizer
24
25 lemmatizer = WordNetLemmatizer()
26 #-----
27
28 stop_words = stopwords.words('english')
29 tfidf_vectorizer = TfidfVectorizer()
30 vectorizer = CountVectorizer()

```

```

In [ ]: 1 # Getting ready the work environment. Importing Libraries and modules:
2 import seaborn as sns
3 import matplotlib.pyplot as plt
4 import pandas as pd
5 import re
6 import nltk
7 import string
8
9 from sklearn.naive_bayes import MultinomialNB
10 from sklearn.model_selection import train_test_split, GridSearchCV
11 from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
12 from sklearn.linear_model import LogisticRegression
13 from sklearn.metrics import accuracy_score, precision_recall_fscore_support
14 from sklearn.metrics import classification_report, roc_curve, roc_auc_score, confusion_matrix
15 from nltk.corpus import stopwords
16 from nltk.tokenize import word_tokenize
17 from collections import Counter
18 from bs4 import BeautifulSoup
19
20
21 #===== Extra tools for the statistic analysis =====
22 from nltk.stem import WordNetLemmatizer
23
24 lemmatizer = WordNetLemmatizer()
25 #-----
26
27 stop_words = stopwords.words('english')
28 tfidf_vectorizer = TfidfVectorizer(stop_words='english')
29 vectorizer = CountVectorizer()

```

```

In [ ]: 1 # Importing dataset from the hugging face
2 from datasets import load_dataset
3
4 dataset1 = load_dataset('financial_phrasebank', 'sentences_50agree')
5 dataset2 = load_dataset('financial_phrasebank', 'sentences_66agree')
6 dataset3 = load_dataset('financial_phrasebank', 'sentences_75agree')

```

```

In [ ]: 1 # Checking dataset
2 print(dataset1)

```

```

In [ ]: 1 # Transforming the data set into a more friendly frame (tables)
2 df50 = pd.DataFrame(dataset1['train'])
3 df66 = pd.DataFrame(dataset2['train'])
4 df75 = pd.DataFrame(dataset3['train'])

```

```
In [ ]: 1 # Checking data
2 print(df50)
3 print("\n")
4 print(df66)
5 print("\n")
6 print(df75)
```

```
In [ ]: 1 # Checking the data we will preprocess
2 print(df50['sentence'])
```

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

```
In [ ]: 1 # Function to remove HTML tags:
2 def remove_html(text):
3     soup = BeautifulSoup(text, "html.parser")
4     return soup.get_text()
5
6 #Sources:
7 #https://stackoverflow.com/questions/328356/extracting-text-from-html-file-using-python?newreg=aa9f4dc4aea34
8 #https://beautiful-soup-4.readthedocs.io/en/latest/
9 #https://www.datacamp.com/tutorial/web-scraping-using-python
10 #https://www.geeksforgeeks.org/how-to-write-the-output-to-html-file-with-python-beautifulsoup/
```

```
In [ ]: 1 def remove_urls(text):
2     return re.sub(r'https?:\/\/\S+|www\.\S+', '', text)
3
4 #Source: https://www.geeksforgeeks.org/remove-urls-from-string-in-python
```

```
In [ ]: 1 def remove_punctuation(text):
2     return text.translate(str.maketrans('', '', string.punctuation))
3
4 #Source: https://stackoverflow.com/questions/34293875/how-to-remove-punctuation-marks-from-a-string-in-python
```

```
In [ ]: 1 # Function to remove HTML tags:
2 def remove_html(text):
3     soup = BeautifulSoup(text, "html.parser")
4     return soup.get_text()
5
6 #Sources:
7 #https://stackoverflow.com/questions/328356/extracting-text-from-html-file-using-python?newreg=aa9f4dc4aea34
8 #https://beautiful-soup-4.readthedocs.io/en/latest/
9 #https://www.datacamp.com/tutorial/web-scraping-using-python
10 #https://www.geeksforgeeks.org/how-to-write-the-output-to-html-file-with-python-beautifulsoup/
```

```
In [ ]: 1 # Function to put together all the previous functions:
2 def preprocess_1(text):
3     text = remove_html(text)
4     text = remove_urls(text)
5     text = remove_punctuation(text)
6     text = text.lower()
7     return text
8
9 df50['sentence_preprocessed_1'] = df50['sentence'].apply(preprocess_1)
```

```
In [ ]: 1 # Function to tokenize and convert to lower case the text in review column
2 def tokenize(text):
3     tokens = re.findall(r'\b\w+\b', text)
4     return tokens
5
6 #Tokenization
7 df50['token'] = df50['sentence_preprocessed_1'].apply(tokenize)
```

```
In [ ]: 1 # Function to remove stop words from the tokenized review column
2 def remove_stopwords(tokens):
3     filtered_tokens = [word for word in tokens if word not in stop_words]
4     return filtered_tokens
5
6 #Remove stopwords
7 df50['token'] = df50['token'].apply(remove_stopwords)
```

```
In [ ]: 1 # Calculating the total tokens for each review
2 df50['token_count'] = df50['token'].apply(lambda x: len(x) if isinstance(x, list) else 0)
3
4 # Dispersion and central tendency measurements
5 statistics = df50.groupby('label')['token_count'].agg(['min', 'max', 'mean', 'var', 'std'])
6
7 # Avg words per review:
8 avg_words = df50['token'].apply(len).mean()
9
10 #Print the statistics
11 print("Statistics by Label: ")
12 print('\n')
13 print(statistics)
14 print('\n')
15 print('\n')
16 print('Average Words: ', f"{avg_words:.0f}")
17
18 #Resources:
19 #https://www.geeksforgeeks.org/pandas-groupby-one-column-and-get-mean-min-and-max-values/
20 #https://www.kaggle.com/code/akshayseghal/ultimate-guide-to-pandas-groupby-aggregate
```

```
In [ ]: 1 # Iterating through the List of Lists(each row) to create a new List with all the tokens
2 def word_freq(list_of_list):
3     single_list = [item for sublist in list_of_list for item in sublist]
4     token_freq = Counter(single_list)
5     return token_freq
6
7 # Counting the frequency for each word.
8 word_frequency = word_freq(df50['token'])
9 print(word_frequency)
10
11 #Sources: https://www.datacamp.com/tutorial/pandas-apply
```

```
In [ ]: 1 # Unique words
2 unique_words = len(word_frequency.keys())
3 print('Unique_words: ', f'{unique_words}')
```

```
In [ ]: 1 # Repeated words in each Label
2 positive_words = Counter()
3 neutral_words = Counter()
4 negative_words = Counter()
5
6 for index, row in df50.iterrows():
7     words = row['token']
8     label = row['label']
9     if label == 1:
10         positive_words.update(words)
11     elif label == 2:
12         negative_words.update(words)
13     else:
14         neutral_words.update(words)
15
16 #Resources:
17 #https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.iterrows.html
18 #https://www.kaggle.com/code/juicykn/imdb-movie-list-analysis-in-python-and-sql
```

```
In [ ]: 1 # Most repeated words in each Label
2 top_positive_words = positive_words.most_common(10)
3 top_negative_words = negative_words.most_common(10)
4 top_neutral_words = neutral_words.most_common(10)
5
6 print('Positive: ', top_positive_words)
7 print('\n')
8 print('Negative: ', top_negative_words)
9 print('\n')
10 print('Neutral: ', top_neutral_words)
```

```
In [ ]: 1 # Splitting the tuple we got earlier
2 positive_words, positive_counts = zip(*top_positive_words)
3 negative_words, negative_counts = zip(*top_negative_words)
4 neutral_words, neutral_counts = zip(*top_neutral_words)
5
6 # Charts-----
7 fig, axs = plt.subplots(3,1,figsize=(10,15))
8
9 # Positive words plot
10 axs[0].bar(positive_words, positive_counts, color='green')
11 axs[0].set_title('Most Frequent Positive Words')
12 axs[0].set_ylabel('Frequency')
13
14 # Negative words plot
15 axs[1].bar(negative_words, negative_counts, color='grey')
16 axs[1].set_title('Most Frequent Negative Words')
17 axs[1].set_ylabel('Frequency')
18
19 # Neutral words plot
20 axs[2].bar(neutral_words, neutral_counts, color='red')
21 axs[2].set_title('Most Frequent Neutral Words')
22 axs[2].set_ylabel('Frequency')
23
24 # Space between charts
25 plt.tight_layout(pad=4.0)
26 plt.show()
27
28 #Resources:
29 # https://realpython.com/python-zip-function/#using-zip-in-python
30 # https://matplotlib.org/stable/index.html
```

```
In [ ]: 1 # Splitting data into train 70%, validation 15%, test 15%
2
3 X_train_val, X_test, y_train_val, y_test = train_test_split(df50['sentence_preprocessed_1'], df50['label'],
4
5 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 [ ]: 1 # Feature extraction: Transforming data into TF-IDF features.
2 X_train = tfidf_vectorizer.fit_transform(X_train)
3 X_val = tfidf_vectorizer.transform(X_val)
4 X_test = tfidf_vectorizer.transform(X_test)
```

```
In [ ]: 1 print(X_train.shape) # Should output (number_of_samples, 33154)
2 print(X_test.shape) # Should also output (number_of_samples, 33154)
3 print(X_val.shape)
```

```
In [ ]: 1 #Turning sparse matrix into dense
2 X_train = X_train.toarray()
3 X_val = X_val.toarray()
4 X_test = X_test.toarray()
5
6 #Turning into PyTorch tensors
7 X_train = torch.tensor(X_train, dtype=torch.float32)
8 X_val = torch.tensor(X_val, dtype=torch.float32)
9 X_test = torch.tensor(X_test, dtype=torch.float32)
10 y_train = torch.tensor(y_train.values, dtype=torch.float32)
11 y_val = torch.tensor(y_val.values, dtype=torch.float32)
12 y_test = torch.tensor(y_test.values, dtype=torch.float32)
13
14 #Resources:
15 # https://pytorch.org/docs/stable/tensors.html
```

```
In [ ]: 1 # Time consumed (starts)
2 start_time = time.time()
3
4 # Building the Multilayer Perceptron model with back propagation.
5 class MLPmodel(nn.Module):
6     def __init__(self):
7         super(MLPmodel, self).__init__()
8         self.fc1 = nn.Linear(9185,610)
9         self.fc2 = nn.Linear(610,377)
10        self.fc3 = nn.Linear(377,23)
11        self.fc4 = nn.Linear(23,1)
12        self.sigmoid = nn.Sigmoid()
13        self.relu = nn.ReLU()
14
15    def forward(self, x):
16        hidden = self.relu(self.fc1(x))
17        hidden = self.relu(self.fc2(hidden))
18        hidden = self.relu(self.fc3(hidden))
19        output = self.sigmoid(self.fc4(hidden))
20        return output
```

```
In [ ]: 1 # Model environment
2 model = MLPmodel() # Define the model
3 criterion = nn.CrossEntropyLoss()
4 optimizer = torch.optim.Adam(model.parameters(), lr=0.0001, weight_decay=0.00001) # Using Adam optimizer
5
6 #Interesting combinations: 0.1/0.00001
7 #0.1/ 0
8 #Is there correlation? what is the relationship?
9
10 #Note: SDG was used earlier in during the experimentation, however, the performance was way worst than the c
11 #Different arguments were used with the different parameters and anything changed barely.
```

```

In [ ]: 1 # Training with early stopping
2 epochs = 5000
3 patience = 10 # Here we define how many epochs wait until we stop
4 best_val_loss = float('inf') #To save the best model/early stop
5 patience_counter = 0 #This one starts a counter to track number of epochs without improvement
6
7 for epoch in range(epochs):
8     model.train()
9     optimizer.zero_grad()
10    #Forward Pass
11    outputs = model(X_train)
12    loss = criterion(outputs.squeeze(), y_train)
13    #Backward and Optimize
14    loss.backward()
15    optimizer.step()
16
17    #Validation
18    model.eval()
19    with torch.no_grad():
20        val_outputs = model(X_val)
21        val_loss = criterion(val_outputs.squeeze(), y_val)
22
23    #Early stop
24    if val_loss < best_val_loss:
25        best_val_loss = val_loss
26        best_model_state = model.state_dict() # Save the best model state
27        patience_counter=0
28    else:
29        patience_counter += 1
30
31    #Print the early stopping
32    if patience_counter > patience:
33        print(f'Stopping early at epoch {epoch+1}.')
34        break
35
36    if (epoch+1) % 5 == 0:
37        print(f'Epoch {epoch+1}/{epochs}, Loss: {loss.item()}, Val Loss: {val_loss.item()}')
38
39 if best_model_state:
40     model.load_state_dict(best_model_state)
41
42 #Resources:
43 #https://pythonguides.com/pytorch-early-stopping/
44 #https://discuss.pytorch.org/t/can-i-deepcopy-a-model/52192
45 #https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early
46 #https://stackoverflow.com/questions/71998978/early-stopping-in-pytorch
47 #https://github.com/Bjarten/early-stopping-pytorch
48 #https://debuggercafe.com/using-Learning-rate-scheduler-and-early-stopping-with-pytorch/

```

```

In [ ]: 1 #Accuracy on validation set
2 model.eval()
3 with torch.no_grad():
4     val_outputs = model(X_val)
5     val_predicted_bin = (val_outputs.squeeze() > 0.5).int()
6
7     #validation accuracy
8     val_accuracy = accuracy_score(y_val.numpy(), val_predicted_bin.numpy())
9     print(f'Validation set accuracy: {val_accuracy}')
10
11 # Resources:
12 # https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html
13 # https://pytorch.org/docs/stable/generated/torch.Tensor.numpy.html
14 # https://www.youtube.com/watch?v=E35CVhVKISA&t=135s

```

```

In [ ]: 1 #Accuracy on test set
2 model.eval()
3
4 with torch.no_grad():
5     test_outputs = model(X_test)
6     predicted_bin = (test_outputs.squeeze() > 0.5).int()
7
8 #accuracy
9 test_accuracy = accuracy_score(y_test.numpy(), predicted_bin.numpy())
10 print(f'Accuracy on test set: {test_accuracy:.2f}')
11
12 #classification report
13 print(classification_report(y_test.numpy(), predicted_bin.numpy()))

```

```
In [ ]: 1 # Total Time Consumed
2 end_time = time.time()
3 execution_time = end_time - start_time
4 print(f"Total Execution Time: {execution_time} seconds")
```

```
In [ ]: 1 import matplotlib.pyplot as plt
2 import seaborn as sns
3 from sklearn.metrics import confusion_matrix
4
5 # Calculate the confusion matrix
6 cm = confusion_matrix(y_test.numpy(), predicted_bin.numpy())
7
8 # Plot the confusion matrix using seaborn heatmap
9 plt.figure(figsize=(8, 6))
10 sns.heatmap(cm, annot=True, fmt='d', cmap='seismic', cbar=False,
11             xticklabels=['Negative', 'Neutral', 'Positive'],
12             yticklabels=['Negative', 'Neutral', 'Positive'])
13 plt.xlabel('Predicted Labels')
14 plt.ylabel('True Labels')
15 plt.title('Confusion Matrix')
16 plt.tight_layout()
17 plt.show()
```

```
In [ ]: 1 # Getting ready the work environment. Importing libraries and modules:
2
3 import pandas as pd
4 import numpy as np
5 import matplotlib.pyplot as plt
6 import nltk
7 import re
8 import seaborn as sns
9 import string
10 import time
11
12 from bs4 import BeautifulSoup
13 from collections import Counter
14 from nltk.corpus import stopwords
15 from nltk.tokenize import word_tokenize
16 from sklearn.naive_bayes import MultinomialNB
17 from sklearn.model_selection import train_test_split, GridSearchCV
18 from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
19 from sklearn.linear_model import LogisticRegression
20 from sklearn.metrics import accuracy_score, precision_recall_fscore_support
21 from sklearn.metrics import classification_report, roc_curve, roc_auc_score, confusion_matrix
22
23 #===== Extra tools for the statistic analysis =====
24 from nltk.stem import WordNetLemmatizer
25
26 lemmatizer = WordNetLemmatizer()
27 #-----
28
29 stop_words = stopwords.words('english')
30 tfidf_vectorizer = TfidfVectorizer(stop_words='english')
31 vectorizer = CountVectorizer()
```

```
In [ ]: 1 # Time consumed (starts)
2 start_time = time.time()
3
4 # Hyperparameters
5 param_grid = {'alpha': np.logspace(10, 15, 5),
6              'fit_prior' : [True],
7              'class_prior':[None, [0.01, 0.15, 0.6, 1]],
8              }
9
10 # Training the model and looking for the best combination
11 grid_search = GridSearchCV(MultinomialNB(), param_grid, cv=3, verbose=2)
12 grid_search.fit(X_train, y_train)
13
14 # Getting the best estimator#####
15 Best_NBmodel = grid_search.best_estimator_
16 Best_score = grid_search.best_score_
17
18 # Print results
19 print(f'Best Model: {Best_NBmodel}\n')
20 #print('\n')
21 print(f'Best CV Score: {Best_score}')
```



```
In [ ]: 1 # Evaluation on the Validation
2 y_pred_val = Best_NBmodel.predict(X_val)
3 val_accuracy = accuracy_score(y_val, y_pred_val)
4
5 # Print validation results
6 print(f'Validation Accuracy: {val_accuracy}')
```

```
In [ ]: 1 # Evaluation on the test set
2 y_pred_test = Best_NBmodel.predict(X_test)
3 test_accuracy = accuracy_score(y_test, y_pred_test)
4
5 # Print test results
6 print(f'Accuracy on test set: {test_accuracy}')
```

```
In [ ]: 1 # Predicting Labels for the test
2 y_pred_test = Best_NBmodel.predict(X_test)
3
4 # Confusion matrix
5 cm = confusion_matrix(y_test, y_pred_test)
6
7 # Plot the confusion matrix using seaborn heatmap
8 plt.figure(figsize=(8, 6))
9
10 sns.heatmap(cm,
11             annot=True,
12             fmt='d',
13             cmap='seismic',
14             cbar=False,
15             xticklabels=['Negative', 'Neutral', 'Positive'],
16             yticklabels=['Negative', 'Neutral', 'Positive'])
17
18 plt.title('Confusion Matrix')
19 plt.xlabel('Predicted Labels')
20 plt.ylabel('True Labels')
21
22 plt.tight_layout()
23
24 plt.show()
```

```
In [ ]: 1 # Total Time Consumed
2 end_time = time.time()
3 execution_time = end_time - start_time
4 print(f"Total Execution Time: {execution_time} seconds")
```

```
In [ ]: data into train 70%, validation 15%, test 15%
2
, X_test, y_train_val, y_test = train_test_split(df50['sentence_preprocessed_1'], df50['label'], test_size=0.15,
4
val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.15, random_state=42)
```

```
In [ ]: 1 # Feature extraction: Transforming data into TF-IDF features.
2 X_train = tfidf_vectorizer.fit_transform(X_train)
3 X_val = tfidf_vectorizer.transform(X_val)
4 X_test = tfidf_vectorizer.transform(X_test)
```

```
In [ ]: 1 # Checking shape
2 print(X_train.shape)
3 print(X_test.shape)
4 print(X_val.shape)
```

```
In [ ]: 1 # Time consumed (starts)
2 start_time = time.time()
3
4 # Multinomial Naive Bayes
5 model = MultinomialNB()
6
7 # Fit the model to the training data
8 model.fit(X_train, y_train)
9
10 # Predict
11 y_pred = model.predict(X_test)
12
13 accuracy = accuracy_score(y_test, y_pred)
14
15 # Print the accuracy
16 print(f"Accuracy: {accuracy:.2f}")
17
18 # Total Time Consumed
19 end_time = time.time()
20 execution_time = end_time - start_time
21 print(f"Total Execution Time: {execution_time} seconds")
```

```
In [ ]: 1 # Total Time Consumed
2 end_time = time.time()
3 execution_time = end_time - start_time
4 print(f"Total Execution Time: {execution_time} seconds")
```

```
In [ ]: 1 # Support Vector Machine working environment.
2 # Getting ready the work environment. Importing Libraries and modules:
3
4 import time
5 import pandas as pd
6 import re
7 import nltk
8 import torch
9 import torch.nn as nn
10 import numpy as np
11 import string
12 import matplotlib.pyplot as plt
13 import seaborn as sns
14
15 from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
16 from sklearn.svm import SVC
17 from sklearn.metrics import roc_curve, roc_auc_score, confusion_matrix
18 from sklearn.metrics import accuracy_score, precision_recall_fscore_support, classification_report
19 from sklearn.model_selection import train_test_split, GridSearchCV
20 from collections import Counter
21 from bs4 import BeautifulSoup
22 from nltk.corpus import stopwords
23 from nltk.tokenize import word_tokenize
24
25 #===== Extra tools for the statistic analysis =====
26 from nltk.stem import WordNetLemmatizer
27
28 lemmatizer = WordNetLemmatizer()
29 #-----
30
31 stop_words = stopwords.words('english')
32 tfidf_vectorizer = TfidfVectorizer(stop_words='english')
33 vectorizer = CountVectorizer()
```

```
In [ ]: data into train 70%, validation 15%, test 15%
2
, X_test, y_train_val, y_test = train_test_split(df50['sentence_preprocessed_1'], df50['label'], test_size=0.15,
4
val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.15, random_state=42)
```

```
In [ ]: 1 # Feature extraction: Transforming data into TF-IDF features.
2 X_train = tfidf_vectorizer.fit_transform(X_train)
3 X_val = tfidf_vectorizer.transform(X_val)
4 X_test = tfidf_vectorizer.transform(X_test)
```

```
In [ ]: 1 # Checking shape
2 print(X_train.shape)
3 print(X_test.shape)
4 print(X_val.shape)
```

```
In [ ]: 1 #Turning sparse matrix into dense
2 X_train = X_train.toarray()
3 X_val = X_val.toarray()
4 X_test = X_test.toarray()
5
6 #Turning into PyTorch tensors
7 X_train = torch.tensor(X_train, dtype=torch.float32)
8 X_val = torch.tensor(X_val, dtype=torch.float32)
9 X_test = torch.tensor(X_test, dtype=torch.float32)
10 y_train = torch.tensor(y_train.values, dtype=torch.float32)
11 y_val = torch.tensor(y_val.values, dtype=torch.float32)
12 y_test = torch.tensor(y_test.values, dtype=torch.float32)
13
14 #Resources:
15 # https://pytorch.org/docs/stable/tensors.html
```

```
In [ ]: 1 # Time Consumed (starts)
2 start_time = time.time()
3
4 # Definition of the SVM model and hyperparameter for tuning on the training set
5 param_grid = {'C': np.logspace(-15, 15, 2), 'kernel': ['rbf'], 'tol' : [1e-3]}
6
7 # Hyperparameters tuning, cross-validation using training set.
8 grid_search = GridSearchCV(SVC(), param_grid, cv=2, scoring='accuracy', verbose=2)
9 grid_search.fit(X_train, y_train)
10
11 # Getting the best estimator
12 Best_SVMmodel = grid_search.best_estimator_
13 Best_score = grid_search.best_score_
14
15 # Print results
16 print(f'Best Model: {Best_SVMmodel}')
17 print('\n')
18 print(f'Best CV Score: {Best_score}')
```

```
In [ ]: 1 # Evaluation of the model on the validation set with the best parameters
2 y_pred = Best_SVMmodel.predict(X_val)
3 val_accuracy = accuracy_score(y_val, y_pred)
4
5 # Print results
6 print(f'Validation set accuracy: {val_accuracy}')
```

```
In [ ]: 1 # Evaluation of the final model using the test set
2 y_pred = Best_SVMmodel.predict(X_test)
3 test_accuracy = accuracy_score(y_test, y_pred)
4
5 #Print results
6 print(f'Accuracy on test set: {test_accuracy}')
```

```
7 print(classification_report(y_test, y_pred))
```

```
In [ ]: 1 # Predicting Labels for the test
2 y_pred_test = Best_SVMmodel.predict(X_test)
3
4 # Confusion matrix
5 cm = confusion_matrix(y_test, y_pred_test)
6
7 # Plot the confusion matrix using seaborn heatmap
8 plt.figure(figsize=(8, 6))
9
10 sns.heatmap(cm,
11             annot=True,
12             fmt='d',
13             cmap='seismic',
14             cbar=False,
15             xticklabels=['Negative', 'Neutral', 'Positive'],
16             yticklabels=['Negative', 'Neutral', 'Positive'])
17
18 plt.title('Confusion Matrix')
19 plt.xlabel('Predicted Labels')
20 plt.ylabel('True Labels')
21
22
23 plt.show()
```

```
In [ ]: 1 # Total Time Consumed
2 end_time = time.time()
3 execution_time = end_time - start_time
4 print(f"Total Execution Time: {execution_time} seconds")
```

```
In [ ]: 1 # Support Vector Machine working environment.
2 # Getting ready the work environment. Importing Libraries and modules:
3
4 import time
5 import pandas as pd
6 import re
7 import nltk
8 import torch
9 import torch.nn as nn
10 import numpy as np
11 import string
12 import matplotlib.pyplot as plt
13 import seaborn as sns
14
15 from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
16 from sklearn.svm import SVC
17 from sklearn.metrics import roc_curve, roc_auc_score, confusion_matrix
18 from sklearn.metrics import accuracy_score, precision_recall_fscore_support, classification_report
19 from sklearn.model_selection import train_test_split, GridSearchCV
20 from collections import Counter
21 from bs4 import BeautifulSoup
22 from nltk.corpus import stopwords
23 from nltk.tokenize import word_tokenize
24
25 #===== Extra tools for the statistic analysis =====
26 from nltk.stem import WordNetLemmatizer
27
28 lemmatizer = WordNetLemmatizer()
29 #-----
30
31 stop_words = stopwords.words('english')
32 tfidf_vectorizer = TfidfVectorizer(stop_words='english')
33 vectorizer = CountVectorizer()
```

```
In [ ]: data into train 70%, validation 15%, test 15%
2
, X_test, y_train_val, y_test = train_test_split(df50['sentence_preprocessed_1'], df50['label'], test_size=0.15,
4
val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.15, random_state=42)
```

```
In [ ]: 1 # Feature extraction: Transforming data into TF-IDF features.
2 X_train = tfidf_vectorizer.fit_transform(X_train)
3 X_val = tfidf_vectorizer.transform(X_val)
4 X_test = tfidf_vectorizer.transform(X_test)
```

```
In [ ]: 1 # Checking shape
2 print(X_train.shape)
3 print(X_test.shape)
4 print(X_val.shape)
```

```
In [ ]: 1 #Turning sparse matrix into dense
2 X_train = X_train.toarray()
3 X_val = X_val.toarray()
4 X_test = X_test.toarray()
5
6 #Turning into PyTorch tensors
7 X_train = torch.tensor(X_train, dtype=torch.float32)
8 X_val = torch.tensor(X_val, dtype=torch.float32)
9 X_test = torch.tensor(X_test, dtype=torch.float32)
10 y_train = torch.tensor(y_train.values, dtype=torch.float32)
11 y_val = torch.tensor(y_val.values, dtype=torch.float32)
12 y_test = torch.tensor(y_test.values, dtype=torch.float32)
13
14 #Resources:
15 # https://pytorch.org/docs/stable/tensors.html
```

```
In [ ]: 1 # Time Consumed (starts)
2 start_time = time.time()
3
4 # Initialize the Support Vector Machine classifier with default parameters
5 svm_model = SVC()
6
7 # Fit the model
8 svm_model.fit(X_train, y_train)
9
10 # Predict the labels
11 y_pred = svm_model.predict(X_test)
12
13 # Calculate the accuracy
14 accuracy = accuracy_score(y_test, y_pred)
15
16 # Print the accuracy
17 print(f"Accuracy: {accuracy:.2f}")
18
19
20 # Total Time Consumed
21 end_time = time.time()
22 execution_time = end_time - start_time
23 print(f"Total Execution Time: {execution_time} seconds")
```

```
In [ ]: 1 #!pip install transformers[torch]
2 #!pip install torch transformers datasets
3 #!pip install accelerate -U
4
5 # Getting ready the work environment. Importing libraries and modules:
6
7 import matplotlib.pyplot as plt
8 import numpy as np
9 import seaborn as sns
10 import torch
11 import time
12 import pandas as pd
13
14 from datasets import load_dataset
15 from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
16 from transformers import BertTokenizer, BertForSequenceClassification, Trainer, TrainingArguments
```

```
In [ ]: 1 # Time consumed (starts)
2 start_time = time.time()
3
4 # Loading the pre-trained FinBERT model and tokenizer
5 model_ = "ProsusAI/finbert"
6 tokenizer = BertTokenizer.from_pretrained(model_)
7 model = BertForSequenceClassification.from_pretrained(model_)
8
9 # We directly load the Financial PhraseBank dataset from The Hugging Face
10 dataset = load_dataset("financial_phrasebank", "sentences_allagree") # https://huggingface.co/datasets/financial_phrasebank
11 # "sentences_50agree"
12 # "sentences_66agree"
13 # "sentences_75agree"
```

```
In [ ]: 1 # Preprocess the dataset
2 def preprocess_function(examples):
3     return tokenizer(examples["sentence"],
4                       padding="longest",
5                       max_length= 5,
6                       truncation = True)
7
8 dataset = dataset.map(preprocess_function, batched=True)
9
10 # Split the dataset into training and validation sets
11 train_test_split = dataset['train'].train_test_split(test_size=0.2, seed=42)
12 train_dataset = train_test_split['train']
13 eval_dataset = train_test_split['test']
14
15
```

```
In [ ]: 1 # Training arguments
2 training_args = TrainingArguments(
3     output_dir = "./results",
4     evaluation_strategy = "epoch",
5     learning_rate = 1,
6     per_device_train_batch_size = 10,
7     per_device_eval_batch_size = 10,
8     num_train_epochs = 1,
9     warmup_steps = 200,
10    save_strategy = "epoch"
11 )
12
13 # Define the trainer
14 trainer = Trainer(
15     model = model,
16     args = training_args,
17     train_dataset = train_dataset,
18     eval_dataset = eval_dataset,
19 )
20
21 # Fine-tune the model
22 trainer.train()
```

```
In [ ]: 1 # Evaluate the fine-tuned model
2 predictions = trainer.predict(eval_dataset)
3 preds = np.argmax(predictions.predictions, axis=1)
4 labels = eval_dataset["label"]
5 accuracy = accuracy_score(labels, preds)
6 report = classification_report(labels, preds)
7
8 print(f"Validation Accuracy: {accuracy:.2f}")
9 print(f"Classification Report:\n{report}")
```

```
In [ ]: 1 # Confusion matrix
2 cm = confusion_matrix(labels, preds)
3
4 # Plot the confusion matrix using seaborn heatmap
5 plt.figure(figsize=(8, 6))
6
7 sns.heatmap(cm,
8             annot=True,
9             fmt='d',
10             cmap='seismic',
11             cbar=False,
12             xticklabels=['Negative', 'Neutral', 'Positive'],
13             yticklabels=['Negative', 'Neutral', 'Positive'])
14
15 plt.title('Confusion Matrix')
16 plt.xlabel('Predicted Labels')
17 plt.ylabel('True Labels')
18
19
20 plt.show()
```

```
In [ ]: 1 # Load the test dataset
2 test_dataset = load_dataset("financial_phrasebank", "sentences_75agree")
3 test_dataset = test_dataset.map(preprocess_function, batched=True)
4
5 # Evaluate the fine-tuned model on the test set
6 test_predictions = trainer.predict(test_dataset['train'])
7 test_preds = np.argmax(test_predictions.predictions, axis=1)
8 test_labels = test_dataset['train']["label"]
9 test_accuracy = accuracy_score(test_labels, test_preds)
10 test_report = classification_report(test_labels, test_preds)
11
```

```
In [ ]: 1 print(f"Test Accuracy: {test_accuracy:.2f}")
2 print(f"Test Classification Report:\n{test_report}")
```

```
In [ ]: 1 # Confusion matrix
2 test_cm = confusion_matrix(test_labels, test_preds)
3
4 # Plot the confusion matrix using seaborn heatmap
5 plt.figure(figsize=(8, 6))
6
7 sns.heatmap(test_cm,
8             annot=True,
9             fmt='d',
10            cmap='seismic',
11            cbar=False,
12            xticklabels=['Negative', 'Neutral', 'Positive'],
13            yticklabels=['Negative', 'Neutral', 'Positive'])
14
15 plt.title('Confusion Matrix')
16 plt.xlabel('Predicted Labels')
17 plt.ylabel('True Labels')
18
19
20 plt.show()
```

```
In [ ]: 1 # Total Time Consumed
2 end_time = time.time()
3 execution_time = end_time - start_time
4 print(f"Total Execution Time: {execution_time} seconds")
```

```
In [ ]: 1 #!pip install transformers[torch]
2 #!pip install torch transformers datasets
3 #!pip install accelerate -U
4
5 # Getting ready the work environment. Importing Libraries and modules:
6
7 import matplotlib.pyplot as plt
8 import numpy as np
9 import seaborn as sns
10 import torch
11 import time
12 import pandas as pd
13
14 from datasets import load_dataset
15 from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
16 from transformers import BertTokenizer, BertForSequenceClassification, Trainer, TrainingArguments
```

```
In [ ]: 1 # Time consumed (starts)
2 start_time = time.time()
3
4 # Loading the pre-trained FinBERT model and tokenizer
5 model_ = "ProsusAI/finbert"
6 tokenizer = BertTokenizer.from_pretrained(model_)
7 model = BertForSequenceClassification.from_pretrained(model_)
8
9 # We directly load the Financial PhraseBank dataset from The Hugging Face
10 dataset = load_dataset("financial_phrasebank", "sentences_allagree") # https://huggingface.co/datasets/financial_phrasebank
11 # "sentences_50agree"
12 # "sentences_66agree"
13 # "sentences_75agree"
```

```
In [ ]: 1 # Preprocess the dataset
2 def preprocess_function(examples):
3     return tokenizer(examples["sentence"],
4                       padding="longest",
5                       max_length= 45,
6                       truncation = True)
7
8 dataset = dataset.map(preprocess_function, batched=True)
9
10 # Split the dataset into training and validation sets
11 train_test_split = dataset['train'].train_test_split(test_size=0.2, seed=42)
12 train_dataset = train_test_split['train']
13 eval_dataset = train_test_split['test']
14
15
```

```
In [ ]: 1 # Training arguments
2 training_args = TrainingArguments(
3     output_dir = "./results",
4     evaluation_strategy = "epoch",
5     learning_rate = 0.0001,
6     per_device_train_batch_size = 25,
7     per_device_eval_batch_size = 25,
8     num_train_epochs = 2,
9     warmup_steps = 500,
10    save_strategy = "epoch"
11 )
12
13 # Define the trainer
14 trainer = Trainer(
15     model = model,
16     args = training_args,
17     train_dataset = train_dataset,
18     eval_dataset = eval_dataset,
19 )
20
21 # Fine-tune the model
22 trainer.train()
```

```
In [ ]: 1 # Evaluate the fine-tuned model
2 predictions = trainer.predict(eval_dataset)
3 preds = np.argmax(predictions.predictions, axis=1)
4 labels = eval_dataset["label"]
5 accuracy = accuracy_score(labels, preds)
6 report = classification_report(labels, preds)
7
8 print(f"Validation Accuracy: {accuracy:.2f}")
9 print(f"Classification Report:\n{report}")
```

```
In [ ]: 1 # Confusion matrix
2 cm = confusion_matrix(labels, preds)
3
4 # Plot the confusion matrix using seaborn heatmap
5 plt.figure(figsize=(8, 6))
6
7 sns.heatmap(cm,
8             annot=True,
9             fmt='d',
10             cmap='seismic',
11             cbar=False,
12             xticklabels=['Negative', 'Neutral', 'Positive'],
13             yticklabels=['Negative', 'Neutral', 'Positive'])
14
15 plt.title('Confusion Matrix')
16 plt.xlabel('Predicted Labels')
17 plt.ylabel('True Labels')
18
19
20 plt.show()
```



```
In [ ]: 1 # Load the test dataset
2 test_dataset = load_dataset("financial_phrasebank", "sentences_75agree")
3 test_dataset = test_dataset.map(preprocess_function, batched=True)
4
5 # Evaluate the fine-tuned model on the test set
6 test_predictions = trainer.predict(test_dataset['train'])
7 test_preds = np.argmax(test_predictions.predictions, axis=1)
8 test_labels = test_dataset['train']["label"]
9 test_accuracy = accuracy_score(test_labels, test_preds)
10 test_report = classification_report(test_labels, test_preds)
11
```

```
In [ ]: 1 print(f"Test Accuracy: {test_accuracy:.2f}")
2 print(f"Test Classification Report:\n{test_report}")
```

```
In [ ]: 1 # Confusion matrix
2 test_cm = confusion_matrix(test_labels, test_preds)
3
4 # Plot the confusion matrix using seaborn heatmap
5 plt.figure(figsize=(8, 6))
6
7 sns.heatmap(test_cm,
8             annot=True,
9             fmt='d',
10            cmap='seismic',
11            cbar=False,
12            xticklabels=['Negative', 'Neutral', 'Positive'],
13            yticklabels=['Negative', 'Neutral', 'Positive'])
14
15 plt.title('Confusion Matrix')
16 plt.xlabel('Predicted Labels')
17 plt.ylabel('True Labels')
18
19
20 plt.show()
```

```
In [ ]: 1 # Total Time Consumed
2 end_time = time.time()
3 execution_time = end_time - start_time
4 print(f"Total Execution Time: {execution_time} seconds")
```