# Predicting interest rates from Federal Reserve documents

## Model Training (Vol. 6)

FE 690: Machine Learning in Finance

Author: Theo Dimitrasopoulos

Advisor: Zachary Feinstein

# Setup

## Environment

```
# -*- coding: utf-8 -*-

# ENVIRONMENT CHECK:
import sys, os, inspect, site, pprint
# Check whether in Colab:
IN_COLAB = 'google.colab' in sys.modules
if IN_COLAB == True:
  print('YES, this is a Google Colaboratory environment.')
else:
  print('NO, this is not a Google Colaboratory environment.')
print(' ')

# Python installation files:
stdlib = os.path.dirname(inspect.getfile(os))
python_version = !python --version
print('Python Standard Library is located in:\n' + stdlib)
print(' ')
print('This environment is using {}'.format(str(python_version[0])))
print(' ')
print('Local system packages are located in:')
pprint.pprint(site.getsitepackages())
print(' ')
print('Local user packages are located in:\n' + site.getusersitepackages())

# Installed packages:
!pip list -v
!pip list --user -v
```

```
pysndfile                    1.3.8          /usr/local/lib/python3.6/dist-packages pip
PySocks                      1.7.1          /usr/local/lib/python3.6/dist-packages pip
pystan                       2.19.1.1       /usr/local/lib/python3.6/dist-packages pip
pytest                       3.6.4          /usr/local/lib/python3.6/dist-packages pip
python-apt                   1.6.5+ubuntu0.5 /usr/lib/python3/dist-packages
```

```
python-chess                  0.23.11        /usr/local/lib/python3.6/dist-packages pip
python-dateutil               2.8.1          /usr/local/lib/python3.6/dist-packages pip
python-louvain                0.15           /usr/local/lib/python3.6/dist-packages pip
python-pptx                   0.6.18         /usr/local/lib/python3.6/dist-packages pip
python-slugify                4.0.1          /usr/local/lib/python3.6/dist-packages pip
python-utils                  2.4.0          /usr/local/lib/python3.6/dist-packages pip
pytz                          2018.9         /usr/local/lib/python3.6/dist-packages pip
pyviz-comms                   2.0.1          /usr/local/lib/python3.6/dist-packages pip
PyWavelets                    1.1.1          /usr/local/lib/python3.6/dist-packages pip
PyYAML                        3.13           /usr/local/lib/python3.6/dist-packages pip
pyzmq                         20.0.0         /usr/local/lib/python3.6/dist-packages pip
qdldl                         0.1.5.post0    /usr/local/lib/python3.6/dist-packages pip
qtconsole                     5.0.1          /usr/local/lib/python3.6/dist-packages pip
QtPy                          1.9.0          /usr/local/lib/python3.6/dist-packages pip
Quandl                        3.5.3          /usr/local/lib/python3.6/dist-packages pip
regex                         2019.12.20     /usr/local/lib/python3.6/dist-packages pip
requests                      2.24.0         /usr/local/lib/python3.6/dist-packages pip
requests-oauthlib             1.3.0          /usr/local/lib/python3.6/dist-packages pip
resampy                       0.2.2          /usr/local/lib/python3.6/dist-packages pip
retrying                      1.3.3          /usr/local/lib/python3.6/dist-packages pip
rpy2                          3.2.7          /usr/local/lib/python3.6/dist-packages pip
rsa                           4.6            /usr/local/lib/python3.6/dist-packages pip
sacremoses                    0.0.43         /usr/local/lib/python3.6/dist-packages pip
scikit-image                  0.16.2         /usr/local/lib/python3.6/dist-packages pip
scikit-learn                  0.22.2.post1   /usr/local/lib/python3.6/dist-packages pip
scikit-plot                   0.3.7          /usr/local/lib/python3.6/dist-packages pip
scipy                         1.4.1          /usr/local/lib/python3.6/dist-packages pip
screen-resolution-extra       0.0.0          /usr/lib/python3/dist-packages
scs                           2.1.2          /usr/local/lib/python3.6/dist-packages pip
seaborn                       0.11.0         /usr/local/lib/python3.6/dist-packages pip
Send2Trash                    1.5.0          /usr/local/lib/python3.6/dist-packages pip
sentencepiece                 0.1.91         /usr/local/lib/python3.6/dist-packages pip
setuptools                    51.3.3         /usr/local/lib/python3.6/dist-packages pip
setuptools-git                1.2            /usr/local/lib/python3.6/dist-packages pip
Shapely                       1.7.1          /usr/local/lib/python3.6/dist-packages pip
simplegeneric                 0.8.1          /usr/local/lib/python3.6/dist-packages pip
six                           1.12.0         /usr/local/lib/python3.6/dist-packages pip
sklearn                       0.0            /usr/local/lib/python3.6/dist-packages pip
sklearn-pandas                1.8.0          /usr/local/lib/python3.6/dist-packages pip
smart-open                    4.1.0          /usr/local/lib/python3.6/dist-packages pip
snowballstemmer               2.0.0          /usr/local/lib/python3.6/dist-packages pip
sortedcontainers              2.3.0          /usr/local/lib/python3.6/dist-packages pip
soupsieve                     2.1            /usr/local/lib/python3.6/dist-packages pip
spacy                         2.2.4          /usr/local/lib/python3.6/dist-packages pip
SpeechRecognition             3.8.1          /usr/local/lib/python3.6/dist-packages pip
Sphinx                        1.8.5          /usr/local/lib/python3.6/dist-packages pip
sphinxcontrib-serializinghtml 1.1.4          /usr/local/lib/python3.6/dist-packages pip
sphinxcontrib-websupport      1.2.4          /usr/local/lib/python3.6/dist-packages pip
SQLAlchemy                    1.3.22         /usr/local/lib/python3.6/dist-packages pip
sqlparse                      0.4.1          /usr/local/lib/python3.6/dist-packages pip
srsly                         1.0.5          /usr/local/lib/python3.6/dist-packages pip
statsmodels                   0.10.2         /usr/local/lib/python3.6/dist-packages pip
sympy                         1.1.1          /usr/local/lib/python3.6/dist-packages pip
tables                        3.4.4          /usr/local/lib/python3.6/dist-packages pip
tabulate                      0.8.7          /usr/local/lib/python3.6/dist-packages pip
```

## Mount Google Drive

```
# Mount Google Drive:
```

```
# Mount Google Drive:
if IN_COLAB:
  from google.colab import drive
  drive.mount('/content/drive', force_remount=True)
```

```
    Mounted at /content/drive
```

## System Environment Variables

```
if IN_COLAB:
  employment_data_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/MarketData/Employment/'
  cpi_data_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/MarketData/CPI/'
  fed_rates_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/MarketData/FEDRates/'
  fx_rates_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/MarketData/FXRates/'
  gdp_data_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/MarketData/GDP/'
  ism_data_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/MarketData/ISM/'
  sales_data_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/MarketData/Sales/'
  treasury_data_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/MarketData/Treasury/'
  fomc_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/FOMC/'
  preprocessed_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/preprocessed/'
  train_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/train_data/'
  output_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/result/'
  keyword_lm_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/LoughranMcDonald/'
  glove_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/GloVe/'
  model_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/models/'
  graph_dir = '/content/drive/My Drive/Colab Notebooks/proj2/src/data/graphs/'
else:
  employment_data_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/MarketData/Employment/'
  cpi_data_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/MarketData/CPI/'
  fed_rates_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/MarketData/FEDRates/'
  fx_rates_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/MarketData/FXRates/'
  gdp_data_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/MarketData/GDP/'
  ism_data_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/MarketData/ISM/'
  sales_data_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/MarketData/Sales/'
  treasury_data_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/MarketData/Treasury/'
  fomc_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/FOMC/'
  preprocessed_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/preprocessed/'
  train_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/train_data/'
  output_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/result/'
  keyword_lm_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/LoughranMcDonald/'
  glove_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/GloVe/'
  model_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/models/'
  graph_dir = 'C:/Users/theon/GDrive/Colab Notebooks/proj2/src/data/graphs/'
```

## Packages

### Uninstall/Install Packages:

```
#if IN_COLAB:
#  # Uninstall existing versions:
#  !pip uninstall bs4 -y
#  !pip uninstall textract -y
#  !pip uninstall numpy -y
#  !pip uninstall pandas -y
#  !pip uninstall requests -y
#  !pip uninstall tqdm -y
#  !pip uninstall nltk -y
#  !pip uninstall quandl -y
#  !pip uninstall scikit-plot -y
#  !pip uninstall seaborn -y
#  !pip uninstall sklearn -y
#  !pip uninstall torch -y
#  !pip uninstall transformers -y
#  !pip uninstall wordcloud -y
#  !pip uninstall xgboost -y
#
#  # Install packages:
#  !pip install bs4==0.0.1
#  !pip install textract==1.6.3
#  !pip install numpy==1.19.4
#  !pip install pandas==1.1.4
#  !pip install requests==2.24.0
#  !pip install tqdm==4.51.0
#  !pip install nltk==3.5
#  !pip install quandl==3.5.3
#  !pip install scikit-plot==0.3.7
#  !pip install seaborn==0.11.0
#  !pip install sklearn==0.0
#  !pip install torch==1.7.1+cu101 torchvision==0.8.2+cu101 -f https://download.pytorch.org/whl/torch_stable.html
#  !pip install transformers==3.5.0
#  !pip install wordcloud==1.8.0
#  !pip install xgboost==1.2.1
#  os.kill(os.getpid(), 9)
```

## Inspect Packages

```
!pip list -v
!pip list --user -v
```

```
    tblib                       1.7.0          /usr/local/lib/python3.6/dist-packages pip
    tensorboard                 2.4.0          /usr/local/lib/python3.6/dist-packages pip
    tensorboard-plugin-wit      1.7.0          /usr/local/lib/python3.6/dist-packages pip
    tensorboardcolab            0.0.22         /usr/local/lib/python3.6/dist-packages pip
    tensorflow                  2.4.0          /usr/local/lib/python3.6/dist-packages pip
    tensorflow-addons           0.8.3          /usr/local/lib/python3.6/dist-packages pip
    tensorflow-datasets         4.0.1          /usr/local/lib/python3.6/dist-packages pip
    tensorflow-estimator        2.4.0          /usr/local/lib/python3.6/dist-packages pip
```

```
tensorflow-gcs-config         2.4.0          /usr/local/lib/python3.6/dist-packages pip
tensorflow-hub                0.11.0         /usr/local/lib/python3.6/dist-packages pip
tensorflow-metadata           0.26.0         /usr/local/lib/python3.6/dist-packages pip
tensorflow-privacy            0.2.2          /usr/local/lib/python3.6/dist-packages pip
tensorflow-probability        0.12.1         /usr/local/lib/python3.6/dist-packages pip
termcolor                     1.1.0          /usr/local/lib/python3.6/dist-packages pip
terminado                     0.9.2          /usr/local/lib/python3.6/dist-packages pip
testpath                      0.4.4          /usr/local/lib/python3.6/dist-packages pip
text-unidecode                1.3            /usr/local/lib/python3.6/dist-packages pip
textblob                      0.15.3         /usr/local/lib/python3.6/dist-packages pip
textgenrnn                    1.4.1          /usr/local/lib/python3.6/dist-packages pip
textract                      1.6.3          /usr/local/lib/python3.6/dist-packages pip
Theano                        1.0.5          /usr/local/lib/python3.6/dist-packages pip
thinc                         7.4.0          /usr/local/lib/python3.6/dist-packages pip
tifffile                      2020.9.3       /usr/local/lib/python3.6/dist-packages pip
tokenizers                    0.9.3          /usr/local/lib/python3.6/dist-packages pip
toml                          0.10.2         /usr/local/lib/python3.6/dist-packages pip
toolz                         0.11.1         /usr/local/lib/python3.6/dist-packages pip
torch                         1.7.1+cu101    /usr/local/lib/python3.6/dist-packages pip
torchsummary                  1.5.1          /usr/local/lib/python3.6/dist-packages pip
torchtext                     0.3.1          /usr/local/lib/python3.6/dist-packages pip
torchvision                   0.8.2+cu101    /usr/local/lib/python3.6/dist-packages pip
tornado                       5.1.1          /usr/local/lib/python3.6/dist-packages pip
tqdm                          4.51.0         /usr/local/lib/python3.6/dist-packages pip
traitlets                     4.3.3          /usr/local/lib/python3.6/dist-packages pip
transformers                  3.5.0          /usr/local/lib/python3.6/dist-packages pip
tweepy                        3.6.0          /usr/local/lib/python3.6/dist-packages pip
typeguard                     2.7.1          /usr/local/lib/python3.6/dist-packages pip
typing-extensions             3.7.4.3        /usr/local/lib/python3.6/dist-packages pip
tzlocal                       1.5.1          /usr/local/lib/python3.6/dist-packages pip
umap-learn                    0.4.6          /usr/local/lib/python3.6/dist-packages pip
uritemplate                   3.0.1          /usr/local/lib/python3.6/dist-packages pip
urllib3                       1.24.3         /usr/local/lib/python3.6/dist-packages pip
vega-datasets                 0.9.0          /usr/local/lib/python3.6/dist-packages pip
wasabi                        0.8.0          /usr/local/lib/python3.6/dist-packages pip
wcwidth                       0.2.5          /usr/local/lib/python3.6/dist-packages pip
webencodings                  0.5.1          /usr/local/lib/python3.6/dist-packages pip
Werkzeug                      1.0.1          /usr/local/lib/python3.6/dist-packages pip
wheel                         0.36.2         /usr/local/lib/python3.6/dist-packages pip
widgetsnbextension            3.5.1          /usr/local/lib/python3.6/dist-packages pip
wordcloud                     1.8.0          /usr/local/lib/python3.6/dist-packages pip
wrapt                         1.12.1         /usr/local/lib/python3.6/dist-packages pip
xarray                        0.15.1         /usr/local/lib/python3.6/dist-packages pip
xgboost                       1.2.1          /usr/local/lib/python3.6/dist-packages pip
xkit                          0.0.0          /usr/lib/python3/dist-packages
xlrd                          1.2.0          /usr/local/lib/python3.6/dist-packages pip
XlsxWriter                    1.3.7          /usr/local/lib/python3.6/dist-packages pip
xlwt                          1.3.0          /usr/local/lib/python3.6/dist-packages pip
yellowbrick                   0.9.1          /usr/local/lib/python3.6/dist-packages pip
zict                          2.0.0          /usr/local/lib/python3.6/dist-packages pip
zipp                          3.4.0          /usr/local/lib/python3.6/dist-packages pip
```

## Import Packages:

```
# Python libraries
import pprint
import datetime as dt
```

```
import re
import pickle
from tqdm.notebook import tqdm
import time
import logging
import random
from collections import defaultdict, Counter
import xgboost as xgb

# Data Science modules
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
plt.style.use('ggplot')

# Import Scikit-learn models
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.metrics import accuracy_score, f1_score, plot_confusion_matrix
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier, ExtraTreesClassifier, VotingClassifier
from sklearn.linear_model import LogisticRegression, Perceptron, SGDClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn import model_selection
from sklearn.model_selection import GridSearchCV, cross_val_score, cross_validate, StratifiedKFold, learning_curve, RandomizedSearchCV
import scikitplot as skplt

# Import nltk modules and download dataset
import nltk
from nltk.corpus import stopwords
from nltk.util import ngrams
from nltk.tokenize import word_tokenize, sent_tokenize

# Import Pytorch modules
import torch
from torch import nn, optim
import torch.nn.functional as F
from torch.utils.data import (DataLoader, RandomSampler, SequentialSampler, TensorDataset)
from torch.autograd import Variable
from torch.optim import Adam, AdamW
```

## Settings

```
# General:
import warnings
warnings.filterwarnings('ignore')
```

```
%matplotlib inline
get_ipython().run_line_magic('matplotlib', 'inline')



# Finalize nltk setup:
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')

stop = set(stopwords.words('english'))

# Test pprint
pprint.pprint(sys.path)
```

```
['',
 '/env/python',
 '/usr/lib/python36.zip',
 '/usr/lib/python3.6',
 '/usr/lib/python3.6/lib-dynload',
 '/usr/local/lib/python3.6/dist-packages',
 '/usr/lib/python3/dist-packages',
 '/usr/local/lib/python3.6/dist-packages/IPython/extensions',
 '/root/.ipython']
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

```
## Use TPU
#if IN_COLAB:
#  assert os.environ['COLAB_TPU_ADDR'], 'Select TPU: Runtime > Change runtime type > Hardware accelerator'
#  VERSION = "20200220"
#  !curl https://raw.githubusercontent.com/pytorch/xla/master/contrib/scripts/env-setup.py -o pytorch-xla-env-setup.py
#  !python pytorch-xla-env-setup.py --version $VERSION


## Use GPU Runtime:
if IN_COLAB:
  if torch.cuda.is_available():
    torch.cuda.get_device_name(0)
    gpu_info = !nvidia-smi
    gpu_info = '\n'.join(gpu_info)
    print(gpu_info)
  else:
    print('Select the Runtime > "Change runtime type" menu to enable a GPU accelerator, and then re-execute this cell.')
    os.kill(os.getpid(), 9)
```

```
Mon Jan 25 13:39:37 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 418.67       CUDA Version: 10.1     |
```

```
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla V100-SXM2...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   33C    P0    25W / 300W |     10MiB / 16130MiB |      0%      Default |
|                               |                      |                 ERR! |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

```
# Set logger
logger = logging.getLogger('mylogger')
logger.setLevel(logging.DEBUG)
timestamp = time.strftime("%Y.%m.%d_%H.%M.%S", time.localtime())
fh = logging.FileHandler('log_model.txt')
fh.setLevel(logging.DEBUG)
ch = logging.StreamHandler()
ch.setLevel(logging.DEBUG)
formatter = logging.Formatter('[%(asctime)s][%(levelname)s] ## %(message)s')
fh.setFormatter(formatter)
ch.setFormatter(formatter)
logger.addHandler(fh)
logger.addHandler(ch)


# Set Random Seed
random.seed(42)
np.random.seed(42)
torch.manual_seed(42)
torch.cuda.manual_seed(42)
rand_seed = 42


# Set Seaborn Style
sns.set(style='white', context='notebook', palette='deep')
```

# Load preprocessed data

```
# Load previously processed non-text data
# Load data
file = open(train_dir + 'nontext_train_small.pickle', 'rb')
train_df = pickle.load(file)
file.close()
#train_df = pd.read_csv(train_dir + 'nontext_train_small.csv')
print(train_df.shape)
```

```
print(train_df.shape)
train_df
```

(398, 10)

| date | target | prev_decision | GDP_diff_prev | PMI_value | Employ_diff_prev | Rsales_diff_year | Unemp_diff_prev | Inertia_diff | Hsales_diff_year | Balanced_diff |
|---|---|---|---|---|---|---|---|---|---|---|
| **1982-10-05** | -1 | 0 | 0.456197 | 38.8 | -169.0 | 1.807631 | -0.166667 | -0.018226 | -15.485275 | 0.003723 |
| **1982-11-16** | -1 | -1 | -0.382295 | 39.4 | -228.0 | 1.807631 | -0.200000 | -0.018226 | -9.537496 | 0.003723 |
| **1982-12-21** | 0 | -1 | -0.382295 | 39.2 | -198.5 | 1.807631 | -0.333333 | -0.018226 | -3.116275 | 0.003723 |
| **1983-01-14** | 0 | 0 | -0.382295 | 42.8 | -68.0 | 1.807631 | -0.233333 | -0.018226 | -0.774432 | 0.003723 |
| **1983-01-21** | 0 | 0 | -0.382295 | 42.8 | -68.0 | 1.807631 | -0.233333 | -0.043785 | -0.774432 | 0.003723 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **2020-03-15** | -1 | -1 | 0.527469 | 50.1 | 232.5 | 2.217385 | 0.000000 | -0.058085 | 13.910886 | 0.004279 |
| **2020-03-19** | 0 | -1 | 0.527469 | 50.1 | 232.5 | 2.217385 | 0.000000 | -0.057139 | 13.910886 | 0.001426 |
| **2020-03-23** | 0 | 0 | 0.527469 | 50.1 | 232.5 | 2.217385 | 0.000000 | -0.057139 | 13.910886 | 0.001426 |
| **2020-03-31** | 0 | 0 | 0.527469 | 50.1 | 232.5 | 2.217385 | 0.000000 | -0.114279 | 13.910886 | 0.006092 |
| **2020-04-29** | 0 | 0 | 0.527469 | 49.1 | -561.0 | -2.491979 | -0.300000 | -0.431520 | 12.468252 | 0.040295 |

398 rows × 10 columns

```
# List of Non-text columns
nontext_columns = train_df.drop(columns=['target']).columns.tolist()
nontext_columns
```

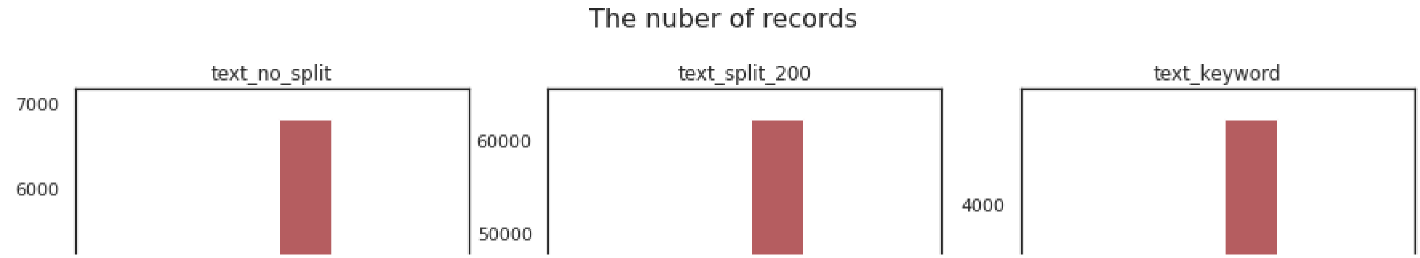
```
['prev_decision',
 'GDP_diff_prev',
 'PMI_value',
 'Employ_diff_prev',
 'Rsales_diff_year',
 'Unemp_diff_prev',
 'Inertia_diff',
 'Hsales_diff_year',
 'Balanced_diff']
```

```
# Load text data
file = open(preprocessed_dir + 'text_no_split.pickle', 'rb') # Original text
```

```
file = open(preprocessed_dir + 'text_no_split.pickle', 'rb') # Original text
text_no_split = pickle.load(file)
file.close()
#text_no_split = pd.read_csv(preprocessed_dir + 'text_no_split.csv')
file = open(preprocessed_dir + 'text_split_200.pickle', 'rb') # Split at 200 words
text_split_200 = pickle.load(file)
file.close()
#text_split_200 = pd.read_csv(preprocessed_dir + 'text_split_200.csv')
file = open(preprocessed_dir + 'text_keyword.pickle', 'rb') # Paragraphs filtered for those having keywords
text_keyword = pickle.load(file)
file.close()
#text_keyword = pd.read_csv(preprocessed_dir + 'text_keyword.csv')
```

## Check statistics of texts

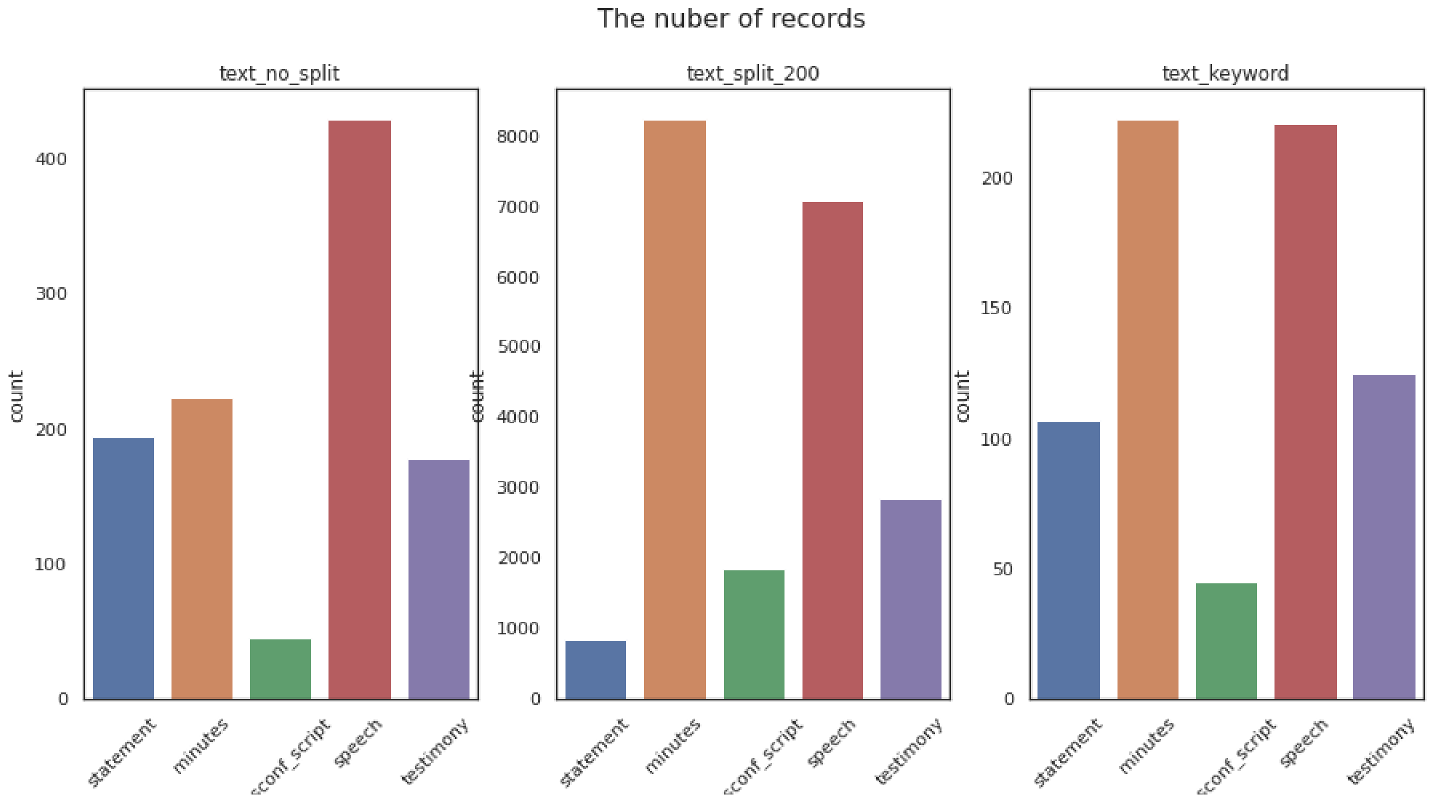
```
# Check the number of records per document type
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15,7))
sns.countplot(x='type', data=text_no_split, ax=ax1)
ax1.set_title('text_no_split')
ax1.tick_params('x', labelrotation=45)
sns.countplot(x='type', data=text_split_200, ax=ax2)
ax2.set_title('text_split_200')
ax2.tick_params('x', labelrotation=45)
sns.countplot(x='type', data=text_keyword, ax=ax3)
ax3.set_title('text_keyword')
ax3.tick_params('x', labelrotation=45)
fig.suptitle("The nuber of records", fontsize=16)
plt.show()
plt.savefig(graph_dir + 'num_rec_per_type_text_data' + '.png')#bbox_inches='tight')
```



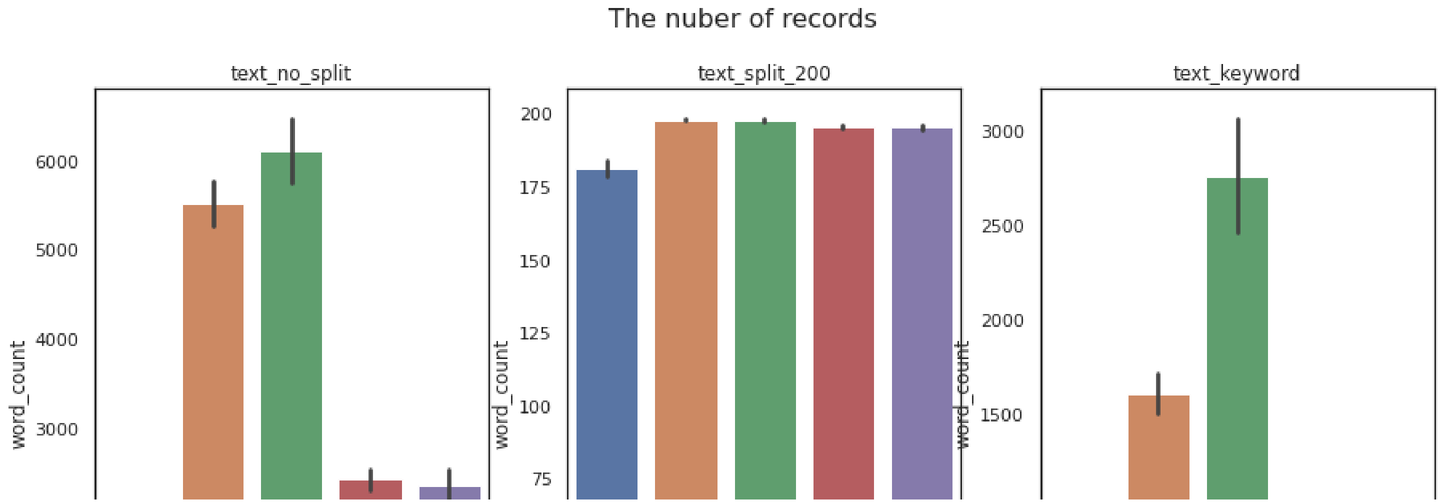

```
text_no_split.loc[text_no_split['type'] == 'meeting_script'].head()
```

| | type | date | title | speaker | word_count | decision | rate | next_meeting | next_decision | next_rate | text |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **463** | meeting_script | 1982-10-05 | FOMC Meeting Transcript | CHAIRMA | 1801 | -1 | 9.5 | 1982-11-16 | -1 | 9.0 | VOLCKER.while but it may be that itsurprised m... |
| **464** | meeting_script | 1982-10-05 | FOMC Meeting Transcript | CHAIRMAN VOLCKER | 8439 | -1 | 9.5 | 1982-11-16 | -1 | 9.0 | Without objection, it is approved. As forthe M... |
| **465** | meeting_script | 1982-10-05 | FOMC Meeting Transcript | MR. AXILROD | 567 | -1 | 9.5 | 1982-11-16 | -1 | 9.0 | Well, the one that is reserveable certainlywou... |
| **466** | meeting_script | 1982-10-05 | FOMC Meeting Transcript | MR. BALLES | 141 | -1 | 9.5 | 1982-11-16 | -1 | 9.0 | Well, coming at this confidence factor from al... |
| **467** | meeting_script | 1982-10-05 | FOMC Meeting Transcript | MR. BLACK | 415 | -1 | 9.5 | 1982-11-16 | -1 | 9.0 | Mr. Chairman, Larry rescued us from the strait... |

```
# Drop meeting script data
text_no_split = text_no_split.loc[text_no_split['type'] != 'meeting_script']
text_split_200 = text_split_200.loc[text_split_200['type'] != 'meeting_script']
text_keyword = text_keyword.loc[text_keyword['type'] != 'meeting_script']

# Check the number of records per document type
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15,7))
sns.countplot(x='type', data=text_no_split, ax=ax1)
ax1.set_title('text_no_split')
ax1.tick_params('x', labelrotation=45)
sns.countplot(x='type', data=text_split_200, ax=ax2)
ax2.set_title('text_split_200')
ax2.tick_params('x', labelrotation=45)
sns.countplot(x='type', data=text_keyword, ax=ax3)
ax3.set_title('text_keyword')
ax3.tick_params('x', labelrotation=45)
fig.suptitle("The nuber of records", fontsize=16)
plt.show()
```




```
# Check the number of words per document type
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15,7))
sns.barplot(data=text_no_split, x='type', y='word_count', ax=ax1)
ax1.set_title('text_no_split')
ax1.tick_params('x', labelrotation=45)
sns.barplot(x='type', y='word_count', data=text_split_200, ax=ax2)
ax2.set_title('text_split_200')
ax2.tick_params('x', labelrotation=45)
sns.barplot(x='type', y='word_count', data=text_keyword, ax=ax3)
ax3.set_title('text_keyword')
ax3.tick_params('x', labelrotation=45)
fig.suptitle("The nuber of records", fontsize=16)
plt.show()
```



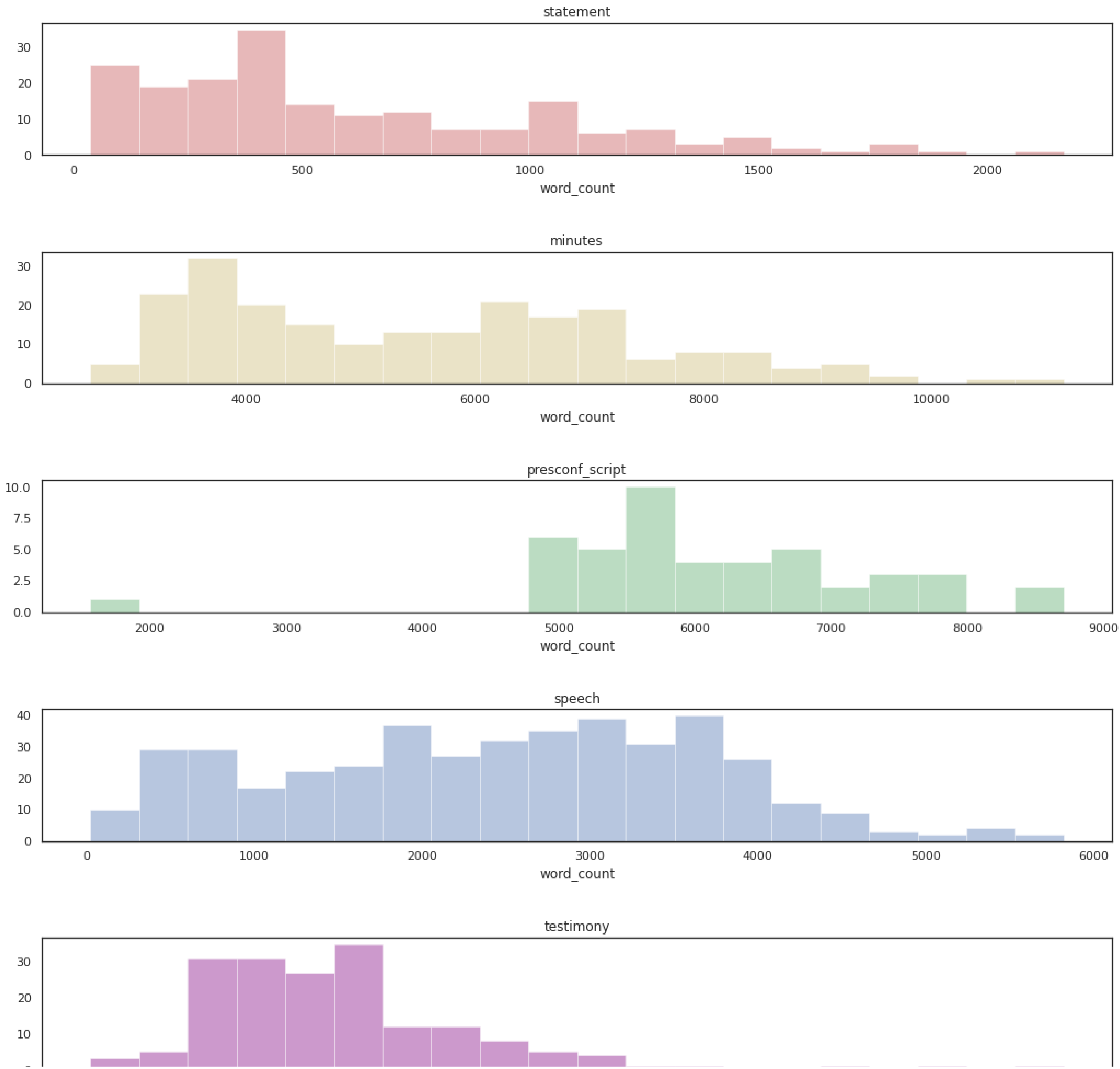

## Select text dataframe

```
# Select one from the above different pre-processed data
text_df = text_no_split
text_df.reset_index(drop=True, inplace=True)
print(text_df.shape)
text_df
```

(1071, 11)

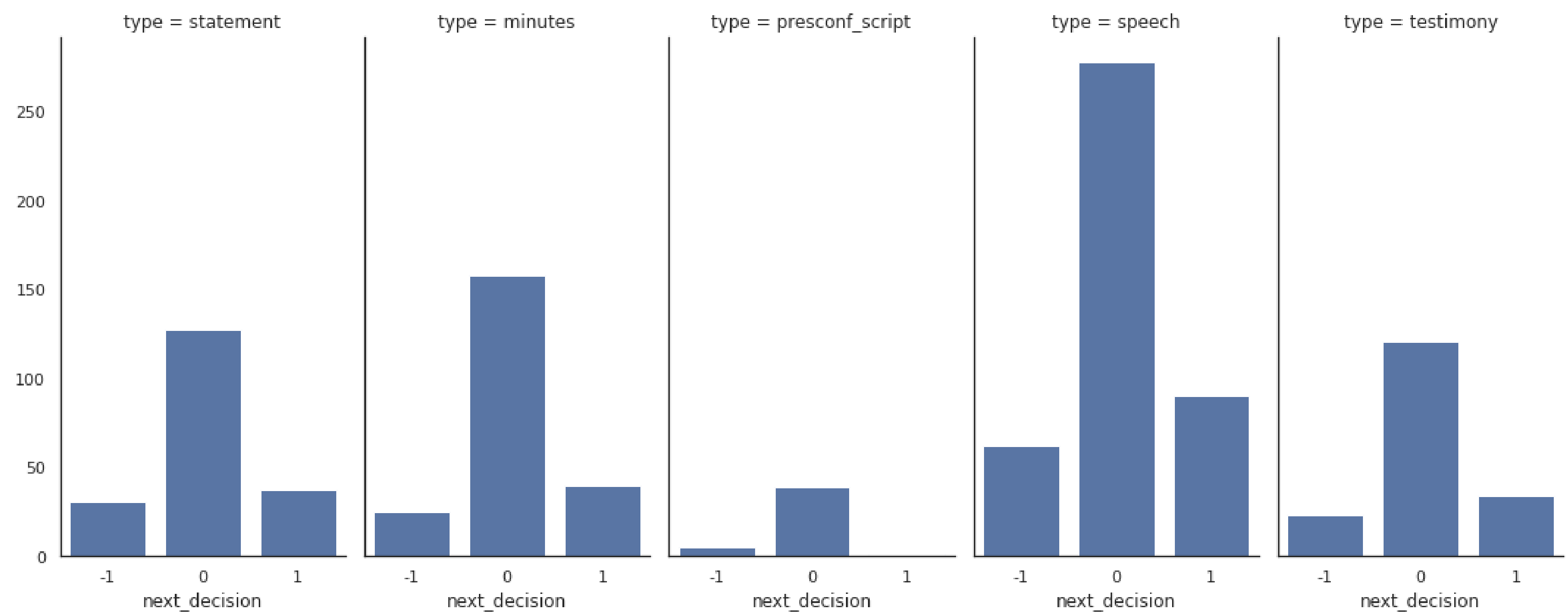
| | type | date | title | speaker | word_count | decision | rate | next_meeting | next_decision | next_rate | text |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | statement | 1994-02-04 | FOMC Statement | Alan Greenspan | 99 | 1 | 3.25 | 1994-02-28 | 0 | 3.25 | Chairman Alan Greenspan announced today that t... |
| **1** | statement | 1994-03-22 | FOMC Statement | Alan Greenspan | 40 | 1 | 3.5 | 1994-04-18 | 1 | 3.75 | Chairman Alan Greenspan announced today that t... |
| **2** | statement | 1994-04-18 | FOMC Statement | Alan Greenspan | 37 | 1 | 3.75 | 1994-05-17 | 1 | 4.25 | Chairman Alan Greenspan announced today that t... |
| **3** | statement | 1994-05-17 | FOMC Statement | Alan Greenspan | 57 | 1 | 4.25 | 1994-07-06 | 0 | 4.25 | In taking the discount action, the Board appro... |
| **4** | statement | 1994-08-16 | FOMC Statement | Alan Greenspan | 51 | 1 | 4.75 | 1994-09-27 | 0 | 4.75 | In taking the discount rate action, the Board ... |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **1066** | testimony | 2020-05-19 | Coronavirus and CARES Act | Jerome Powell | 1802 | <NA> | None | 2020-06-10 | 0 | 0.00 | I would like to begin by acknowledging the tra... |
| **1067** | testimony | 2020-06-16 | Semiannual Monetary Policy Report to the Congress | Jerome Powell | 1433 | <NA> | None | 2020-07-29 | 0 | 0.00 | Our country continues to face a difficult and ... |

```
# Check distribution

fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(5, 1, figsize=(15,15))
doc_type = 'statement'
sns.distplot(text_df.loc[text_df['type'] == doc_type]['word_count'], bins=20, ax=ax1, kde=False, color='r')
ax1.set_title(doc_type)
doc_type = 'minutes'
sns.distplot(text_df.loc[text_df['type'] == doc_type]['word_count'], bins=20, ax=ax2, kde=False, color='y')
ax2.set_title(doc_type)
doc_type = 'presconf_script'
sns.distplot(text_df.loc[text_df['type'] == doc_type]['word_count'], bins=20, ax=ax3, kde=False, color='g')
ax3.set_title(doc_type)
doc_type = 'speech'
sns.distplot(text_df.loc[text_df['type'] == doc_type]['word_count'], bins=20, ax=ax4, kde=False, color='b')
ax4.set_title(doc_type)
doc_type = 'testimony'
sns.distplot(text_df.loc[text_df['type'] == doc_type]['word_count'], bins=20, ax=ax5, kde=False, color='purple')
ax5.set_title(doc_type)

fig.tight_layout(pad=3.0)
plt.show()
plt.savefig(graph_dir + 'word_count_distribution_per_doc_type_text_df' + '.png')#bbox_inches='tight')
```




```
# Check balance of Rate Decision
g = sns.FacetGrid(text_df, col='type', height 6, aspect 0.5)
```

```
g = sns.FacetGrid(text_df, col='type', height=6, aspect=0.5)
g.map(sns.countplot, 'next_decision')
plt.show()
```




The label is highly biased to 0(Hold). Need to consider how to mitigate the biased data.

## Merge text_df with train_df

```
from collections import defaultdict

doc_types = text_df['type'].unique()

merged_dict = defaultdict(list)

for i, row in train_df.iterrows():
    text_rows = text_df.loc[text_df['next_meeting'] == i]
    merged_text_all = ""
    for doc_type in doc_types:
        merged_text = ""
        for text in text_rows.loc[text_rows['type'] == doc_type]['text']:
            merged_text += " " + text
        merged_dict[doc_type].append(merged_text)
        merged_text_all += merged_text
    merged_dict['text'].append(merged_text_all)
```

```
for key in merged_dict.keys():
    train_df[key] = merged_dict[key]

train_df
```

| | target | prev_decision | GDP_diff_prev | PMI_value | Employ_diff_prev | Rsales_diff_year | Unemp_diff_prev | Inertia_diff | Hsales_diff_year | Balanced_diff | state |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **date** | | | | | | | | | | | |
| **1982-10-05** | -1 | 0 | 0.456197 | 38.8 | -169.0 | 1.807631 | -0.166667 | -0.018226 | -15.485275 | 0.003723 | |
| **1982-11-16** | -1 | -1 | -0.382295 | 39.4 | -228.0 | 1.807631 | -0.200000 | -0.018226 | -9.537496 | 0.003723 | |
| **1982-12-21** | 0 | -1 | -0.382295 | 39.2 | -198.5 | 1.807631 | -0.333333 | -0.018226 | -3.116275 | 0.003723 | |
| **1983-01-14** | 0 | 0 | -0.382295 | 42.8 | -68.0 | 1.807631 | -0.233333 | -0.018226 | -0.774432 | 0.003723 | |
| **1983-01-21** | 0 | 0 | -0.382295 | 42.8 | -68.0 | 1.807631 | -0.233333 | -0.043785 | -0.774432 | 0.003723 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **2020-03-15** | -1 | -1 | 0.527469 | 50.1 | 232.5 | 2.217385 | 0.000000 | -0.058085 | 13.910886 | 0.004279 | fundame of the ecor remai |
| **2020-03-19** | 0 | -1 | 0.527469 | 50.1 | 232.5 | 2.217385 | 0.000000 | -0.057139 | 13.910886 | 0.001426 | corona outbreal ha commu |
| **2020-03-23** | 0 | 0 | 0.527469 | 50.1 | 232.5 | 2.217385 | 0.000000 | -0.057139 | 13.910886 | 0.001426 | |
| **2020-03-31** | 0 | 0 | 0.527469 | 50.1 | 232.5 | 2.217385 | 0.000000 | -0.114279 | 13.910886 | 0.006092 | The Fe Open M Committ takir |
| **2020-04-29** | 0 | 0 | 0.527469 | 49.1 | -561.0 | -2.491979 | -0.300000 | -0.431520 | 12.468252 | 0.040295 | The Fe Reser Tue annou t |

398 rows × 16 columns

```
# Check if most of docs are merged

count_text, count_train = 0, 0
```

```
count_text, count_train = 0, 0

for doc_type in doc_types:
    count = 0
    for text in text_df.loc[text_df['type']==doc_type]['text']:
        count += len(text.split())
    print("{} words in original text for {}".format(count, doc_type))
    count_text += count

    count = 0
    for text in train_df[doc_type]:
        count += len(text.split())
    print("{} words in merged text for {}".format(count, doc_type))
    count_train += count

print("Total: {} words in original text".format(count_text))
print("Total: {} words in merged text".format(count_train))
print("Total: {} words in text column of merged text".format(train_df['text'].apply(lambda x: len(x.split())).sum()))
```

```
120036 words in original text for statement
117456 words in merged text for statement
1227702 words in original text for minutes
1180210 words in merged text for minutes
260491 words in original text for presconf_script
219491 words in merged text for presconf_script
1044550 words in original text for speech
1037401 words in merged text for speech
421275 words in original text for testimony
410549 words in merged text for testimony
Total: 3074054 words in original text
Total: 2965107 words in merged text
Total: 2965107 words in text column of merged text
```

```
print("Before dropping: ", train_df.shape)
train_df = train_df.loc[train_df['text'] != ""]
print("After dropping: ", train_df.shape)
train_df
```

```
Before dropping:  (398, 16)
After dropping:  (237, 16)
```

| date | target | prev_decision | GDP_diff_prev | PMI_value | Employ_diff_prev | Rsales_diff_year | Unemp_diff_prev | Inertia_diff | Hsales_diff_year | Balanced_diff | state |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **1993-02-18** | 0 | 0 | 1.043165 | 55.8 | 261.0 | 1.807631 | 0.000000 | -0.015902 | 14.901418 | 0.035879 | |
| **1993-05-18** | 0 | 0 | 0.167400 | 50.2 | 126.0 | 3.092456 | 0.066667 | -0.000720 | 13.455236 | 0.111134 | |
| **1993-07-07** | 0 | 0 | 0.167400 | 49.6 | 226.5 | 4.263357 | 0.000000 | 0.050013 | 13.446869 | -0.016140 | |
| **1993-08-17** | 0 | 0 | 0.582420 | 50.2 | 243.5 | 4.611673 | 0.066667 | 0.001967 | 11.927296 | 0.028625 | |
| **1993-09-21** | 0 | 0 | 0.582420 | 50.7 | 228.5 | 4.894733 | 0.100000 | -0.006682 | 10.302509 | -0.010715 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **2020-03-03** | -1 | 0 | 0.527469 | 50.9 | 199.0 | 2.786082 | 0.000000 | 0.007269 | 13.625558 | -0.026466 | Inform rece sinc Fe Oper |
| **2020-03-15** | -1 | -1 | 0.527469 | 50.1 | 232.5 | 2.217385 | 0.000000 | -0.058085 | 13.910886 | 0.004279 | fundame of the ecor remai |
| **2020-** | 0 | -1 | 0.527469 | 50.1 | 232.5 | 2.217385 | 0.000000 | -0.057139 | 13.910886 | 0.001426 | corona outbreal |

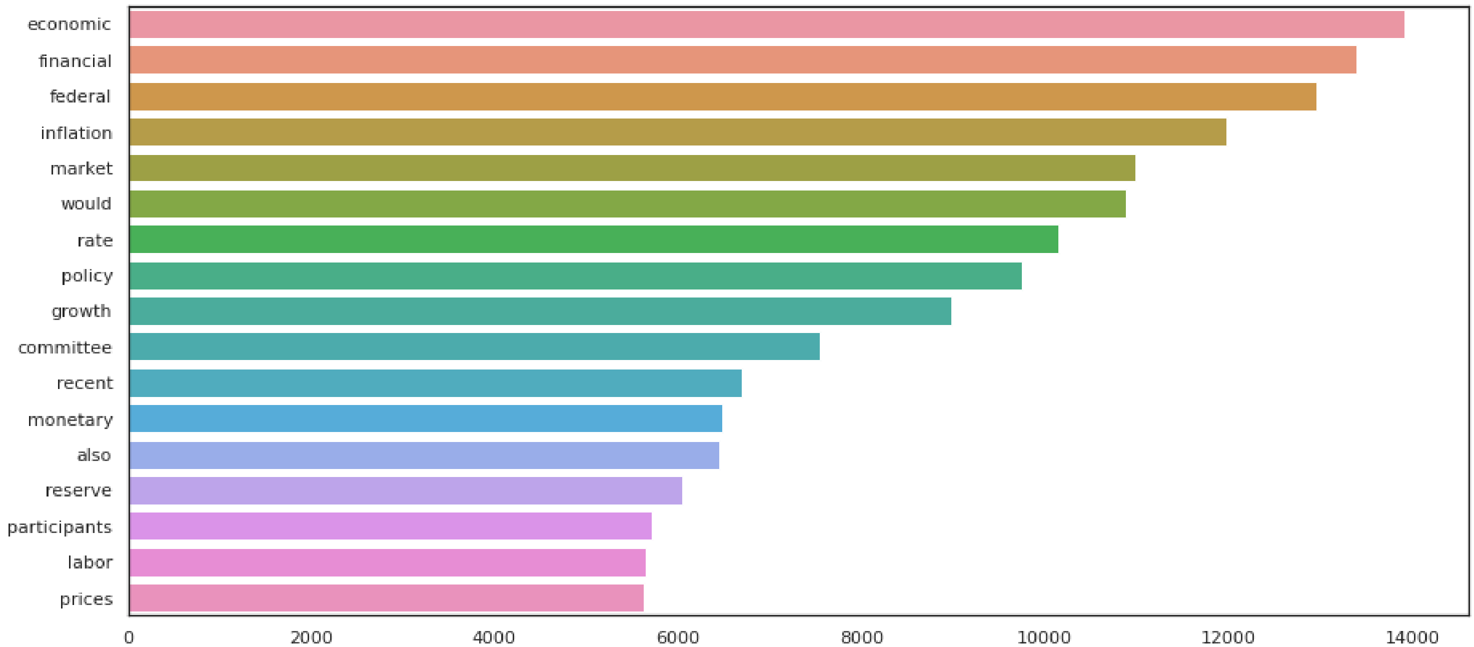
## Explore the text

```
# Corpus
def create_corpus(df):
    corpus = []

    for x in df['text'].str.split():
        for i in x:
            corpus.append(i.lower())
    return corpus
```

```
# Returns Top X frequent stop words
def get_frequent_stop_words(corpus, top_n=10):
    dic = defaultdict(int)
    for word in corpus:
        if word in stop:
            dic[word] += 1

    top = sorted(dic.items(), key=lambda x: x[1], reverse=True)[:top_n]

    return zip(*top)

# Returns Top X frequent non stop words
def get_frequent_nonstop_words(corpus, top_n=10):
    dic = defaultdict(int)
    for word in corpus:
        if word not in stop:
            dic[word] += 1

    top = sorted(dic.items(), key=lambda x: x[1], reverse=True)[:top_n]

    return zip(*top)


corpus = create_corpus(text_df)


x, y = get_frequent_stop_words(corpus)


print(x)
print(y)

     ('the', 'of', 'in', 'to', 'and', 'a', 'that', 'for', 'on', 'as')
     (210128, 114683, 95769, 91901, 89630, 47939, 45998, 32379, 23220, 22360)


x, y = get_frequent_nonstop_words(corpus)


print(x)
print(y)

     ('economic', 'financial', 'federal', 'inflation', 'market', 'would', 'rate', 'policy', 'growth', 'committee')
     (13910, 13389, 12950, 11980, 10979, 10868, 10139, 9747, 8972, 7544)


# Check most frequent words which are not in stopwords
counter = Counter(corpus)
most = counter.most_common()[:60]
x, y = [], []
for word, count in most:
    if word not in stop:
        x.append(word)
        y.append(count)
```

```
plt.figure(figsize=(15,7))
sns.barplot(x=y, y=x)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f80a1b0bef0>
```




```
# Generate Word Cloud image
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

# Create stopword list:
stopwords = set(STOPWORDS)
stopwords.update(["federal", "federal reserve", "financial", "committee", "market", "would", "also"])

text = " ".join(corpus)

# Generate a word cloud image
wordcloud = WordCloud(stopwords=stopwords, max_font_size=50, max_words=100, background_color="white").generate(text)
plt.figure(figsize=(15,7))
# Display the generated image:
# the matplotlib way:
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
plt.savefig(graph_dir + 'word_cloud_image_corpus_text_df' + '.png')#bbox_inches='tight')
# Generate a word cloud image
wordcloud = WordCloud(stopwords=stopwords, background_color="white").generate(text)
```


## Add sentiment

*Use Loughran and McDonald Sentiment Word Lists ([https://sraf.nd.edu/textual-analysis/resources/](https://sraf.nd.edu/textual-analysis/resources/)) for sentiment analysis. Use the master word list, combined in two columns (sentiment and word).*

*Note: This data requires license to use for commercial application. Please check their website.*

```
# Load sentiment data
sentiment_df = pd.read_csv('/content/drive/My Drive/Colab Notebooks/proj2/src/data/LoughranMcDonald/LoughranMcDonald_SentimentWordLists_2018.csv')
print(sentiment_df.shape)
sentiment_df
```

(4140, 2)

| sentiment | word |
|---|---|

```
# Make all words lower case
sentiment_df['word'] = sentiment_df['word'].str.lower()
sentiments = sentiment_df['sentiment'].unique()
sentiment_df.groupby(by=['sentiment']).count()
```

| | word |
|---|---|
| **sentiment** | |
| **Constraining** | 184 |
| **Litigious** | 904 |
| **Negative** | 2355 |
| **Positive** | 354 |
| **StrongModal** | 19 |
| **Uncertainty** | 297 |
| **WeakModal** | 27 |

```
sentiment_dict = { sentiment: sentiment_df.loc[sentiment_df['sentiment']==sentiment]['word'].values.tolist() for sentiment in sentiments}
sentiment_dict
```

```
       'unseasonable',
       'unseasonably',
       'unsettled',
       'unspecific',
       'unspecified',
       'untested',
       'unusual',
       'unusually',
       'unwritten',
       'vagaries',
       'vague',
       'vaguely',
       'vagueness',
       'vaguenesses',
       'vaguer',
       'vaguest',
       'variability',
       'variable',
       'variables',
       'variably',
       'variance',
       'variances',
       'variant',
       'variants',
       'variation',
       'variations',
       'varied',
       'varies',
       'vary',
```

```
     'varying',
     'volatile',
     'volatilities',
     'volatility'],
    'WeakModal': ['almost',
     'apparently',
     'appeared',
     'appearing',
     'appears',
     'conceivable',
     'could',
     'depend',
     'depended',
     'depending',
     'depends',
     'may',
     'maybe',
     'might',
     'nearly',
     'occasionally',
     'perhaps',
     'possible',
     'possibly',
     'seldom',
     'seldomly',
     'sometimes',
     'somewhat',
     'suggest',
     'suggests',
     'uncertain',
     'uncertainly']}
```

## Analyze the tone

*With negation without lemmatization*

```
# Consider Negation
negate = ["aint", "arent", "cannot", "cant", "couldnt", "darent", "didnt", "doesnt", "ain't", "aren't", "can't",
          "couldn't", "daren't", "didn't", "doesn't", "dont", "hadnt", "hasnt", "havent", "isnt", "mightnt", "mustnt",
          "neither", "don't", "hadn't", "hasn't", "haven't", "isn't", "mightn't", "mustn't", "neednt", "needn't",
          "never", "none", "nope", "nor", "not", "nothing", "nowhere", "oughtnt", "shant", "shouldnt", "wasnt",
          "werent", "oughtn't", "shan't", "shouldn't", "wasn't", "weren't", "without", "wont", "wouldnt", "won't",
          "wouldn't", "rarely", "seldom", "despite", "no", "nobody"]

def negated(word):
    """
    Determine if preceding word is a negation word
    """
    if word.lower() in negate:
        return True
    else:
        return False

def tone_count_with_negation_check(dict, article):
    """
```

```
    Count positive and negative words with negation check. Account for simple negation only for positive words.
    Simple negation is taken to be observations of one of negate words occurring within three words
    preceding a positive words.
    """
    pos_count = 0
    neg_count = 0
    tone_score = 0

    pos_words = []
    neg_words = []

    input_words = re.findall(r'\b([a-zA-Z]+n\'t|[a-zA-Z]+\'s|[a-zA-Z]+)\b', article.lower())
    word_count = len(input_words)

    for i in range(0, word_count):
        if input_words[i] in dict['Negative']:
            neg_count += 1
            neg_words.append(input_words[i])
        if input_words[i] in dict['Positive']:
            if i >= 3:
                if negated(input_words[i - 1]) or negated(input_words[i - 2]) or negated(input_words[i - 3]):
                    neg_count += 1
                    neg_words.append(input_words[i] + ' (with negation)')
                else:
                    pos_count += 1
                    pos_words.append(input_words[i])
            elif i == 2:
                if negated(input_words[i - 1]) or negated(input_words[i - 2]):
                    neg_count += 1
                    neg_words.append(input_words[i] + ' (with negation)')
                else:
                    pos_count += 1
                    pos_words.append(input_words[i])
            elif i == 1:
                if negated(input_words[i - 1]):
                    neg_count += 1
                    neg_words.append(input_words[i] + ' (with negation)')
                else:
                    pos_count += 1
                    pos_words.append(input_words[i])
            elif i == 0:
                pos_count += 1
                pos_words.append(input_words[i])

    if word_count > 0:
        tone_score = 100 * (pos_count - neg_count) / word_count
    else:
        tone_score = 0

    results = [tone_score, word_count, pos_count, neg_count, pos_words, neg_words]

    return results

columns = ['tone_score', 'word_count', 'n_pos_words', 'n_neg_words', 'pos_words', 'neg_words']
```

```
# Analyze tone for original text dataframe
print(text_df.shape)
tone_keyword_lm = [tone_count_with_negation_check(sentiment_dict, x) for x in tqdm(text_df['text'], total=text_df.shape[0])]
tone_keyword_lm_df = pd.DataFrame(tone_keyword_lm, columns=columns)
text_df = pd.concat([text_df, tone_keyword_lm_df.reindex(text_df.index)], axis=1)
text_df
```

```
(1071, 11)
```

100% 1071/1071 [01:37<00:00, 11.02it/s]

| | type | date | title | speaker | word_count | decision | rate | next_meeting | next_decision | next_rate | text | tone_score | word_count | n_pos_words |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | statement | 1994-02-04 | FOMC Statement | Alan Greenspan | 99 | 1 | 3.25 | 1994-02-28 | 0 | 3.25 | Chairman Alan Greenspan announced today that t... | 0.000000 | 99 | 1 |
| **1** | statement | 1994-03-22 | FOMC Statement | Alan Greenspan | 40 | 1 | 3.5 | 1994-04-18 | 1 | 3.75 | Chairman Alan Greenspan announced today that t... | 0.000000 | 40 | 0 |
| | | | | | | | | | | | Chairman Alan | | | |

```
# Analyze tone for training dataframe
tone_lmdict_list = []
for doc_type in doc_types:
    tone_lmdict = [tone_count_with_negation_check(sentiment_dict, x)[0] for x in tqdm(train_df[doc_type],
                                                                                   total=train_df.shape[0],
                                                                                   desc=doc_type)]

    tone_lmdict_list.append(tone_lmdict)

train_df['tone'] = np.mean(tone_lmdict_list, axis=0)
train_df
```

statement: 100% 237/237 [00:03<00:00, 61.10it/s]

minutes: 100% 237/237 [00:37<00:00, 6.36it/s]

presconf_script: 100% 237/237 [00:39<00:00, 6.04it/s]

speech: 100% 237/237 [00:32<00:00, 7.35it/s]

testimony: 100% 237/237 [00:12<00:00, 18.33it/s]

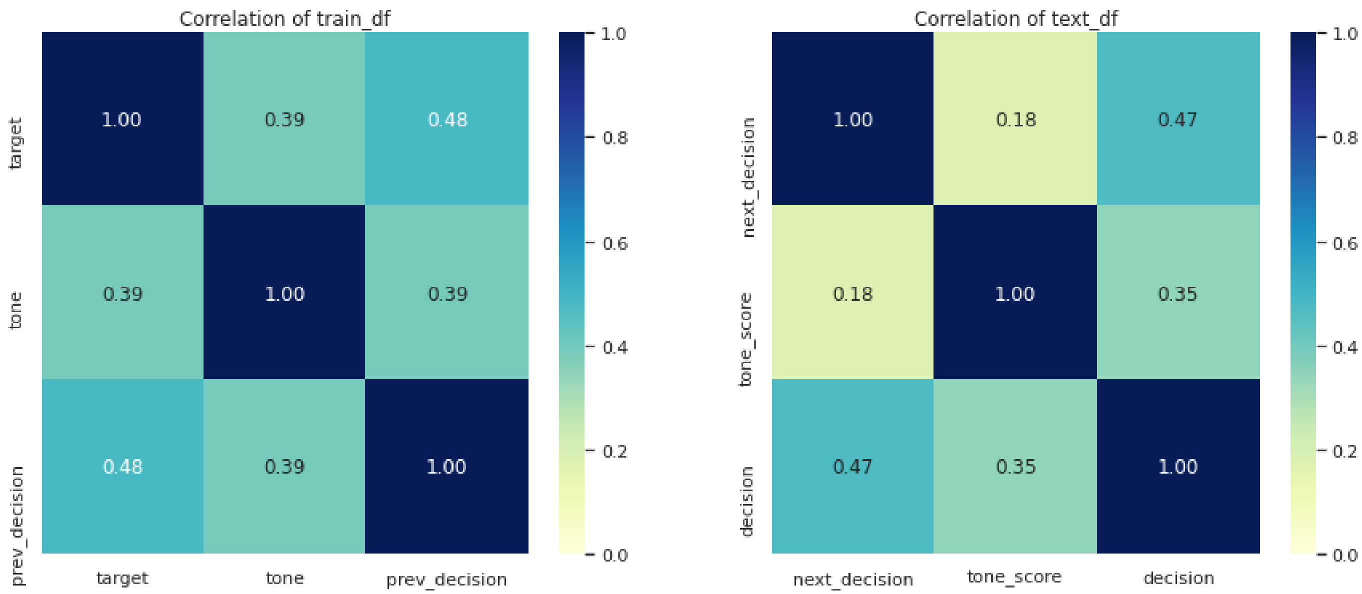
| | target | prev_decision | GDP_diff_prev | PMI_value | Employ_diff_prev | Rsales_diff_year | Unemp_diff_prev | Inertia_diff | Hsales_diff_year | Balanced_diff | state |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **date** | | | | | | | | | | | |
| **1993-02-18** | 0 | 0 | 1.043165 | 55.8 | 261.0 | 1.807631 | 0.000000 | -0.015902 | 14.901418 | 0.035879 | |
| **1993-05-18** | 0 | 0 | 0.167400 | 50.2 | 126.0 | 3.092456 | 0.066667 | -0.000720 | 13.455236 | 0.111134 | |
| **1993-07-07** | 0 | 0 | 0.167400 | 49.6 | 226.5 | 4.263357 | 0.000000 | 0.050013 | 13.446869 | -0.016140 | |
| **1993-08-17** | 0 | 0 | 0.582420 | 50.2 | 243.5 | 4.611673 | 0.066667 | 0.001967 | 11.927296 | 0.028625 | |
| **1993-09-21** | 0 | 0 | 0.582420 | 50.7 | 228.5 | 4.894733 | 0.100000 | -0.006682 | 10.302509 | -0.010715 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **2020-03-03** | -1 | 0 | 0.527469 | 50.9 | 199.0 | 2.786082 | 0.000000 | 0.007269 | 13.625558 | -0.026466 | Inform rece sinc Fe Oper |

```
text_df
```

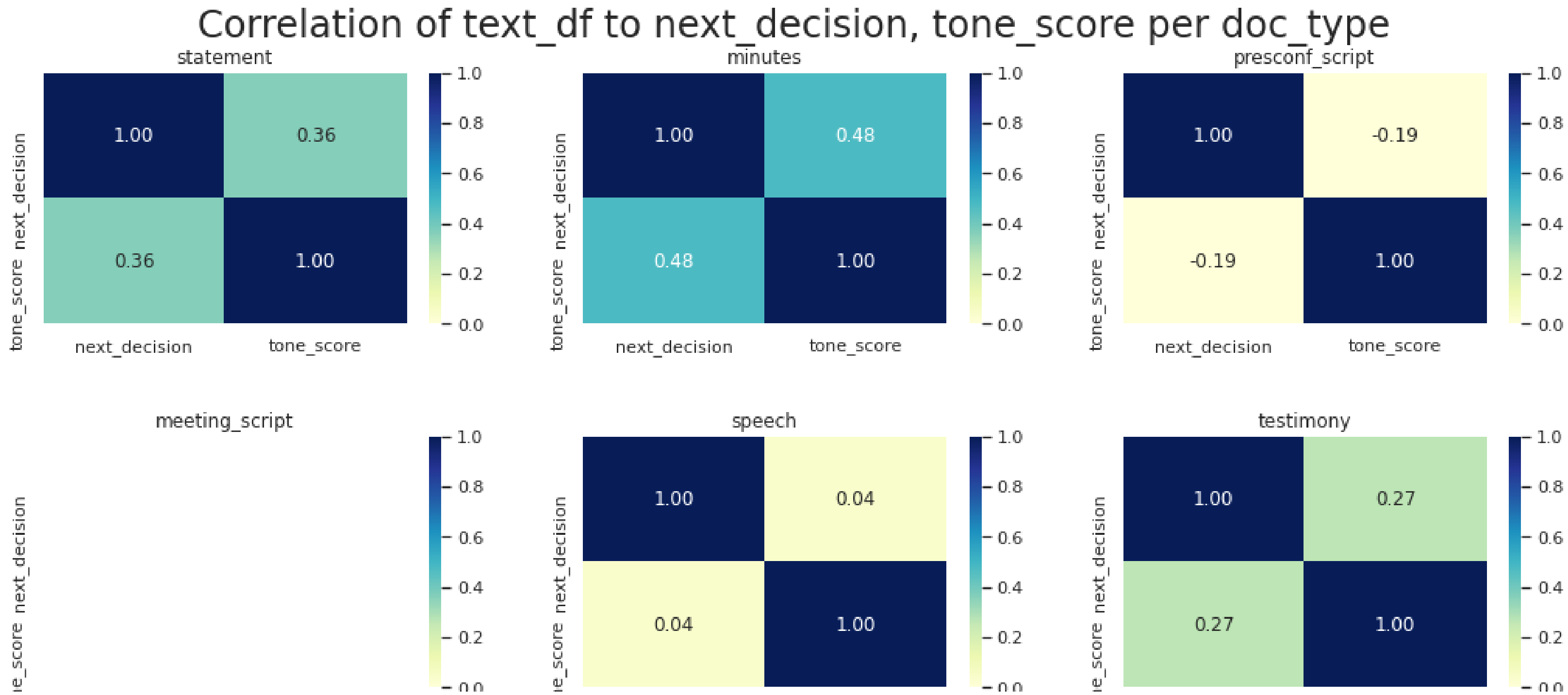
| | type | date | title | speaker | word_count | decision | rate | next_meeting | next_decision | next_rate | text | tone_score | word_count | n_pos_words |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | statement | 1994-02-04 | FOMC Statement | Alan Greenspan | 99 | 1 | 3.25 | 1994-02-28 | 0 | 3.25 | Chairman Alan Greenspan announced today that t... | 0.000000 | 99 | 1 |
| **1** | statement | 1994-03-22 | FOMC Statement | Alan Greenspan | 40 | 1 | 3.5 | 1994-04-18 | 1 | 3.75 | Chairman Alan Greenspan announced today that t... | 0.000000 | 40 | 0 |
| **2** | statement | 1994-04-18 | FOMC Statement | Alan Greenspan | 37 | 1 | 3.75 | 1994-05-17 | 1 | 4.25 | Chairman Alan Greenspan announced today that t... | 0.000000 | 37 | 0 |
| **3** | statement | 1994-05-17 | FOMC Statement | Alan Greenspan | 57 | 1 | 4.25 | 1994-07-06 | 0 | 4.25 | In taking the discount action, the Board appro... | 0.000000 | 57 | 0 |
| **4** | statement | 1994-08-16 | FOMC Statement | Alan Greenspan | 51 | 1 | 4.75 | 1994-09-27 | 0 | 4.75 | In taking the discount rate action, the Board ... | 0.000000 | 51 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **1066** | testimony | 2020-05-19 | Coronavirus and CARES Act | Jerome Powell | 1802 | <NA> | None | 2020-06-10 | 0 | 0.00 | I would like to begin by acknowledging the tra... | -0.665927 | 1802 | 33 |
| **1067** | testimony | 2020-06-16 | Semiannual Monetary Policy Report to the Congress | Jerome Powell | 1433 | <NA> | None | 2020-07-29 | 0 | 0.00 | Our country continues to face a difficult and ... | -0.907188 | 1433 | 30 |
| **1068** | testimony | 2020-06-30 | Coronavirus and CARES Act | Jerome Powell | 2759 | <NA> | None | 2020-07-29 | 0 | 0.00 | We meet as the pandemic continues to cause tre... | -0.108735 | 2759 | 46 |
| **1069** | testimony | 2020-09-22 | Coronavirus Aid, Relief, and Economic Security... | Jerome Powell | 2400 | <NA> | None | 2020-11-05 | 0 | NaN | Chairwoman Waters, Ranking Member McHenry, and... | -0.041667 | 2400 | 25 |
| | | | Coronavirus | | | | | | | | The Federal | | | |

```
# Show corelations to next_decision
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15,6))

corr_columns = ['target', 'tone', 'prev_decision']
sns.heatmap(train_df[corr_columns].astype(float).corr(), cmap="YlGnBu", annot=True, fmt=".2f", ax=ax1, vmin=0, vmax=1)
```

```
sns.heatmap(train_df[corr_columns].astype(float).corr(), cmap="YlGnBu", annot=True, fmt=".2f", ax=ax1, vmin=0, vmax=1)
ax1.set_title("Correlation of train_df")

corr_columns = ['next_decision', 'tone_score', 'decision']
tmp_df = pd.DataFrame()
for column in corr_columns:
    tmp_df[column] = pd.to_numeric(text_df[column], errors='coerce')
sns.heatmap(tmp_df.astype(float).corr(), cmap="YlGnBu", annot=True, fmt=".2f", ax=ax2, vmin=0, vmax=1)
ax2.set_title("Correlation of text_df")

plt.show()
```




```
# Per document type
corr_columns = ['next_decision', 'tone_score', 'type']
doc_types = ['statement', 'minutes', 'presconf_script', 'meeting_script', 'speech', 'testimony']

fig, ((ax1, ax2, ax3), (ax4, ax5, ax6)) = plt.subplots(2, 3, figsize=(15,7))

axes = [ax1, ax2, ax3, ax4, ax5, ax6]
df = text_df[corr_columns]
for i, doc_type in enumerate(doc_types):
    sns.heatmap(df.loc[df['type'] == doc_type].drop(columns=['type']).astype(float).corr(), cmap="YlGnBu", annot=True, fmt=".2f", vmin=0, vmax=1, a
    axes[i].set_title(doc_type)

fig.suptitle('Correlation of text_df to next_decision, tone_score per doc_type', fontsize=24)
fig.tight_layout(pad=3.0)
plt.show()
plt.savefig(graph_dir + 'corr_per_doc_type_text_df.png', )#bbox_inches='tight')
```




## Tokenize and vectorize

```
def lemmatize_word(word):
    wnl = nltk.stem.WordNetLemmatizer()
    return wnl.lemmatize(wnl.lemmatize(word, 'n'), 'v')

def tokenize_df(df, col='text'):
    tokenized = []
    wnl = nltk.stem.WordNetLemmatizer()
    for text in tqdm(df[col]):
        # Filter alphabet words only and non stop words, make it loser case
        words = [word.lower() for word in word_tokenize(text) if ((word.isalpha()==1) & (word not in stop))]
        # Lemmatize words
        tokens = [lemmatize_word(word) for word in words]
        tokenized.append(tokens)
    return tokenized
```

### Tokenize text_df

```
tokenized_org = tokenize_df(text_df)
print('len(tokenized_org): ', len(tokenized_org))
print(tokenized_org[0])
```

100% 1071/1071 [00:38<00:00, 27.64it/s]

```
len(tokenized_org):  1071
['chairman', 'alan', 'greenspan', 'announce', 'today', 'federal', 'open', 'market', 'committee', 'decide', 'increase', 'slightly', 'degree', 'pressure', 'reserve',
```

```
# Concat the list to create docs
lemma_docs_org = [" ".join(words) for words in tokenized_org]
print('len(lemma_docs_org): ', len(lemma_docs_org))
print(lemma_docs_org[0])
```

```
len(lemma_docs_org):  1071
chairman alan greenspan announce today federal open market committee decide increase slightly degree pressure reserve position the action expect associate small in
```

```
# Create a list of all the words in the dataframe
all_words_org = [word for text in tokenized_org for word in text]
print('len(all_words_org): ', len(all_words_org))
print(all_words_org[0])

# Counter object of all the words
counts_org = Counter(all_words_org)
print('len(counts_org): ', len(counts_org))

# Create a Bag of Word, sorted by the count of words
bow_org = sorted(counts_org, key=counts_org.get, reverse=True)
print('bow_org[:20]', bow_org[:20])

# Indexing vocabrary, starting from 1.
vocab_org = {word: ii for ii, word in enumerate(counts_org, 1)}
id2vocab_org = {v: k for k, v in vocab_org.items()}

print("vocab_org['chairman']: ", vocab_org['chairman'])
print("vocab_org['market']: ", vocab_org['market'])
```

```
len(all_words_org):  1813378
chairman
len(counts_org):  28024
bow_org[:20] ['market', 'rate', 'the', 'inflation', 'economic', 'financial', 'price', 'policy', 'federal', 'bank', 'committee', 'would', 'increase', 'growth', 'yea
vocab_org['chairman']:  1
vocab_org['market']:  8
```

```
# Create token id list
token_ids_org = [[vocab_org[word] for word in text_words] for text_words in tokenized_org]
print(len(token_ids_org))
```

```
1071
```

```
# Add to the dataframe
text_df['tokenized'] = tokenized_org
text_df['token_ids'] = token_ids_org
```

```
# # Filter by frequency of words
# # This time, switch it off as the frequency is already considered while creating the vocabrary

# freq = {}
# num_words = len(all_words)
# print('len(all_words): ', len(all_words))

# for key in counts:
#     freq[key] = counts[key]/num_words

# print('len(freq): ', len(freq))
# print(freq['rate'])

# low_cutoff = 0.000001
# high_cutoff = 20

# K_most_common, K_most_common_values = zip(*counts.most_common()[:high_cutoff])

# filtered_words = [word for word in freqs if (freqs[word] > low_cutoff and word not in K_most_common)]
# print(K_most_common)
# print('len(filtered_words): ', len(filtered_words))
```

## Tokenize train_df

```
tokenized = tokenize_df(train_df)
print('len(tokenized): ', len(tokenized))
print(tokenized[0])

# Concat the list to create docs
lemma_docs = [" ".join(words) for words in tokenized]
print('len(lemma_docs): ', len(lemma_docs))
print(lemma_docs[0])

# Create a list of all the words in the dataframe
all_words = [word for text in tokenized for word in text]
print('len(all_words): ', len(all_words))
print(all_words[0])

# Counter object of all the words
counts = Counter(all_words)
print('len(counts): ', len(counts))

# Create a Bag of Word, sorted by the count of words
bow = sorted(counts, key=counts.get, reverse=True)
print('bow[:20]', bow[:20])
```

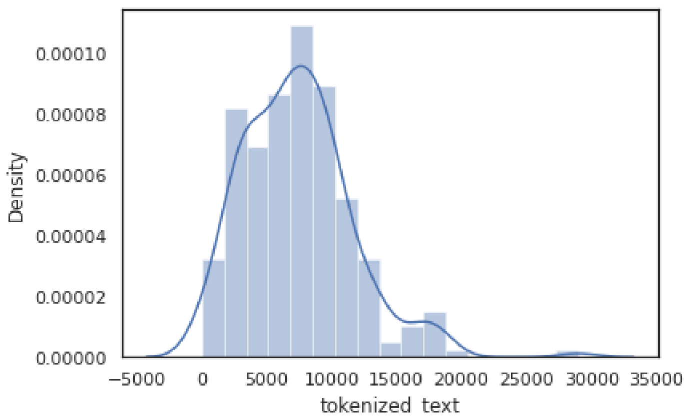
```
# Indexing vocabrary, starting from 1.
vocab = {word: ii for ii, word in enumerate(counts, 1)}
id2vocab = {v: k for k, v in vocab.items()}

# Create token id list
token_ids = [[vocab[word] for word in text_words] for text_words in tokenized]
print(len(token_ids))

# Add to the dataframe
train_df['tokenized'] = tokenized
train_df['token_ids'] = token_ids
train_df['tokenized_text'] = train_df['tokenized'].apply(lambda x: " ".join(x))
```

100% 237/237 [00:35<00:00, 6.61it/s]

```
len(tokenized):  237
['the', 'secretary', 'report', 'advice', 'election', 'reserve', 'bank', 'member', 'alternate', 'member', 'federal', 'open', 'market', 'committee', 'period', 'comme
len(lemma_docs):  237
the secretary report advice election reserve bank member alternate member federal open market committee period commence january end december receive individual exe
len(all_words):  1751290
the
len(counts):  26920
bow[:20] ['market', 'rate', 'the', 'inflation', 'economic', 'financial', 'price', 'federal', 'policy', 'bank', 'committee', 'growth', 'increase', 'would', 'year',
237
```

```
sns.distplot(train_df['tokenized_text'].apply(lambda x: len(x.split())))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f80759787b8>
```




```
len(token_ids[0])
```

```
4809
```

## Lemmatize sentiment

```
# pd.get_dummies(sentiment_df, prefix=None, dtype=bool)
# sentiment_df.columns = [column.lower() for column in sentiment_df.columns]

# Lemmertize sentiment words as well
lemma_sentiment_df = sentiment_df.copy(deep=True)
lemma_sentiment_df['word'] = [lemmatize_word(word) for word in lemma_sentiment_df['word']]
# Drop duplicates
lemma_sentiment_df = sentiment_df.drop_duplicates('word')
# Sentiment list
lemma_sentiments = list(lemma_sentiment_df['sentiment'].unique())
```

```
lemma_sentiment_df.groupby(by=['sentiment']).count()
```

| | word |
|---|---|
| **sentiment** | |
| **Constraining** | 145 |
| **Litigious** | 750 |
| **Negative** | 2355 |
| **Positive** | 354 |
| **StrongModal** | 15 |
| **Uncertainty** | 257 |

## Tfidf

```
from sklearn.feature_extraction.text import TfidfVectorizer

def get_tfidf(sentiment_words, docs):
    """
    Generate TFIDF values from documents for a certain sentiment

    Parameters
    ----------
    sentiment_words: Pandas Series
        Words that signify a certain sentiment
    docs : list of str
        List of documents used to generate bag of words

    Returns
```

```
    tfidf : 2-d Numpy Ndarray of float
        TFIDF sentiment for each document
        The first dimension is the document.
        The second dimension is the word.
    """
    vectorizer = TfidfVectorizer(analyzer='word', vocabulary=sentiment_words)
    tfidf = vectorizer.fit_transform(docs)
    features = vectorizer.get_feature_names()

    return tfidf.toarray()
```

## Text dataframe

```
# Using the get_tfidf function, let's generate the TFIDF values for all the documents.
sentiment_tfidf_org = {
        sentiment: get_tfidf(lemma_sentiment_df.loc[lemma_sentiment_df['sentiment'] == sentiment]['word'], lemma_docs_org)
        for sentiment in lemma_sentiments}

print(len(sentiment_tfidf_org['Negative']))
print(len(sentiment_tfidf_org['Negative'][0]))
```

```
1071
2355
```

```
text_df.shape
```

```
(1071, 19)
```

```
for sentiment in lemma_sentiments:
    text_df['tfidf_' + sentiment] = list(sentiment_tfidf_org[sentiment])

text_df
```

| | type | date | title | speaker | word_count | decision | rate | next_meeting | next_decision | next_rate | text | tone_score | word_count | n_pos_words |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | statement | 1994-02-04 | FOMC Statement | Alan Greenspan | 99 | 1 | 3.25 | 1994-02-28 | 0 | 3.25 | Chairman Alan Greenspan announced today that t... | 0.000000 | 99 | 1 |
| **1** | statement | 1994-03-22 | FOMC Statement | Alan Greenspan | 40 | 1 | 3.5 | 1994-04-18 | 1 | 3.75 | Chairman Alan Greenspan announced today that t... | 0.000000 | 40 | 0 |
| **2** | statement | 1994-04-18 | FOMC Statement | Alan Greenspan | 37 | 1 | 3.75 | 1994-05-17 | 1 | 4.25 | Chairman Alan Greenspan announced today that t... | 0.000000 | 37 | 0 |
| **3** | statement | 1994-05-17 | FOMC Statement | Alan Greenspan | 57 | 1 | 4.25 | 1994-07-06 | 0 | 4.25 | In taking the discount action, the Board appro... | 0.000000 | 57 | 0 |
| **4** | statement | 1994-08-16 | FOMC Statement | Alan Greenspan | 51 | 1 | 4.75 | 1994-09-27 | 0 | 4.75 | In taking the discount rate action, the Board ... | 0.000000 | 51 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **1066** | testimony | 2020-05-19 | Coronavirus and CARES Act | Jerome Powell | 1802 | <NA> | None | 2020-06-10 | 0 | 0.00 | I would like to begin by acknowledging the tra... | -0.665927 | 1802 | 33 |
| **1067** | testimony | 2020-06-16 | Semiannual Monetary Policy Report to the Congress | Jerome Powell | 1433 | <NA> | None | 2020-07-29 | 0 | 0.00 | Our country continues to face a difficult and ... | -0.907188 | 1433 | 30 |
| **1068** | testimony | 2020-06-30 | Coronavirus and CARES Act | Jerome Powell | 2759 | <NA> | None | 2020-07-29 | 0 | 0.00 | We meet as the pandemic continues to cause tre... | -0.108735 | 2759 | 46 |
| | | | Coronavirus ... Security... | | | | | | | | Chairwoman Waters, ... and... | | | |

Train dataframe

```python
# Using the get_tfidf function, let's generate the TFIDF values for all the documents.
sentiment_tfidf = {
        sentiment: get_tfidf(lemma_sentiment_df.loc[lemma_sentiment_df['sentiment'] == sentiment]['word'], lemma_docs)
        for sentiment in lemma_sentiments}

print(len(sentiment_tfidf['Negative']))
print(len(sentiment_tfidf['Negative'][0]))

for sentiment in lemma_sentiments:
    train_df['tfidf_' + sentiment] = list(sentiment_tfidf[sentiment])

train_df
```

237
2355

| date | target | prev_decision | GDP_diff_prev | PMI_value | Employ_diff_prev | Rsales_diff_year | Unemp_diff_prev | Inertia_diff | Hsales_diff_year | Balanced_diff | state |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **1993-02-18** | 0 | 0 | 1.043165 | 55.8 | 261.0 | 1.807631 | 0.000000 | -0.015902 | 14.901418 | 0.035879 | |
| **1993-05-18** | 0 | 0 | 0.167400 | 50.2 | 126.0 | 3.092456 | 0.066667 | -0.000720 | 13.455236 | 0.111134 | |
| **1993-07-07** | 0 | 0 | 0.167400 | 49.6 | 226.5 | 4.263357 | 0.000000 | 0.050013 | 13.446869 | -0.016140 | |
| **1993-08-17** | 0 | 0 | 0.582420 | 50.2 | 243.5 | 4.611673 | 0.066667 | 0.001967 | 11.927296 | 0.028625 | |

## Cosine Similarity

*Using the TFIDF values, we'll calculate the cosine similarity and plot it over time. Implement `get_cosine_similarity` to return the cosine similarities between each tick in time. Since the input, `tfidf_matrix`, is a TFIDF vector for each time period in order, you just need to computer the cosine similarities for each neighboring vector.*

rec

```
from sklearn.metrics.pairwise import cosine_similarity

def get_cosine_similarity(tfidf_matrix):
    """
    Get cosine similarities for each neighboring TFIDF vector/document

    Parameters
    ----------
    tfidf : 2-d Numpy Ndarray of float
        TFIDF sentiment for each document
        The first dimension is the document.
        The second dimension is the word.

    Returns
    -------
    cosine_similarities : list of float
        Cosine similarities for neighboring documents
    """
```

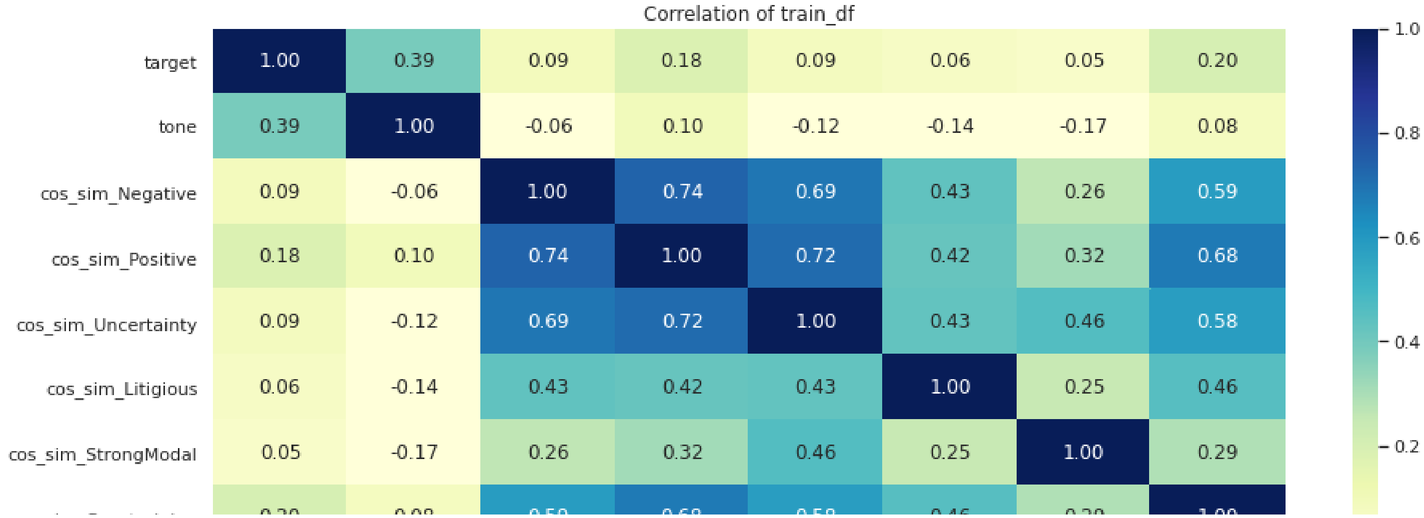
```
    #print(tfidf_matrix)
    return [cosine_similarity(u.reshape(1,-1), v.reshape(1,-1))[0][0].tolist() for u, v in zip(tfidf_matrix, tfidf_matrix[1:])]


cosine_similarities = {
    sentiment_name: get_cosine_similarity(sentiment_values)
    for sentiment_name, sentiment_values in sentiment_tfidf.items()}

print(len(cosine_similarities['Negative']))
```

```
236
```

```
for sentiment in lemma_sentiments:
    # Add 0 to the first element as there is no comparison available to a previous value
    cosine_similarities[sentiment].insert(0, 0)
    train_df['cos_sim_' + sentiment] = cosine_similarities[sentiment]

train_df
```

| date | target | prev_decision | GDP_diff_prev | PMI_value | Employ_diff_prev | Rsales_diff_year | Unemp_diff_prev | Inertia_diff | Hsales_diff_year | Balanced_diff | state |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1993-02-18 | 0 | 0 | 1.043165 | 55.8 | 261.0 | 1.807631 | 0.000000 | -0.015902 | 14.901418 | 0.035879 | |
| 1993-05-18 | 0 | 0 | 0.167400 | 50.2 | 126.0 | 3.092456 | 0.066667 | -0.000720 | 13.455236 | 0.111134 | |
| 1993-07-07 | 0 | 0 | 0.167400 | 49.6 | 226.5 | 4.263357 | 0.000000 | 0.050013 | 13.446869 | -0.016140 | |
| 1993-08-17 | 0 | 0 | 0.582420 | 50.2 | 243.5 | 4.611673 | 0.066667 | 0.001967 | 11.927296 | 0.028625 | |
| 1993-09-21 | 0 | 0 | 0.582420 | 50.7 | 228.5 | 4.894733 | 0.100000 | -0.006682 | 10.302509 | -0.010715 | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 2020- | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | Inform rec |

```
# Show corelations to target
fig, ax = plt.subplots(figsize=(15,6))
corr_columns = ['target', 'tone', 'cos_sim_Negative', 'cos_sim_Positive', 'cos_sim_Uncertainty', 'cos_sim_Litigious', 'cos_sim_StrongModal', 'cos_s
sns.heatmap(train_df[corr_columns].astype(float).corr(), cmap="YlGnBu", annot=True, fmt=".2f", ax=ax, vmin=0, vmax=1)
ax.set_title("Correlation of train_df")
plt.show()
plt.savefig(graph_dir + 'corr_per_doc_type_train_df.png', )#bbox_inches='tight')
```




## Convert target class for classification

```
def convert_class(x):
    if x == 1:
        return 2
    elif x == 0:
        return 1
    elif x == -1:
        return 0
```

```
train_df['target'] = train_df['target'].map(convert_class)
```

```
train_df['prev_decision'] = train_df['prev_decision'].map(convert_class)
```

# Modeling and Training

## Setup

```
# Use Stratified KFold Cross Validation
# Training data is not so many, keep n_split <= 5
kfold = StratifiedKFold(n_splits=3)
kfold
```

```
    StratifiedKFold(n_splits=3, random_state=None, shuffle=False)
```

```python
def metric(y_true, y_pred):
    acc = accuracy_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred, average='macro')
    return acc, f1

scoring = {'Accuracy': 'accuracy', 'F1': 'f1_macro'}
refit = 'F1'


def train_grid_search(estimator, param_grid, scoring, refit, cv=5, verbose=1, plot=True):
    model = GridSearchCV(estimator, param_grid=param_grid, cv=cv, scoring=scoring, verbose=verbose,
                         refit=refit, n_jobs=-1, return_train_score=True)
    model.fit(X_train, Y_train)

    results = model.cv_results_
    best_estimator = model.best_estimator_
    train_scores = results['mean_train_' + refit]
    test_scores = results['mean_test_' + refit]
    train_time = results['mean_fit_time']

    print("Best Score: ", model.best_score_)
    print("Best Param: ", model.best_params_)

    pred_train = best_estimator.predict(X_train)
    pred_test = best_estimator.predict(X_test)

    acc, f1 = metric(Y_train, pred_train)
    logger.info('Training - acc: %.8f, f1: %.8f' % (acc, f1))
    acc, f1 = metric(Y_test, pred_test)
    logger.info('Test - acc: %.8f, f1: %.8f' % (acc, f1))

    if plot:
        fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5))
        fig.suptitle("GridSearchCV Result", fontsize=20)

        ### First plot ###
        ax1.plot(train_scores, test_scores, 'bo')
        ax1.set_title("Train Score v.s. Test Score", fontsize=16)
        ax1.set_xlabel("Train Score")
        ax1.set_ylabel("Test Score")
        ax1.set_xlim(0, 1)
        ax1.set_ylim(0, 1)
        ax1.grid(True)

        ### Second plot ###
        x_param = list(param_grid.keys())[0]
        x_param_min = np.min(list(param_grid.values())[0])
        x_param_max = np.max(list(param_grid.values())[0])

        ax2.set_title("Score over the first param", fontsize=16)
        ax2.set_xlabel(x_param)
        ax2.set_ylabel("Score")
        ax2.set_xlim(x_param_min, x_param_max)
```

```
ax2.set_ylim(0, 1)

# Get the regular numpy array from the MaskedArray
X_axis = np.array(results['param_' + x_param].data, dtype=float)

for scorer, color in zip(sorted(scoring), ['r', 'g']):
    for sample, style in (('train', '--'), ('test', '-')):
        sample_score_mean = results['mean_%s_%s' % (sample, scorer)]
        sample_score_std = results['std_%s_%s' % (sample, scorer)]
        ax2.fill_between(X_axis, sample_score_mean - sample_score_std,
                        sample_score_mean + sample_score_std,
                        alpha=0.1 if sample == 'test' else 0, color=color)
        ax2.plot(X_axis, sample_score_mean, style, color=color,
                alpha=1 if sample == 'test' else 0.7,
                label="%s (%s)" % (scorer, sample.capitalize()))

    best_index = np.nonzero(results['rank_test_%s' % scorer] == 1)[0][0]
    best_score = results['mean_test_%s' % scorer][best_index]

    # Plot a dotted vertical line at the best score for that scorer marked by x
    ax2.plot([X_axis[best_index], ] * 2, [0, best_score],
            linestyle='-.', color=color, marker='x', markeredgewidth=3, ms=8)

    # Annotate the best score for that scorer
    ax2.annotate("%0.2f" % best_score,
                (X_axis[best_index], best_score + 0.005))

ax2.legend(loc="best")
ax2.grid(False)

### Third plot (Learning Curve) ###
# Calculate learning curve (Accuracy)
lc_acc_train_sizes, lc_acc_train_scores, lc_acc_test_scores = learning_curve(
    best_estimator, X_train, Y_train, cv=kfold, n_jobs=-1, scoring=scoring['Accuracy'],
    train_sizes=np.linspace(.1, 1.0, 5))
lc_acc_train_mean = np.mean(lc_acc_train_scores, axis=1)
lc_acc_train_std = np.std(lc_acc_train_scores, axis=1)
lc_acc_test_mean = np.mean(lc_acc_test_scores, axis=1)
lc_acc_test_std = np.std(lc_acc_test_scores, axis=1)

# Calculate learning curve (F1 Score)
lc_f1_train_sizes, lc_f1_train_scores, lc_f1_test_scores = learning_curve(
    best_estimator, X_train, Y_train, cv=kfold, n_jobs=-1, scoring=scoring['F1'],
    train_sizes=np.linspace(.1, 1.0, 5))
lc_f1_train_mean = np.mean(lc_f1_train_scores, axis=1)
lc_f1_train_std = np.std(lc_f1_train_scores, axis=1)
lc_f1_test_mean = np.mean(lc_f1_test_scores, axis=1)
lc_f1_test_std = np.std(lc_f1_test_scores, axis=1)

ax3.set_title("Learning Curve", fontsize=16)
ax3.set_xlabel("Training examples")
ax3.set_ylabel("Score")

# Plot learning curve (Accuracy)
```

```
# Plot learning curve (Accuracy)
ax3.fill_between(lc_acc_train_sizes,
                 lc_acc_train_mean - lc_acc_train_std,
                 lc_acc_train_mean + lc_acc_train_std, alpha=0.1, color="r")
ax3.fill_between(lc_acc_train_sizes,
                 lc_acc_test_mean - lc_acc_test_std,
                 lc_acc_test_mean + lc_acc_test_std, alpha=0.1, color="r")
ax3.plot(lc_acc_train_sizes, lc_acc_train_mean, 'o--', color="r",
         label="Accuracy (Train)")
ax3.plot(lc_acc_train_sizes, lc_acc_test_mean, 'o-', color="r",
         label="Accuracy (Test)")

# Plot learning curve (F1 Score)
ax3.fill_between(lc_f1_train_sizes,
                 lc_f1_train_mean - lc_f1_train_std,
                 lc_f1_train_mean + lc_f1_train_std, alpha=0.1, color="g")
ax3.fill_between(lc_f1_train_sizes,
                 lc_f1_test_mean - lc_f1_test_std,
                 lc_f1_test_mean + lc_f1_test_std, alpha=0.1, color="g")
ax3.plot(lc_f1_train_sizes, lc_f1_train_mean, 'o--', color="g",
         label="F1 (Train)")
ax3.plot(lc_f1_train_sizes, lc_f1_test_mean, 'o-', color="g",
         label="F1 (Test)")

ax3.legend(loc="best")
ax3.grid(True)

plt.tight_layout(pad=3.0)
plt.show()
plt.savefig(graph_dir + 'tgs_learning_curve_full' + '.png')#, bbox_inches='tight')

### Confusion Matrix ###
class_names = ['Lower', 'Hold', 'Raise']
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15, 10))
fig.suptitle("Confusion Matrix", fontsize=20)

plot_confusion_matrix(best_estimator, X_train, Y_train, display_labels=class_names,
                      cmap=plt.cm.Blues, normalize=None, ax=ax1)
ax1.set_title("Train Data: Actual Count")
ax1.grid(False)

plot_confusion_matrix(best_estimator, X_train, Y_train, display_labels=class_names,
                      cmap=plt.cm.Blues, normalize='all', ax=ax2)
ax2.set_title=("Train Data: Normalized")
ax2.grid(False)

plot_confusion_matrix(best_estimator, X_test, Y_test, display_labels=class_names,
                      cmap=plt.cm.Blues, normalize=None, ax=ax3)
ax3.set_title=("Test Data: Actual Count")
ax3.grid(False)

plot_confusion_matrix(best_estimator, X_test, Y_test, display_labels=class_names,
                      cmap=plt.cm.Blues, normalize='all', ax=ax4)
ax4.set_title("Test Data: Normalized")
```

```
        ax4.grid(False)

        plt.tight_layout(pad=3.0)
        plt.show()
        plt.savefig(graph_dir + 'conf_mats_full' + '.png')#, bbox_inches='tight')

    return model
```

## I. Cosin Similarity

### Train and Test Data

```
train_df.columns
```

```
     Index(['target', 'prev_decision', 'GDP_diff_prev', 'PMI_value',
            'Employ_diff_prev', 'Rsales_diff_year', 'Unemp_diff_prev',
            'Inertia_diff', 'Hsales_diff_year', 'Balanced_diff', 'statement',
            'minutes', 'presconf_script', 'speech', 'testimony', 'text', 'tone',
            'tokenized', 'token_ids', 'tokenized_text', 'tfidf_Negative',
            'tfidf_Positive', 'tfidf_Uncertainty', 'tfidf_Litigious',
            'tfidf_StrongModal', 'tfidf_Constraining', 'cos_sim_Negative',
            'cos_sim_Positive', 'cos_sim_Uncertainty', 'cos_sim_Litigious',
            'cos_sim_StrongModal', 'cos_sim_Constraining'],
           dtype='object')
```

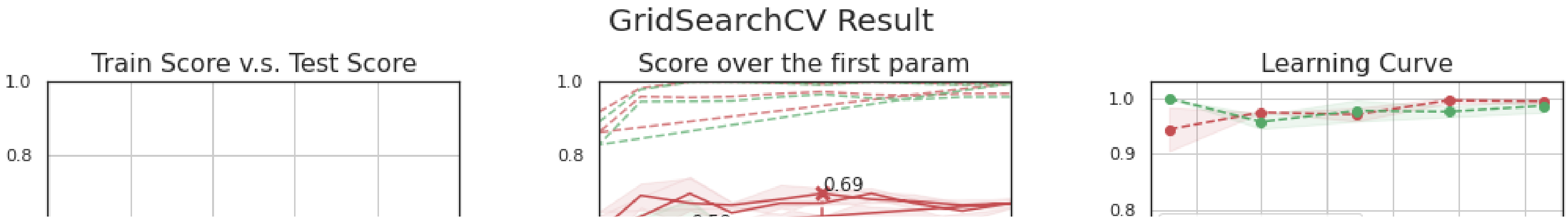
```
# X and Y data used
Y_data = train_df['target']
X_data = train_df[nontext_columns + ['tone', 'cos_sim_Negative', 'cos_sim_Positive', 'cos_sim_Uncertainty',
                                     'cos_sim_Litigious', 'cos_sim_StrongModal', 'cos_sim_Constraining']]

# Train test split (Shuffle=False will make the test data for the most recent ones)
X_train, X_test, Y_train, Y_test = \
model_selection.train_test_split(X_data.values, Y_data.values, test_size=0.2, shuffle=True)
```

### Train Model

```
# Random Forest
rf_clf = RandomForestClassifier()

# Perform Grid Search
param_grid = {'n_estimators': np.linspace(1, 60, 10, dtype=int),
              'min_samples_split': [3, 10],
              'min_samples_leaf': [3],
              'max_features': [7],
              'max_depth': [None],
              'criterion': ['gini'],
              'bootstrap': [False]}
```

```
rf_model = train_grid_search(rf_clf, param_grid, scoring, refit, cv=kfold, verbose=1, plot=True)
rf_best = rf_model.best_estimator_
```

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  53 out of  60 | elapsed:    1.0s remaining:    0.1s
[Parallel(n_jobs=-1)]: Done  60 out of  60 | elapsed:    1.2s finished
[2021-01-25 13:48:02,058][INFO] ## Training - acc: 0.98941799, f1: 0.98723232
[2021-01-25 13:48:02,060][INFO] ## Test - acc: 0.75000000, f1: 0.58127358
Best Score:  0.5907814855184087
Best Param:  {'bootstrap': False, 'criterion': 'gini', 'max_depth': None, 'max_features': 7, 'min_samples_leaf': 3, 'min_samples_split': 3, 'n_estimators': 14}
```




```
# Feature Importance
fig, ax = plt.subplots(figsize=(10,8))

indices = np.argsort(rf_best.feature_importances_)[::-1][:40]
g = sns.barplot(y=X_data.columns[indices][:40], x=rf_best.feature_importances_[indices][:40] , orient='h', ax=ax)
g.set_xlabel("Relative importance", fontsize=12)
g.set_ylabel("Features", fontsize=12)
g.tick_params(labelsize=9)
g.set_title("Feature importance")
```

```
Text(0.5, 1.0, 'Feature importance')
```

## II. Tfidf

*Use Tfidf instead of cosin similarity:*


### Train and Test Data

```
train_df
```

| date | target | prev_decision | GDP_diff_prev | PMI_value | Employ_diff_prev | Rsales_diff_year | Unemp_diff_prev | Inertia_diff | Hsales_diff_year | Balanced_diff | state |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **1993-02-18** | 1 | 1 | 1.043165 | 55.8 | 261.0 | 1.807631 | 0.000000 | -0.015902 | 14.901418 | 0.035879 | |
| **1993-05-18** | 1 | 1 | 0.167400 | 50.2 | 126.0 | 3.092456 | 0.066667 | -0.000720 | 13.455236 | 0.111134 | |
| **1993-07-07** | 1 | 1 | 0.167400 | 49.6 | 226.5 | 4.263357 | 0.000000 | 0.050013 | 13.446869 | -0.016140 | |
| **1993-08-17** | 1 | 1 | 0.582420 | 50.2 | 243.5 | 4.611673 | 0.066667 | 0.001967 | 11.927296 | 0.028625 | |

```
vocabulary=sentiment_dict['Negative']+sentiment_dict['Positive']
vocabulary
```

```
 'evicting',
 'eviction',
 'evictions',
 'evicts',
 'exacerbate',
 'exacerbated',
 'exacerbates',
 'exacerbating',
 'exacerbation',
 'exacerbations',
 'exaggerate',
 'exaggerated',
 'exaggerates',
 'exaggerating',
 'exaggeration',
 'excessive',
 'excessively',
 'exculpate',
 'exculpated',
 'exculpates',
 'exculpating',
 'exculpation',
 'exculpations',
 'exculpatory',
 'exonerate',
 'exonerated',
 'exonerates',
 'exonerating',
 'exoneration',
 'exonerations',
```

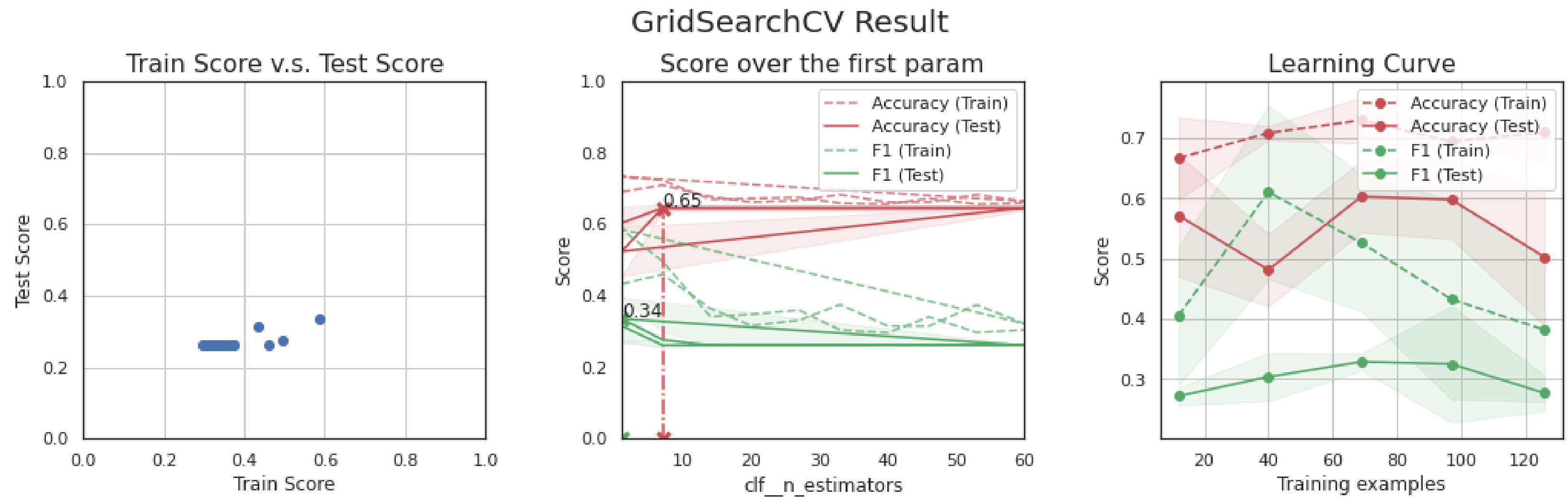
```
      'exploit',
      'exploitation',
      'exploitations',
      'exploitative',
      'exploited',
      'exploiting',
      'exploits',
      'expose',
      'exposed',
      'exposes',
      'exposing',
      'expropriate',
      'expropriated',
      'expropriates',
      'expropriating',
      'expropriation',
      'expropriations',
      'expulsion',
      'expulsions',
      'extenuating',
      'fail',
      'failed',
      'failing',
      'failings',
      'fails',
      'failure',
      'failures',
      'fallout',
      'false',
      ...]

# X and Y data used
Y_data = train_df['target']
X_data = train_df[nontext_columns + ['tone', 'tokenized_text']]

# Train test split (Shuffle=False will make the test data for the most recent ones)
X_train, X_test, Y_train, Y_test = \
model_selection.train_test_split(X_data.values, Y_data.values, test_size=0.2, shuffle=True)


import scipy
def get_numeric_data(x):
    return [record[:-2].astype(float) for record in x]

def get_text_data(x):
    return [record[-1] for record in x]

from sklearn.preprocessing import FunctionTransformer
transfomer_numeric = FunctionTransformer(get_numeric_data)
transformer_text = FunctionTransformer(get_text_data)
```

## Train Model

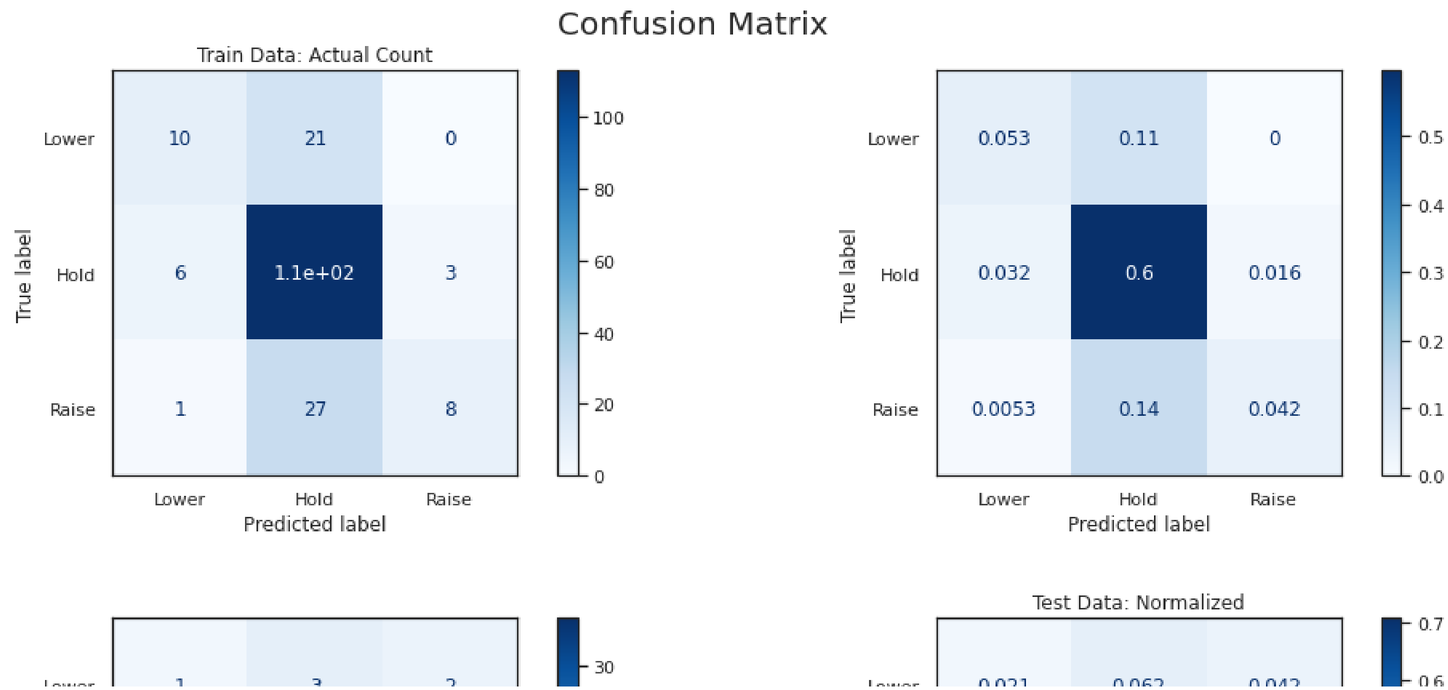
```
clf = Pipeline([
```

```
    ('features', FeatureUnion([
            ('numeric_features', Pipeline([
                ('selector', transfomer_numeric)
            ])),
             ('text_features', Pipeline([
                ('selector', transformer_text),
                ('vec', TfidfVectorizer(analyzer='word', vocabulary=vocabulary))
            ]))
         ])),
    ('clf', RandomForestClassifier())
])

pipeline = Pipeline([
    ('features', FeatureUnion([
            ('numeric_features', Pipeline([
                ('selector', transfomer_numeric)
            ])),
             ('text_features', Pipeline([
                ('selector', transformer_text),
                ('vec', TfidfVectorizer(analyzer='word'))
            ]))
         ])),
    ('clf', RandomForestClassifier())
])

# Perform Grid Search
param_grid = {'clf__n_estimators': np.linspace(1, 60, 10, dtype=int),
              'clf__min_samples_split': [3, 10],
              'clf__min_samples_leaf': [3],
              'clf__max_features': [7],
              'clf__max_depth': [None],
              'clf__criterion': ['gini'],
              'clf__bootstrap': [False]}

rf_model = train_grid_search(pipeline, param_grid, scoring, refit, cv=kfold, verbose=1, plot=True)
rf_best = rf_model.best_estimator_
```

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   28.0s
[Parallel(n_jobs=-1)]: Done  60 out of  60 | elapsed:   38.2s finished
Best Score:  0.3356701786710447
Best Param:  {'clf__bootstrap': False, 'clf__criterion': 'gini', 'clf__max_depth': None, 'clf__max_features': 7, 'clf__min_samples_leaf': 3, 'clf__min_samples_spli
[2021-01-25 13:49:59,631][INFO] ## Training - acc: 0.69312169, f1: 0.51855959
[2021-01-25 13:49:59,633][INFO] ## Test - acc: 0.72916667, f1: 0.36099391
```




```
<Figure size 432x288 with 0 Axes>
```




## III. LSTM (RNN)

*Instead of Tfidf, use LSTM. Concatenate the lstm output and the meta data at the end and dense layer to fully connect them:*

## Input Data

```
# # Split data into training and validation datasets. Use an appropriate split size.

# split_frac = 0.8

# split_idx = int(len(token_ids)*split_frac)

# train_features = token_ids[:split_idx]
# valid_features = token_ids[split_idx:]
# train_labels = Y_data[:split_idx]
# valid_labels = Y_data[split_idx:]

# print("len(token_ids): ", len(token_ids))
# print("len(train_features): ", len(train_features))
# print("len(valid_features): ", len(valid_features))
# print("len(train_labels): ", len(train_labels))
# print("len(valid_labels): ", len(valid_labels))

# X and Y data used
y_data = train_df['target']
X_data = train_df[nontext_columns + ['tone', 'token_ids']]

# Train test split (Shuffle=False will make the test data for the most recent ones)
X_train, X_valid, y_train, y_valid = \
model_selection.train_test_split(X_data.values, y_data.values, test_size=0.2, shuffle=True)

X_train_meta = get_numeric_data(X_train)
X_train_text = get_text_data(X_train)
X_valid_meta = get_numeric_data(X_valid)
X_valid_text = get_text_data(X_valid)

print('Shape of train meta', len(X_train_meta))
print('Shape of train text', len(X_train_text))
print("Shape of valid meta ", len(X_valid_meta))
print("Shape of valid text ", len(X_valid_text))

meta_size = len(X_train_meta[0])
print("Meta data size: ", meta_size)
```

```
Shape of train meta 189
Shape of train text 189
Shape of valid meta  48
Shape of valid text  48
Meta data size:  9
```

## Model

*Embed -> RNN -> Dense -> Softmax*

```python
class TextClassifier(nn.Module):
    def __init__(self, vocab_size, embed_size, lstm_size, dense_size, meta_size, output_size, lstm_layers=1, dropout=0.1):
        """
        Initialize the model
        """
        super().__init__()
        self.vocab_size = vocab_size
        self.embed_size = embed_size
        self.lstm_size = lstm_size
        self.output_size = output_size
        self.lstm_layers = lstm_layers
        self.dropout = dropout

        self.embedding = nn.Embedding(vocab_size, embed_size)
        self.lstm = nn.LSTM(embed_size, lstm_size, lstm_layers, dropout=dropout, batch_first=False)
        self.dropout = nn.Dropout(0.2)
        self.fc1 = nn.Linear(lstm_size, dense_size)
        self.fc2 = nn.Linear(dense_size + meta_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def init_hidden(self, batch_size):
        """
        Initialize the hidden state
        """

        weight = next(self.parameters()).data
        # print('initial weight size: ', weight.shape)
        # print('initial weight: ', weight)
        # print('initial weight new: ', weight.new(self.lstm_layers, batch_size, self.lstm_size))

        hidden = (weight.new(self.lstm_layers, batch_size, self.lstm_size).zero_(),
                  weight.new(self.lstm_layers, batch_size, self.lstm_size).zero_())

        return hidden

    def forward(self, nn_input_text, nn_input_meta, hidden_state):
        """
        Perform a forward pass of the model on nn_input
        """
        batch_size = nn_input_text.size(0)
        nn_input_text = nn_input_text.long()
        embeds = self.embedding(nn_input_text)
        lstm_out, hidden_state = self.lstm(embeds, hidden_state)
        # Stack up LSTM outputs, apply dropout
        lstm_out = lstm_out[-1,:,:]
        lstm_out = self.dropout(lstm_out)
        # Dense layer
        dense_out = self.fc1(lstm_out)
        # Concatinate the dense output and meta inputs
        concat_layer = torch.cat((dense_out, nn_input_meta.float()), 1)
        out = self.fc2(concat_layer)
        logps = self.softmax(out)
```

```
        return logps, hidden_state
```

## DataLoaders and Batching

*Could use keras functions. This is built from scratch:*

```
#from keras.preprocessing.text import Tokenizer
#from keras.preprocessing.sequence import pad_sequences
#MAX_LEN = 100
#tokenizer_obj = Tokenizer()
#tokenizer_obj.fit_on_texts(balanced['texts'])
#sequences = tokenizer_obj.texts_to_sequences(balanced['texts'])
#text_pad = pad_sequences(sequences, maxlen=MAX_LEN, truncating='post', padding='post')
#text_pad
```

```
def dataloader(messages, meta, labels, sequence_length=200, batch_size=16, shuffle=False):
    """
    Build a dataloader.
    """
    if shuffle:
        indices = list(range(len(messages)))
        random.shuffle(indices)
        messages = [messages[idx] for idx in indices]
        meta = [meta[idx] for idx in indices]
        labels = [labels[idx] for idx in indices]

    total_sequences = len(messages)

    for ii in range(0, total_sequences, batch_size):
        batch_messages = messages[ii: ii+batch_size]

        # First initialize a tensor of all zeros
        batch = torch.zeros((sequence_length, len(batch_messages)), dtype=torch.int64)
        for batch_num, tokens in enumerate(batch_messages):
            token_tensor = torch.tensor(tokens)
            # print(len(tokens))
            # print(len(tokens[0]))
            # print(token_tensor.shape)
            # Left pad!
            start_idx = max(sequence_length - len(token_tensor), 0)
            # print(token_tensor[:sequence_length].shape)
            # print(start_idx, batch_num)
            batch[start_idx:, batch_num] = token_tensor[:sequence_length]
        label_tensor = torch.tensor(labels[ii: ii+len(batch_messages)])
        meta_tensor = torch.tensor(meta[ii: ii+len(batch_messages)])

        yield batch, meta_tensor, label_tensor
```

```
# Test
text_batch, meta_batch, labels = next(iter(dataloader(X_train_text, X_train_meta, y_train)))
```

```
model = TextClassifier(len(vocab), 512, 128, 8, meta_size, 3)
hidden = model.init_hidden(16)
logps, hidden = model.forward(text_batch, meta_batch, hidden)
print(logps)
```

```
tensor([[-2.5526e+01, -3.1298e+01,  0.0000e+00],
        [-1.5616e-05, -2.3710e+01, -1.1069e+01],
        [-1.1590e+01, -2.3517e+01, -9.2983e-06],
        [-4.9131e+01, -4.5162e+01,  0.0000e+00],
        [-4.9298e+01, -4.4733e+01,  0.0000e+00],
        [ 0.0000e+00, -3.5084e+01, -3.5863e+01],
        [-2.5399e+01, -3.1202e+01,  0.0000e+00],
        [ 0.0000e+00, -2.5850e+01, -2.2903e+01],
        [ 0.0000e+00, -2.6333e+01, -2.4720e+01],
        [ 0.0000e+00, -3.0131e+01, -2.5198e+01],
        [-2.2539e-03, -2.0351e+01, -6.0963e+00],
        [-1.5892e+01, -2.8374e+01, -1.1921e-07],
        [ 0.0000e+00, -3.1515e+01, -2.7112e+01],
        [-3.8288e+01, -3.8450e+01,  0.0000e+00],
        [-5.6442e+01, -4.5757e+01,  0.0000e+00],
        [-3.2831e+01, -3.5207e+01,  0.0000e+00]], grad_fn=<LogSoftmaxBackward>)
```

## Configure Model

```
# Set model
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = TextClassifier(len(vocab)+1, 512, 128, 8, meta_size, 3, lstm_layers=2, dropout=0.2)
model.embedding.weight.data.uniform_(-1, 1)
model.to(device)
```

```
TextClassifier(
  (embedding): Embedding(26921, 512)
  (lstm): LSTM(512, 128, num_layers=2, dropout=0.2)
  (dropout): Dropout(p=0.2, inplace=False)
  (fc1): Linear(in_features=128, out_features=8, bias=True)
  (fc2): Linear(in_features=17, out_features=3, bias=True)
  (softmax): LogSoftmax(dim=1)
)
```

## Train Model

```
def train_model(model, epochs=3, batch_size=8, learning_rate=1e-4, sequence_length=200, clip=5, print_every=10):
    criterion = nn.NLLLoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)
    model.train()

    for epoch in range(epochs):
        print('Starting epoch {}'.format(epoch + 1))
        hidden = model.init_hidden(batch_size)
        steps = 0
        y_valid_epoch = []
        predicted_valid_epoch = []
```

```
for text_batch, meta_batch, labels in dataloader(
        X_train_text, X_train_meta, y_train, batch_size=batch_size, sequence_length=sequence_length, shuffle=False):
    steps += 1
    # Skip the last batch of which size is not equal to batch_size
    if text_batch.size(1) != batch_size:
        break

    # Creating new variables for the hidden state to avoid backprop entire training history
    hidden = tuple([each.data for each in hidden])

    # Set Device
    text_batch, meta_batch, labels = text_batch.to(device), meta_batch.to(device), labels.to(device)
    for each in hidden:
        each.to(device)

    # optimizer.zero_grad()
    model.zero_grad()

    # Get output and hidden state from the model
    output, hidden = model(text_batch, meta_batch, hidden)

    # Calculate the loss and perform backprop
    loss = criterion(output, labels)
    loss.backward()

    # Clip the gradient to prevent the exploading gradient problem in RNN/LSTM
    nn.utils.clip_grad_norm_(model.parameters(), clip)

    # Optimize
    optimizer.step()

    if steps % print_every == 0:
        model.eval()

        valid_losses = []
        accuracy = []
        predicted_valid = []
        y_valid_batch = []
        valid_hidden = model.init_hidden(batch_size)

        for text_batch, meta_batch, labels in dataloader(
                X_valid_text, X_valid_meta, y_valid, batch_size=batch_size, sequence_length=sequence_length, shuffle=False):

            # Skip the last batch of which size is not equal to batch_size
            if text_batch.size(1) != batch_size:
                break

            # Initialize within the loop to use label shape because batch_size did not work
            # valid_hidden = model.init_hidden(labels.shape[0])

            # Creating new variables for the hidden state
            valid_hidden = tuple([each.data for each in valid_hidden])
```

```
            # Set Device
            text_batch, meta_batch, labels = text_batch.to(device), meta_batch.to(device), labels.to(device)
            for each in valid_hidden:
                each.to(device)

            # Get output and hidden state from the model
            valid_output, valid_hidden = model(text_batch, meta_batch, valid_hidden)

            # Calculate the loss
            valid_loss = criterion(valid_output.squeeze(), labels)
            valid_losses.append(valid_loss.item())

            # Accuracy
            ps = torch.exp(valid_output)
            top_p, top_class = ps.topk(1, dim=1)
            equals = top_class == labels.view(*top_class.shape)
            accuracy.append(torch.mean(equals.type(torch.FloatTensor)).item())

            predicted_valid.extend(top_class.squeeze().cpu().numpy())
            y_valid_batch.extend(labels.view(*top_class.shape).squeeze().cpu().numpy())

        model.train()
        acc, f1 = metric(y_valid_batch, predicted_valid)
        predicted_valid_epoch.extend(predicted_valid)
        y_valid_epoch.extend(y_valid_batch)

        print("Epoch: {}/{}...".format(epoch+1, epochs),
              "Step: {}...".format(steps),
              "Loss: {:.6f}...".format(loss.item()),
              "Val Loss: {:.6f}".format(np.mean(valid_losses)),
              "Accuracy: {:.6f}".format(acc),
              "F1 Score: {:.6f}".format(f1))
print("{} steps in epoch {}".format(steps, epoch+1))
class_names = ['Lower', 'Hold', 'Raise']
y_valid_class = [class_names[int(idx)] for idx in y_valid_batch]
predicted_valid_class = [class_names[int(idx)] for idx in predicted_valid]
titles_options = [("Confusion matrix, without normalization", None), ("Confusion matrix, with normalization", 'true')]
for title, normalize in titles_options:
    disp = skplt.metrics.plot_confusion_matrix(y_valid_class, predicted_valid_class, normalize=normalize, title=title)
acc, f1 = metric(y_valid_class, predicted_valid_class)
print("\nEpoch: %d, Average Accuracy: %.8f, Average f1: %.8f\n" % (epoch+1, acc, f1))
plt.show()
plt.savefig(graph_dir + 'conf_mats_full_training.png')#bbox_inches='tight')
```

## IV. Glove Word Embed. + LSTM

*Use GloVe word embedding instead of Tfidf:*

### Input Data

```
# Use 6B 300d
```

```
# Use 6B 300d
glove_file = 'glove.6B.300d.pickle'
glove_path = glove_dir + glove_file

# Download Glove file if not exist
if not os.path.exists(glove_dir):
    if not os.path.exists(glove_dir):
        os.mkdir(glove_dir)
    !wget -o ${glove_path} http://nlp.stanford.edu/data/glove.6B.zip
    !unzip ${glove_path}glove*.zip

    embedding_dict = {}

    with open(glove_dir + "glove.6B.300d.txt", 'r') as f:
        for line in f:
            values = line.split()
            word = values[0]
            vectors = np.asarray(values[1:], 'float32')
            embedding_dict[word] = vectors
    f.close()

    pickle.dump(embedding_dict, open(glove_path, 'wb'))


glove_dict = pickle.load(open(glove_path, 'rb'))
print(len(glove_dict))
glove_dict['the']
```

```
            -2.6091e-01,  3.2434e-02,  5.6621e-02, -4.3296e-02, -2.1672e-02,
             2.2476e-01, -7.5129e-02, -6.7018e-02, -1.4247e-01,  3.8825e-02,
            -1.8951e-01,  2.9977e-01,  3.9305e-01,  1.7887e-01, -1.7343e-01,
            -2.1178e-01,  2.3617e-01, -6.3681e-02, -4.2318e-01, -1.1661e-01,
             9.3754e-02,  1.7296e-01, -3.3073e-01,  4.9112e-01, -6.8995e-01,
            -9.2462e-02,  2.4742e-01, -1.7991e-01,  9.7908e-02,  8.3118e-02,
             1.5299e-01, -2.7276e-01, -3.8934e-02,  5.4453e-01,  5.3737e-01,
             2.9105e-01, -7.3514e-03,  4.7880e-02, -4.0760e-01, -2.6759e-02,
             1.7919e-01,  1.0977e-02, -1.0963e-01, -2.6395e-01,  7.3990e-02,
             2.6236e-01, -1.5080e-01,  3.4623e-01,  2.5758e-01,  1.1971e-01,
            -3.7135e-02, -7.1593e-02,  4.3898e-01, -4.0764e-02,  1.6425e-02,
            -4.4640e-01,  1.7197e-01,  4.6246e-02,  5.8639e-02,  4.1499e-02,
             5.3948e-01,  5.2495e-01,  1.1361e-01, -4.8315e-02, -3.6385e-01,
             1.8704e-01,  9.2761e-02, -1.1129e-01, -4.2085e-01,  1.3992e-01,
            -3.9338e-01, -6.7945e-02,  1.2188e-01,  1.6707e-01,  7.5169e-02,
            -1.5529e-02, -1.9499e-01,  1.9638e-01,  5.3194e-02,  2.5170e-01,
            -3.4845e-01, -1.0638e-01, -3.4692e-01, -1.9024e-01, -2.0040e-01,
             1.2154e-01, -2.9208e-01,  2.3353e-02, -1.1618e-01, -3.5768e-01,
             6.2304e-02,  3.5884e-01,  2.9060e-02,  7.3005e-03,  4.9482e-03,
            -1.5048e-01, -1.2313e-01,  1.9337e-01,  1.2173e-01,  4.4503e-01,
             2.5147e-01,  1.0781e-01, -1.7716e-01,  3.8691e-02,  8.1530e-02,
             1.4667e-01,  6.3666e-02,  6.1332e-02, -7.5569e-02, -3.7724e-01,
             1.5850e-02, -3.0342e-01,  2.8374e-01, -4.2013e-02, -4.0715e-02,
            -1.5269e-01,  7.4980e-02,  1.5577e-01,  1.0433e-01,  3.1393e-01,
             1.9309e-01,  1.9429e-01,  1.5185e-01, -1.0192e-01, -1.8785e-02,
             2.0791e-01,  1.3366e-01,  1.9038e-01, -2.5558e-01,  3.0400e-01,
            -1.8960e-02,  2.0147e-01, -4.2110e-01, -7.5156e-03, -2.7977e-01,
            -1.9314e-01,  4.6204e-02,  1.9971e-01, -3.0207e-01,  2.5735e-01,
```

```
        6.8107e-01, -1.9409e-01,  2.3984e-01,  2.2493e-01,  6.5224e-01,
       -1.3561e-01, -1.7383e-01, -4.8209e-02, -1.1860e-01,  2.1588e-03,
       -1.9525e-02,  1.1948e-01,  1.9346e-01, -4.0820e-01, -8.2966e-02,
        1.6626e-01, -1.0601e-01,  3.5861e-01,  1.6922e-01,  7.2590e-02,
       -2.4803e-01, -1.0024e-01, -5.2491e-01, -1.7745e-01, -3.6647e-01,
        2.6180e-01, -1.2077e-02,  8.3190e-02, -2.1528e-01,  4.1045e-01,
        2.9136e-01,  3.0869e-01,  7.8864e-02,  3.2207e-01, -4.1023e-02,
       -1.0970e-01, -9.2041e-02, -1.2339e-01, -1.6416e-01,  3.5382e-01,
       -8.2774e-02,  3.3171e-01, -2.4738e-01, -4.8928e-02,  1.5746e-01,
        1.8988e-01, -2.6642e-02,  6.3315e-02, -1.0673e-02,  3.4089e-01,
        1.4106e+00,  1.3417e-01,  2.8191e-01, -2.5940e-01,  5.5267e-02,
       -5.2425e-02, -2.5789e-01,  1.9127e-02, -2.2084e-02,  3.2113e-01,
        6.8818e-02,  5.1207e-01,  1.6478e-01, -2.0194e-01,  2.9232e-01,
        9.8575e-02,  1.3145e-02, -1.0652e-01,  1.3510e-01, -4.5332e-02,
        2.0697e-01, -4.8425e-01, -4.4706e-01,  3.3305e-03,  2.9264e-03,
       -1.0975e-01, -2.3325e-01,  2.2442e-01, -1.0503e-01,  1.2339e-01,
        1.0978e-01,  4.8994e-02, -2.5157e-01,  4.0319e-01,  3.5318e-01,
        1.8651e-01, -2.3622e-02, -1.2734e-01,  1.1475e-01,  2.7359e-01,
       -2.1866e-01,  1.5794e-02,  8.1754e-01, -2.3792e-02, -8.5469e-01,
       -1.6203e-01,  1.8076e-01,  2.8014e-02, -1.4340e-01,  1.3139e-03,
       -9.1735e-02, -8.9704e-02,  1.1105e-01, -1.6703e-01,  6.8377e-02,
       -8.7388e-02, -3.9789e-02,  1.4184e-02,  2.1187e-01,  2.8579e-01,
       -2.8797e-01, -5.8996e-02, -3.2436e-02, -4.7009e-03, -1.7052e-01,
       -3.4741e-02, -1.1489e-01,  7.5093e-02,  9.9526e-02,  4.8183e-02,
       -7.3775e-02, -4.1817e-01,  4.1268e-03,  4.4414e-01, -1.6062e-01,
        1.4294e-01, -2.2628e+00, -2.7347e-02,  8.1311e-01,  7.7417e-01,
       -2.5639e-01, -1.1576e-01, -1.1982e-01, -2.1363e-01,  2.8429e-02,
        2.7261e-01,  3.1026e-02,  9.6782e-02,  6.7769e-03,  1.4082e-01,
       -1.3064e-02, -2.9686e-01, -7.9913e-02,  1.9500e-01,  3.1549e-02,
        2.8506e-01, -8.7461e-02,  9.0611e-03, -2.0989e-01,  5.3913e-02],
      dtype=float32)
```

```python
weight_matrix = np.zeros((len(vocab), 300))
words_found = 0

for i, word in enumerate(vocab):
    try:
        weight_matrix[i] = glove_dict[word]
        words_found += 1
    except KeyError:
        weight_matrix[i] = np.random.normal(scale=0.6, size=(300,))

print('{} words found out of {} words in vocab.'.format(words_found, len(vocab)))
print(weight_matrix.shape)
```

```
11766 words found out of 26920 words in vocab.
(26920, 300)
```

```python
type(weight_matrix)
```

```
numpy.ndarray
```

## Model

```python
class GloveTextClassifier(nn.Module):
    def __init__(self, weight_matrix, lstm_size, dense_size, meta_size, output_size, lstm_layers=1, dropout=0.1):
        super().__init__()
        vocab_size, embed_size = weight_matrix.shape
        self.lstm_size = lstm_size
        self.output_size = output_size
        self.lstm_layers = lstm_layers
        self.dropout = dropout

        self.embedding = nn.Embedding(vocab_size, embed_size)
        self.embedding.load_state_dict({'weight': torch.tensor(weight_matrix)})
        self.embedding.weight.requires_grad = False
        self.lstm = nn.LSTM(embed_size, lstm_size, lstm_layers, dropout=dropout, batch_first=False)
        self.dropout = nn.Dropout(0.2)
        self.fc1 = nn.Linear(lstm_size, dense_size)
        self.fc2 = nn.Linear(dense_size + meta_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def init_hidden(self, batch_size):
        weight = next(self.parameters()).data
        hidden = (weight.new(self.lstm_layers, batch_size, self.lstm_size).zero_(),
                  weight.new(self.lstm_layers, batch_size, self.lstm_size).zero_())

        return hidden

    def forward(self, nn_input_text, nn_input_meta, hidden_state):
        batch_size = nn_input_text.size(0)
        nn_input_text = nn_input_text.long()
        embeds = self.embedding(nn_input_text)
        lstm_out, hidden_state = self.lstm(embeds, hidden_state)
        # Stack up LSTM outputs, apply dropout
        lstm_out = lstm_out[-1,:,:]
        lstm_out = self.dropout(lstm_out)
        # Dense layer
        dense_out = self.fc1(lstm_out)
        # Concatinate the dense output and meta inputs
        concat_layer = torch.cat((dense_out, nn_input_meta.float()), 1)
        out = self.fc2(concat_layer)
        logps = self.softmax(out)

        return logps, hidden_state
```

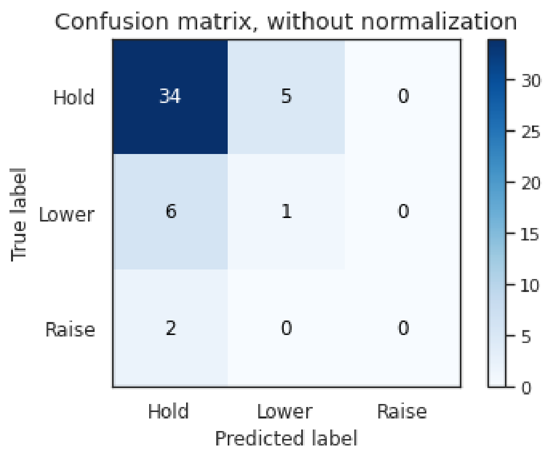
## Configure Model

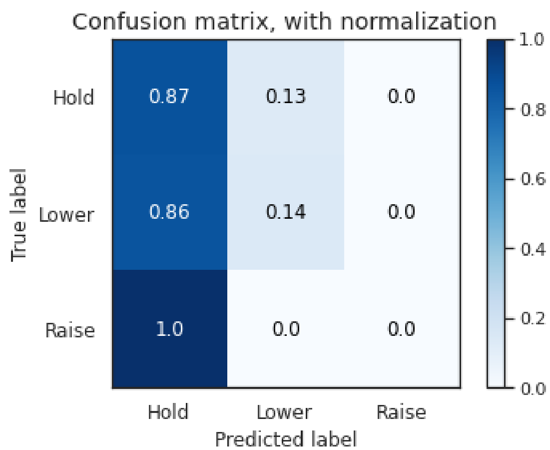
```python
# Set model
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = GloveTextClassifier(weight_matrix, 128, 8, meta_size, 3, lstm_layers=2, dropout=0.2)
model.to(device)
```

```
GloveTextClassifier(
  (embedding): Embedding(26920, 300)
  (lstm): LSTM(300, 128, num_layers=2, dropout=0.2)
  (dropout): Dropout(p=0.2, inplace=False)
  (fc1): Linear(in_features=128, out_features=8, bias=True)
  (fc2): Linear(in_features=17, out_features=3, bias=True)
  (softmax): LogSoftmax(dim=1)
)
```

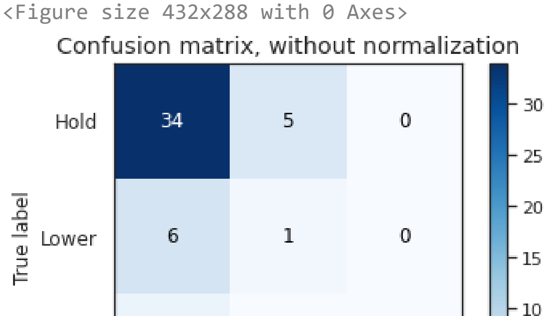
```
train_model(model)
```

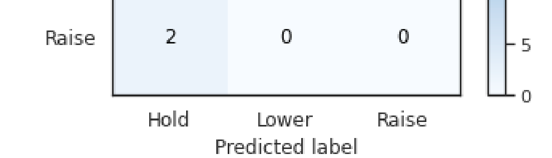
```
Starting epoch 1
Epoch: 1/3... Step: 10... Loss: 9.112595... Val Loss: 2.936284 Accuracy: 0.729167 F1 Score: 0.331117
Epoch: 1/3... Step: 20... Loss: 14.855077... Val Loss: 2.885410 Accuracy: 0.729167 F1 Score: 0.331117
24 steps in epoch 1

Epoch: 1, Average Accuracy: 0.72916667, Average f1: 0.33111744
```

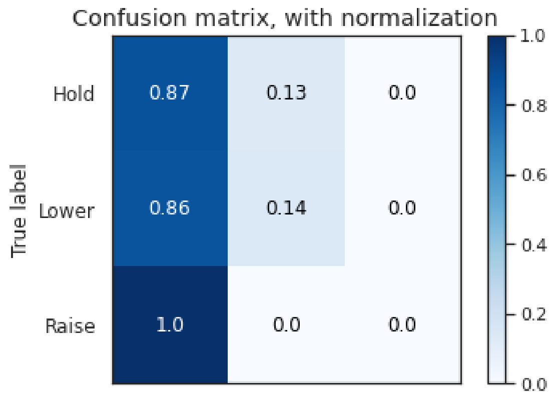



```
Starting epoch 2
Epoch: 2/3... Step: 10... Loss: 8.817862... Val Loss: 2.819411 Accuracy: 0.729167 F1 Score: 0.331117
Epoch: 2/3... Step: 20... Loss: 14.346961... Val Loss: 2.767932 Accuracy: 0.729167 F1 Score: 0.331117
24 steps in epoch 2

Epoch: 2, Average Accuracy: 0.72916667, Average f1: 0.33111744

<Figure size 432x288 with 0 Axes>
```






The result does not look good. In fact, only the first hunderds of text can be used. Now, consider to split the text to the length of 200 with overlapping 50 words again.

```
    Epoch: 3/3     Step: 20     Loss: 13.841166     Val Loss: 2.643033 Accuracy: 0.729167 F1 Score: 0.331117
```

```
train_df.columns
```

```
    Index(['target', 'prev_decision', 'GDP_diff_prev', 'PMI_value',
           'Employ_diff_prev', 'Rsales_diff_year', 'Unemp_diff_prev',
           'Inertia_diff', 'Hsales_diff_year', 'Balanced_diff', 'statement',
           'minutes', 'presconf_script', 'speech', 'testimony', 'text', 'tone',
           'tokenized', 'token_ids', 'tokenized_text', 'tfidf_Negative',
           'tfidf_Positive', 'tfidf_Uncertainty', 'tfidf_Litigious',
           'tfidf_StrongModal', 'tfidf_Constraining', 'cos_sim_Negative',
           'cos_sim_Positive', 'cos_sim_Uncertainty', 'cos_sim_Litigious',
           'cos_sim_StrongModal', 'cos_sim_Constraining'],
          dtype='object')
```

```
split_train_df = train_df.drop(columns=['statement',
       'minutes', 'speech', 'testimony',
       'tokenized', 'token_ids', 'tokenized_text', 'tfidf_Negative',
       'tfidf_Positive', 'tfidf_Uncertainty', 'tfidf_Litigious',
       'tfidf_StrongModal', 'tfidf_Constraining', 'cos_sim_Negative',
       'cos_sim_Positive', 'cos_sim_Uncertainty', 'cos_sim_Litigious',
       'cos_sim_StrongModal', 'cos_sim_Constraining'])
```

```
split_train_df.shape
```

```
    (237, 13)
```

```
# Split functions to process long text in machine learning based NLP
```

```
def get_split(text, split_len=200, overlap=50):
    '''
    Returns a list of split text of $split_len with overlapping of $overlap.
    Each item of the list will have around split_len length of text.
    '''
    l_total = []
    words = re.findall(r'\b([a-zA-Z]+n\'t|[a-zA-Z]+\'s|[a-zA-Z]+)\b', text)

    if len(words) < split_len:
        n = 1
    else:
        n = (len(words) - overlap) // (split_len - overlap) + 1

    for i in range(n):
        l_parcial = words[(split_len - overlap) * i: (split_len - overlap) * i + split_len]
        l_total.append(" ".join(l_parcial))
    return l_total

def get_split_df(df, split_len=200, overlap=50):
    '''
    Returns a dataframe which is an extension of an input dataframe.
    Each row in the new dataframe has less than $split_len words in 'text'.
    '''
    split_data_list = []

    for i, row in tqdm(df.iterrows(), total=df.shape[0]):
        #print("Original Word Count: ", row['word_count'])
        text_list = get_split(row["text"], split_len, overlap)
        for text in text_list:
            row['text'] = text
            #print(len(re.findall(r'\b([a-zA-Z]+n\'t|[a-zA-Z]+\'s|[a-zA-Z]+)\b', text)))
            #row['word_count'] = len(re.findall(r'\b([a-zA-Z]+n\'t|[a-zA-Z]+\'s|[a-zA-Z]+)\b', text))
            split_data_list.append(list(row))

    split_df = pd.DataFrame(split_data_list, columns=df.columns)

    return split_df



split_train_df = get_split_df(split_train_df)
split_train_df.shape
```

100% 237/237 [00:01<00:00, 131.79it/s]

```
(19974, 13)
```

```
tokenized = tokenize_df(split_train_df)
lemma_docs = [" ".join(words) for words in tokenized]
all_words = [word for text in tokenized for word in text]
counts = Counter(all_words)
```

```
bow = sorted(counts, key=counts.get, reverse=True)
vocab = {word: ii for ii, word in enumerate(counts, 1)}
id2vocab = {v: k for k, v in vocab.items()}
token_ids = [[vocab[word] for word in text_words] for text_words in tokenized]

# Add to the dataframe
split_train_df['token_ids'] = token_ids
```

100% 19974/19974 [00:39<00:00, 500.64it/s]

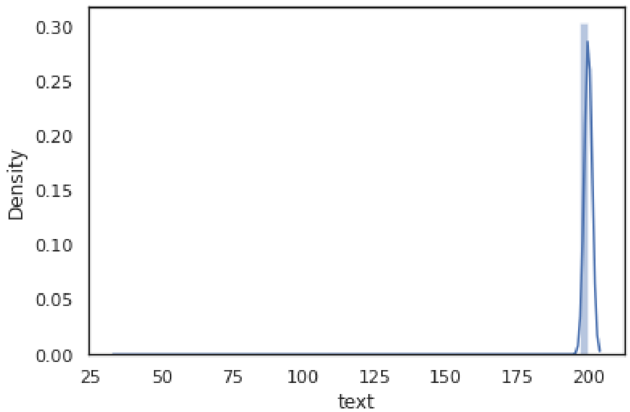
```
weight_matrix = np.zeros((len(vocab)+1, 300))
words_found = 0

for i, word in enumerate(vocab):
    try:
        weight_matrix[i] = glove_dict[word]
        words_found += 1
    except KeyError:
        weight_matrix[i] = np.random.normal(scale=0.6, size=(300,))
```

```
split_train_df.head()
```

| | target | prev_decision | GDP_diff_prev | PMI_value | Employ_diff_prev | Rsales_diff_year | Unemp_diff_prev | Inertia_diff | Hsales_diff_year | Balanced_diff | presconf_sc |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 1 | 1.043165 | 55.8 | 261.0 | 1.807631 | 0.0 | -0.015902 | 14.901418 | 0.035879 | |

```
sns.distplot(split_train_df['text'].apply(lambda x: len(x.split())))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8008475b38>
```




```
# X and Y data used
y_data = split_train_df['target']
X_data = split_train_df[nontext_columns + ['tone', 'token_ids']]

# Train test split (Shuffle=False will make the test data for the most recent ones)
X_train, X_valid, y_train, y_valid = \
model_selection.train_test_split(X_data.values, y_data.values, test_size=0.2, shuffle=True)

X_train_meta = get_numeric_data(X_train)
X_train_text = get_text_data(X_train)
X_valid_meta = get_numeric_data(X_valid)
X_valid_text = get_text_data(X_valid)

print('Shape of train meta', len(X_train_meta))
print('Shape of train text', len(X_train_text))
print("Shape of valid meta ", len(X_valid_meta))
print("Shape of valid text ", len(X_valid_text))

meta_size = len(X_train_meta[0])
print("Meta data size: ", meta_size)
```

```
Shape of train meta 15979
Shape of train text 15979
Shape of valid meta  3995
Shape of valid text  3995
Meta data size:  9
```

```
len(weight_matrix)
```

```
27716
```

```
# Set model
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = GloveTextClassifier(weight_matrix, 128, 8, meta_size, 3, lstm_layers=2, dropout=0.2)
model.to(device)
```

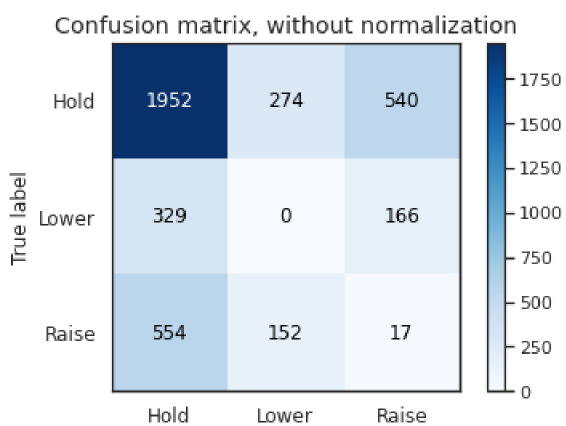
```
GloveTextClassifier(
  (embedding): Embedding(27716, 300)
  (lstm): LSTM(300, 128, num_layers=2, dropout=0.2)
  (dropout): Dropout(p=0.2, inplace=False)
  (fc1): Linear(in_features=128, out_features=8, bias=True)
  (fc2): Linear(in_features=17, out_features=3, bias=True)
  (softmax): LogSoftmax(dim=1)
)
```

## Train Model

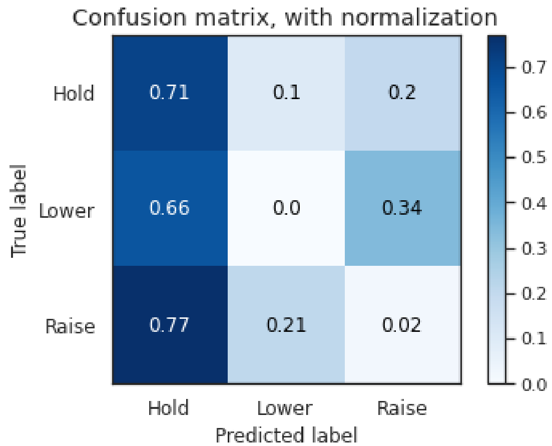
```
# Train the model (TODO long waiting times; gpu mounting issues, assign processes to gpu threads?):
train_model(model, epochs=3, batch_size=16, learning_rate=1e-4, sequence_length=200, clip=5, print_every=10)
```

```
Starting epoch 1
Epoch: 1/3... Step: 10... Loss: 18.428333... Val Loss: 28.339404 Accuracy: 0.075301 F1 Score: 0.055640
Epoch: 1/3... Step: 20... Loss: 25.012129... Val Loss: 27.903581 Accuracy: 0.075301 F1 Score: 0.055640
Epoch: 1/3... Step: 30... Loss: 20.320839... Val Loss: 27.465781 Accuracy: 0.080572 F1 Score: 0.060863
Epoch: 1/3... Step: 40... Loss: 20.154690... Val Loss: 27.014359 Accuracy: 0.080572 F1 Score: 0.060863
Epoch: 1/3... Step: 50... Loss: 23.493519... Val Loss: 26.516652 Accuracy: 0.083082 F1 Score: 0.063785
Epoch: 1/3... Step: 60... Loss: 17.672779... Val Loss: 25.871492 Accuracy: 0.082078 F1 Score: 0.064063
Epoch: 1/3... Step: 70... Loss: 13.999308... Val Loss: 24.852895 Accuracy: 0.078815 F1 Score: 0.062116
Epoch: 1/3... Step: 80... Loss: 28.473644... Val Loss: 23.739015 Accuracy: 0.131024 F1 Score: 0.108177
Epoch: 1/3... Step: 90... Loss: 25.520273... Val Loss: 22.849410 Accuracy: 0.132279 F1 Score: 0.108783
Epoch: 1/3... Step: 100... Loss: 20.978685... Val Loss: 22.076225 Accuracy: 0.150853 F1 Score: 0.119250
Epoch: 1/3... Step: 110... Loss: 20.252333... Val Loss: 21.396975 Accuracy: 0.188253 F1 Score: 0.142276
Epoch: 1/3... Step: 120... Loss: 34.064781... Val Loss: 20.821464 Accuracy: 0.209337 F1 Score: 0.150352
Epoch: 1/3... Step: 130... Loss: 38.053909... Val Loss: 20.359433 Accuracy: 0.255271 F1 Score: 0.173608
Epoch: 1/3... Step: 140... Loss: 5.569947... Val Loss: 19.946742 Accuracy: 0.270331 F1 Score: 0.176873
Epoch: 1/3... Step: 150... Loss: 16.247223... Val Loss: 19.582000 Accuracy: 0.295934 F1 Score: 0.185785
Epoch: 1/3... Step: 160... Loss: 22.159605... Val Loss: 19.268604 Accuracy: 0.323795 F1 Score: 0.192406
Epoch: 1/3... Step: 170... Loss: 42.622784... Val Loss: 19.002430 Accuracy: 0.369729 F1 Score: 0.208441
Epoch: 1/3... Step: 180... Loss: 24.883438... Val Loss: 18.765462 Accuracy: 0.395582 F1 Score: 0.217400
Epoch: 1/3... Step: 190... Loss: 19.679958... Val Loss: 18.545589 Accuracy: 0.417922 F1 Score: 0.220227
Epoch: 1/3... Step: 200... Loss: 14.096807... Val Loss: 18.336613 Accuracy: 0.440512 F1 Score: 0.227937
Epoch: 1/3... Step: 210... Loss: 25.639980... Val Loss: 18.144224 Accuracy: 0.449548 F1 Score: 0.223219
Epoch: 1/3... Step: 220... Loss: 8.383085... Val Loss: 17.961726 Accuracy: 0.452811 F1 Score: 0.223835
Epoch: 1/3... Step: 230... Loss: 31.213572... Val Loss: 17.774215 Accuracy: 0.457580 F1 Score: 0.225396
Epoch: 1/3... Step: 240... Loss: 15.263031... Val Loss: 17.590240 Accuracy: 0.457831 F1 Score: 0.225477
Epoch: 1/3... Step: 250... Loss: 21.990002... Val Loss: 17.400303 Accuracy: 0.457831 F1 Score: 0.225477
Epoch: 1/3... Step: 260... Loss: 17.953205... Val Loss: 17.216941 Accuracy: 0.457831 F1 Score: 0.225477
Epoch: 1/3... Step: 270... Loss: 12.837781... Val Loss: 17.047120 Accuracy: 0.457831 F1 Score: 0.225477
Epoch: 1/3... Step: 280... Loss: 22.192322... Val Loss: 16.881308 Accuracy: 0.464608 F1 Score: 0.225182
Epoch: 1/3... Step: 290... Loss: 23.225422... Val Loss: 16.701313 Accuracy: 0.475653 F1 Score: 0.227915
Epoch: 1/3... Step: 300... Loss: 17.581619... Val Loss: 16.531313 Accuracy: 0.482430 F1 Score: 0.230042
Epoch: 1/3... Step: 310... Loss: 17.409735... Val Loss: 16.354581 Accuracy: 0.485693 F1 Score: 0.231059
Epoch: 1/3... Step: 320... Loss: 13.158725... Val Loss: 16.170684 Accuracy: 0.485693 F1 Score: 0.231059
Epoch: 1/3... Step: 330... Loss: 11.570838... Val Loss: 15.986672 Accuracy: 0.479167 F1 Score: 0.229020
Epoch: 1/3... Step: 340... Loss: 9.901147... Val Loss: 15.801609 Accuracy: 0.475653 F1 Score: 0.227915
Epoch: 1/3... Step: 350... Loss: 9.603326... Val Loss: 15.616491 Accuracy: 0.462349 F1 Score: 0.224456
Epoch: 1/3... Step: 360... Loss: 16.448822... Val Loss: 15.436438 Accuracy: 0.462349 F1 Score: 0.225238
Epoch: 1/3... Step: 370... Loss: 21.823013... Val Loss: 15.255454 Accuracy: 0.457831 F1 Score: 0.223776
Epoch: 1/3... Step: 380... Loss: 11.619324... Val Loss: 15.084956 Accuracy: 0.462349 F1 Score: 0.225238
Epoch: 1/3... Step: 390... Loss: 30.097195... Val Loss: 14.914146 Accuracy: 0.462349 F1 Score: 0.225238
Epoch: 1/3... Step: 400... Loss: 17.687571... Val Loss: 14.738082 Accuracy: 0.462349 F1 Score: 0.225238
Epoch: 1/3... Step: 410... Loss: 12.214677... Val Loss: 14.565123 Accuracy: 0.462349 F1 Score: 0.225238
Epoch: 1/3... Step: 420... Loss: 16.074863... Val Loss: 14.388211 Accuracy: 0.462349 F1 Score: 0.225238
Epoch: 1/3... Step: 430... Loss: 5.280365... Val Loss: 14.214615 Accuracy: 0.457831 F1 Score: 0.224437
Epoch: 1/3... Step: 440... Loss: 16.545734... Val Loss: 14.046828 Accuracy: 0.457831 F1 Score: 0.224437
Epoch: 1/3... Step: 450... Loss: 2.844280... Val Loss: 13.875641 Accuracy: 0.457831 F1 Score: 0.224437
Epoch: 1/3... Step: 460... Loss: 24.816601... Val Loss: 13.705091 Accuracy: 0.457831 F1 Score: 0.224437
Epoch: 1/3... Step: 470... Loss: 13.723431... Val Loss: 13.535408 Accuracy: 0.457831 F1 Score: 0.224230
Epoch: 1/3... Step: 480... Loss: 6.789166... Val Loss: 13.358692 Accuracy: 0.457831 F1 Score: 0.224437
Epoch: 1/3... Step: 490... Loss: 19.655033... Val Loss: 13.188511 Accuracy: 0.462349 F1 Score: 0.225238
Epoch: 1/3... Step: 500... Loss: 20.062843... Val Loss: 13.017516 Accuracy: 0.457831 F1 Score: 0.224437
Epoch: 1/3... Step: 510... Loss: 15.146551... Val Loss: 12.849956 Accuracy: 0.457831 F1 Score: 0.224437
Epoch: 1/3... Step: 520... Loss: 20.167570... Val Loss: 12.683941 Accuracy: 0.473394 F1 Score: 0.228772
Epoch: 1/3... Step: 530... Loss: 25.879658... Val Loss: 12.510490 Accuracy: 0.476406 F1 Score: 0.229726
Epoch: 1/3... Step: 540... Loss: 12.845112... Val Loss: 12.337474 Accuracy: 0.473645 F1 Score: 0.228852
Epoch: 1/3... Step: 550... Loss: 13.419373... Val Loss: 12.173291 Accuracy: 0.457831 F1 Score: 0.224437
Epoch: 1/3... Step: 560... Loss: 15.631526... Val Loss: 12.011836 Accuracy: 0.457831 F1 Score: 0.224437
Epoch: 1/3... Step: 570... Loss: 11.588209... Val Loss: 11.842390 Accuracy: 0.457831 F1 Score: 0.224437
Epoch: 1/3... Step: 580... Loss: 19.460735... Val Loss: 11.669813 Accuracy: 0.457831 F1 Score: 0.224437
Epoch: 1/3... Step: 590... Loss: 3.735446... Val Loss: 11.498912 Accuracy: 0.460843 F1 Score: 0.225414
```

```
Epoch: 1/3... Step: 600... Loss: 10.774132... Val Loss: 11.332746 Accuracy: 0.460843 F1 Score: 0.225414
Epoch: 1/3... Step: 610... Loss: 8.981198... Val Loss: 11.172849 Accuracy: 0.465110 F1 Score: 0.226792
Epoch: 1/3... Step: 620... Loss: 13.045036... Val Loss: 11.013392 Accuracy: 0.475151 F1 Score: 0.229998
Epoch: 1/3... Step: 630... Loss: 7.886061... Val Loss: 10.848494 Accuracy: 0.479920 F1 Score: 0.231253
Epoch: 1/3... Step: 640... Loss: 7.747796... Val Loss: 10.681129 Accuracy: 0.466616 F1 Score: 0.227026
Epoch: 1/3... Step: 650... Loss: 14.897986... Val Loss: 10.514312 Accuracy: 0.464608 F1 Score: 0.226380
Epoch: 1/3... Step: 660... Loss: 6.418825... Val Loss: 10.350416 Accuracy: 0.464608 F1 Score: 0.226380
Epoch: 1/3... Step: 670... Loss: 13.299671... Val Loss: 10.194853 Accuracy: 0.483434 F1 Score: 0.231685
Epoch: 1/3... Step: 680... Loss: 14.172043... Val Loss: 10.035909 Accuracy: 0.482932 F1 Score: 0.230199
Epoch: 1/3... Step: 690... Loss: 11.557758... Val Loss: 9.877290 Accuracy: 0.488705 F1 Score: 0.231744
Epoch: 1/3... Step: 700... Loss: 7.132579... Val Loss: 9.711685 Accuracy: 0.482932 F1 Score: 0.230737
Epoch: 1/3... Step: 710... Loss: 10.983963... Val Loss: 9.540447 Accuracy: 0.480673 F1 Score: 0.230820
Epoch: 1/3... Step: 720... Loss: 11.199901... Val Loss: 9.371531 Accuracy: 0.480673 F1 Score: 0.230820
Epoch: 1/3... Step: 730... Loss: 12.325286... Val Loss: 9.204194 Accuracy: 0.480673 F1 Score: 0.230820
Epoch: 1/3... Step: 740... Loss: 8.232181... Val Loss: 9.037120 Accuracy: 0.461847 F1 Score: 0.225490
Epoch: 1/3... Step: 750... Loss: 12.792260... Val Loss: 8.870055 Accuracy: 0.461847 F1 Score: 0.225490
Epoch: 1/3... Step: 760... Loss: 7.517048... Val Loss: 8.700635 Accuracy: 0.461847 F1 Score: 0.225490
Epoch: 1/3... Step: 770... Loss: 14.573689... Val Loss: 8.534048 Accuracy: 0.461847 F1 Score: 0.225035
Epoch: 1/3... Step: 780... Loss: 3.787906... Val Loss: 8.371194 Accuracy: 0.461847 F1 Score: 0.225035
Epoch: 1/3... Step: 790... Loss: 2.109604... Val Loss: 8.209075 Accuracy: 0.480673 F1 Score: 0.230362
Epoch: 1/3... Step: 800... Loss: 9.483181... Val Loss: 8.044889 Accuracy: 0.480673 F1 Score: 0.230362
Epoch: 1/3... Step: 810... Loss: 9.501197... Val Loss: 7.879369 Accuracy: 0.480673 F1 Score: 0.229575
Epoch: 1/3... Step: 820... Loss: 5.491369... Val Loss: 7.718685 Accuracy: 0.493725 F1 Score: 0.232795
Epoch: 1/3... Step: 830... Loss: 11.238077... Val Loss: 7.561200 Accuracy: 0.500502 F1 Score: 0.234079
Epoch: 1/3... Step: 840... Loss: 21.226372... Val Loss: 7.400829 Accuracy: 0.504016 F1 Score: 0.235143
Epoch: 1/3... Step: 850... Loss: 3.381847... Val Loss: 7.232000 Accuracy: 0.504016 F1 Score: 0.235143
Epoch: 1/3... Step: 860... Loss: 7.755480... Val Loss: 7.060799 Accuracy: 0.493725 F1 Score: 0.232795
Epoch: 1/3... Step: 870... Loss: 9.529484... Val Loss: 6.895130 Accuracy: 0.493725 F1 Score: 0.232795
Epoch: 1/3... Step: 880... Loss: 6.186252... Val Loss: 6.727483 Accuracy: 0.480673 F1 Score: 0.229534
Epoch: 1/3... Step: 890... Loss: 6.564061... Val Loss: 6.567119 Accuracy: 0.475402 F1 Score: 0.228537
Epoch: 1/3... Step: 900... Loss: 11.590181... Val Loss: 6.408165 Accuracy: 0.475402 F1 Score: 0.228537
Epoch: 1/3... Step: 910... Loss: 1.714688... Val Loss: 6.246611 Accuracy: 0.486195 F1 Score: 0.231256
Epoch: 1/3... Step: 920... Loss: 3.271265... Val Loss: 6.085230 Accuracy: 0.485442 F1 Score: 0.231022
Epoch: 1/3... Step: 930... Loss: 7.857314... Val Loss: 5.923467 Accuracy: 0.475402 F1 Score: 0.228537
Epoch: 1/3... Step: 940... Loss: 3.747850... Val Loss: 5.760836 Accuracy: 0.456576 F1 Score: 0.222508
Epoch: 1/3... Step: 950... Loss: 6.777191... Val Loss: 5.597384 Accuracy: 0.459588 F1 Score: 0.223483
Epoch: 1/3... Step: 960... Loss: 3.909441... Val Loss: 5.434447 Accuracy: 0.486195 F1 Score: 0.231256
Epoch: 1/3... Step: 970... Loss: 7.416017... Val Loss: 5.274814 Accuracy: 0.488454 F1 Score: 0.236425
Epoch: 1/3... Step: 980... Loss: 7.994610... Val Loss: 5.116904 Accuracy: 0.488454 F1 Score: 0.234972
Epoch: 1/3... Step: 990... Loss: 4.246825... Val Loss: 4.960866 Accuracy: 0.494227 F1 Score: 0.240177
999 steps in epoch 1

Epoch: 1, Average Accuracy: 0.49422691, Average f1: 0.24017718
```



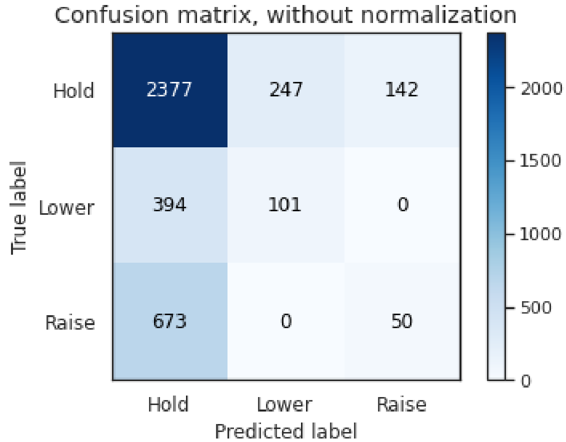
Predicted label

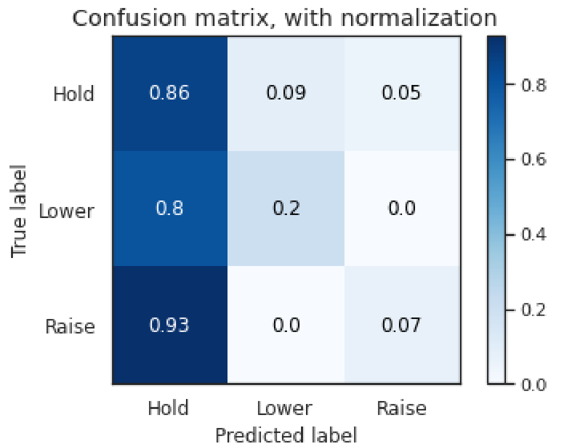



```
Starting epoch 2
Epoch: 2/3... Step: 10... Loss: 1.206359... Val Loss: 4.678074 Accuracy: 0.485944 F1 Score: 0.243745
Epoch: 2/3... Step: 20... Loss: 3.008859... Val Loss: 4.527054 Accuracy: 0.485944 F1 Score: 0.242956
Epoch: 2/3... Step: 30... Loss: 1.954596... Val Loss: 4.379509 Accuracy: 0.501757 F1 Score: 0.247202
Epoch: 2/3... Step: 40... Loss: 2.630062... Val Loss: 4.236898 Accuracy: 0.510291 F1 Score: 0.250742
Epoch: 2/3... Step: 50... Loss: 2.434989... Val Loss: 4.097671 Accuracy: 0.505271 F1 Score: 0.249025
Epoch: 2/3... Step: 60... Loss: 7.451366... Val Loss: 3.958860 Accuracy: 0.507530 F1 Score: 0.250626
Epoch: 2/3... Step: 70... Loss: 3.632784... Val Loss: 3.827883 Accuracy: 0.504769 F1 Score: 0.249904
Epoch: 2/3... Step: 80... Loss: 2.657770... Val Loss: 3.710281 Accuracy: 0.507028 F1 Score: 0.250681
Epoch: 2/3... Step: 90... Loss: 2.244355... Val Loss: 3.596107 Accuracy: 0.497741 F1 Score: 0.246954
Epoch: 2/3... Step: 100... Loss: 1.155500... Val Loss: 3.482940 Accuracy: 0.503514 F1 Score: 0.248398
Epoch: 2/3... Step: 110... Loss: 3.966704... Val Loss: 3.370847 Accuracy: 0.514809 F1 Score: 0.251228
Epoch: 2/3... Step: 120... Loss: 6.025296... Val Loss: 3.274631 Accuracy: 0.509538 F1 Score: 0.249029
Epoch: 2/3... Step: 130... Loss: 2.225093... Val Loss: 3.184395 Accuracy: 0.515060 F1 Score: 0.248722
Epoch: 2/3... Step: 140... Loss: 0.558510... Val Loss: 3.101327 Accuracy: 0.515060 F1 Score: 0.248663
Epoch: 2/3... Step: 150... Loss: 0.862765... Val Loss: 3.032582 Accuracy: 0.521335 F1 Score: 0.249442
Epoch: 2/3... Step: 160... Loss: 1.873758... Val Loss: 2.967671 Accuracy: 0.523343 F1 Score: 0.249996
Epoch: 2/3... Step: 170... Loss: 2.479295... Val Loss: 2.912967 Accuracy: 0.538404 F1 Score: 0.254590
Epoch: 2/3... Step: 180... Loss: 3.950841... Val Loss: 2.865900 Accuracy: 0.577309 F1 Score: 0.265986
Epoch: 2/3... Step: 190... Loss: 4.247682... Val Loss: 2.813878 Accuracy: 0.592620 F1 Score: 0.270877
Epoch: 2/3... Step: 200... Loss: 2.082726... Val Loss: 2.757204 Accuracy: 0.592620 F1 Score: 0.270877
Epoch: 2/3... Step: 210... Loss: 2.998930... Val Loss: 2.695205 Accuracy: 0.575803 F1 Score: 0.265245
Epoch: 2/3... Step: 220... Loss: 1.281903... Val Loss: 2.627056 Accuracy: 0.564759 F1 Score: 0.261556
Epoch: 2/3... Step: 230... Loss: 2.180455... Val Loss: 2.577133 Accuracy: 0.542169 F1 Score: 0.262148
Epoch: 2/3... Step: 240... Loss: 2.913838... Val Loss: 2.525546 Accuracy: 0.538655 F1 Score: 0.271184
Epoch: 2/3... Step: 250... Loss: 1.933268... Val Loss: 2.469819 Accuracy: 0.538906 F1 Score: 0.278017
Epoch: 2/3... Step: 260... Loss: 4.042858... Val Loss: 2.420744 Accuracy: 0.538906 F1 Score: 0.277991
Epoch: 2/3... Step: 270... Loss: 2.285530... Val Loss: 2.364162 Accuracy: 0.552962 F1 Score: 0.263825
Epoch: 2/3... Step: 280... Loss: 4.467858... Val Loss: 2.326975 Accuracy: 0.610191 F1 Score: 0.275082
Epoch: 2/3... Step: 290... Loss: 3.231866... Val Loss: 2.285587 Accuracy: 0.617972 F1 Score: 0.277061
Epoch: 2/3... Step: 300... Loss: 3.860025... Val Loss: 2.242661 Accuracy: 0.617972 F1 Score: 0.277110
Epoch: 2/3... Step: 310... Loss: 2.542134... Val Loss: 2.187827 Accuracy: 0.598143 F1 Score: 0.284932
Epoch: 2/3... Step: 320... Loss: 1.106793... Val Loss: 2.139409 Accuracy: 0.593373 F1 Score: 0.289122
Epoch: 2/3... Step: 330... Loss: 0.733704... Val Loss: 2.102462 Accuracy: 0.600402 F1 Score: 0.295275
Epoch: 2/3... Step: 340... Loss: 1.279887... Val Loss: 2.055841 Accuracy: 0.586345 F1 Score: 0.290975
Epoch: 2/3... Step: 350... Loss: 1.119753... Val Loss: 2.001904 Accuracy: 0.574297 F1 Score: 0.280929
Epoch: 2/3... Step: 360... Loss: 1.204106... Val Loss: 1.948284 Accuracy: 0.568273 F1 Score: 0.294444
Epoch: 2/3... Step: 370... Loss: 1.865187... Val Loss: 1.905935 Accuracy: 0.560241 F1 Score: 0.297899
Epoch: 2/3... Step: 380... Loss: 1.597906... Val Loss: 1.856457 Accuracy: 0.573795 F1 Score: 0.287783
Epoch: 2/3... Step: 390... Loss: 2.142184... Val Loss: 1.813165 Accuracy: 0.584086 F1 Score: 0.284695
Epoch: 2/3... Step: 400... Loss: 2.386818... Val Loss: 1.772914 Accuracy: 0.584839 F1 Score: 0.288637
Epoch: 2/3... Step: 410... Loss: 1.824174... Val Loss: 1.725866 Accuracy: 0.585090 F1 Score: 0.288698
```

```
Epoch: 2/3... Step: 420... Loss: 2.561416... Val Loss: 1.682773 Accuracy: 0.575552 F1 Score: 0.296080
Epoch: 2/3... Step: 430... Loss: 1.212438... Val Loss: 1.644377 Accuracy: 0.552711 F1 Score: 0.306503
Epoch: 2/3... Step: 440... Loss: 1.420536... Val Loss: 1.610392 Accuracy: 0.548695 F1 Score: 0.314412
Epoch: 2/3... Step: 450... Loss: 0.660223... Val Loss: 1.562900 Accuracy: 0.570281 F1 Score: 0.293647
Epoch: 2/3... Step: 460... Loss: 2.907702... Val Loss: 1.540976 Accuracy: 0.628514 F1 Score: 0.297616
Epoch: 2/3... Step: 470... Loss: 2.355066... Val Loss: 1.506574 Accuracy: 0.628765 F1 Score: 0.297710
Epoch: 2/3... Step: 480... Loss: 0.852550... Val Loss: 1.461679 Accuracy: 0.613956 F1 Score: 0.292848
Epoch: 2/3... Step: 490... Loss: 1.505022... Val Loss: 1.426756 Accuracy: 0.604920 F1 Score: 0.292720
Epoch: 2/3... Step: 500... Loss: 1.631076... Val Loss: 1.392660 Accuracy: 0.581074 F1 Score: 0.292208
Epoch: 2/3... Step: 510... Loss: 1.540705... Val Loss: 1.363085 Accuracy: 0.574799 F1 Score: 0.304128
Epoch: 2/3... Step: 520... Loss: 1.538772... Val Loss: 1.333579 Accuracy: 0.572540 F1 Score: 0.294587
Epoch: 2/3... Step: 530... Loss: 1.648823... Val Loss: 1.305132 Accuracy: 0.580572 F1 Score: 0.291073
Epoch: 2/3... Step: 540... Loss: 1.716034... Val Loss: 1.281219 Accuracy: 0.603414 F1 Score: 0.292670
Epoch: 2/3... Step: 550... Loss: 1.264195... Val Loss: 1.257672 Accuracy: 0.599900 F1 Score: 0.291674
Epoch: 2/3... Step: 560... Loss: 2.038176... Val Loss: 1.238865 Accuracy: 0.604920 F1 Score: 0.293087
Epoch: 2/3... Step: 570... Loss: 1.103141... Val Loss: 1.223950 Accuracy: 0.607681 F1 Score: 0.291192
Epoch: 2/3... Step: 580... Loss: 1.349122... Val Loss: 1.201666 Accuracy: 0.605673 F1 Score: 0.295777
Epoch: 2/3... Step: 590... Loss: 0.796593... Val Loss: 1.183039 Accuracy: 0.612199 F1 Score: 0.297478
Epoch: 2/3... Step: 600... Loss: 1.236634... Val Loss: 1.170833 Accuracy: 0.626757 F1 Score: 0.297353
Epoch: 2/3... Step: 610... Loss: 1.741099... Val Loss: 1.159478 Accuracy: 0.637550 F1 Score: 0.311550
Epoch: 2/3... Step: 620... Loss: 1.488015... Val Loss: 1.148390 Accuracy: 0.635542 F1 Score: 0.305304
Epoch: 2/3... Step: 630... Loss: 1.203978... Val Loss: 1.124998 Accuracy: 0.628012 F1 Score: 0.315331
Epoch: 2/3... Step: 640... Loss: 0.700863... Val Loss: 1.101923 Accuracy: 0.622741 F1 Score: 0.322956
Epoch: 2/3... Step: 650... Loss: 2.546109... Val Loss: 1.087306 Accuracy: 0.631024 F1 Score: 0.357623
Epoch: 2/3... Step: 660... Loss: 0.904279... Val Loss: 1.069776 Accuracy: 0.620482 F1 Score: 0.329958
Epoch: 2/3... Step: 670... Loss: 1.942629... Val Loss: 1.057898 Accuracy: 0.622239 F1 Score: 0.325482
Epoch: 2/3... Step: 680... Loss: 1.761355... Val Loss: 1.046734 Accuracy: 0.630522 F1 Score: 0.325588
Epoch: 2/3... Step: 690... Loss: 1.659758... Val Loss: 1.035377 Accuracy: 0.619729 F1 Score: 0.320838
Epoch: 2/3... Step: 700... Loss: 1.278997... Val Loss: 1.022600 Accuracy: 0.621737 F1 Score: 0.328035
Epoch: 2/3... Step: 710... Loss: 0.888218... Val Loss: 1.017099 Accuracy: 0.617470 F1 Score: 0.314601
Epoch: 2/3... Step: 720... Loss: 1.275081... Val Loss: 1.006779 Accuracy: 0.619980 F1 Score: 0.328885
Epoch: 2/3... Step: 730... Loss: 1.290386... Val Loss: 0.996511 Accuracy: 0.603163 F1 Score: 0.321600
Epoch: 2/3... Step: 740... Loss: 0.846276... Val Loss: 0.985424 Accuracy: 0.604418 F1 Score: 0.346806
Epoch: 2/3... Step: 750... Loss: 0.783552... Val Loss: 0.980872 Accuracy: 0.621235 F1 Score: 0.360631
Epoch: 2/3... Step: 760... Loss: 0.743672... Val Loss: 0.980356 Accuracy: 0.620733 F1 Score: 0.328151
Epoch: 2/3... Step: 770... Loss: 0.847377... Val Loss: 0.971461 Accuracy: 0.629267 F1 Score: 0.355171
Epoch: 2/3... Step: 780... Loss: 0.500257... Val Loss: 0.965975 Accuracy: 0.630522 F1 Score: 0.359783
Epoch: 2/3... Step: 790... Loss: 0.369929... Val Loss: 0.968338 Accuracy: 0.630773 F1 Score: 0.314470
Epoch: 2/3... Step: 800... Loss: 1.646808... Val Loss: 0.958833 Accuracy: 0.639307 F1 Score: 0.353174
Epoch: 2/3... Step: 810... Loss: 0.680134... Val Loss: 0.948777 Accuracy: 0.622741 F1 Score: 0.344971
Epoch: 2/3... Step: 820... Loss: 1.047904... Val Loss: 0.942335 Accuracy: 0.631275 F1 Score: 0.374946
Epoch: 2/3... Step: 830... Loss: 1.655854... Val Loss: 0.943032 Accuracy: 0.636546 F1 Score: 0.372334
Epoch: 2/3... Step: 840... Loss: 0.871464... Val Loss: 0.935684 Accuracy: 0.636546 F1 Score: 0.372334
Epoch: 2/3... Step: 850... Loss: 0.635389... Val Loss: 0.931070 Accuracy: 0.612199 F1 Score: 0.382821
Epoch: 2/3... Step: 860... Loss: 0.762603... Val Loss: 0.929133 Accuracy: 0.597892 F1 Score: 0.355358
Epoch: 2/3... Step: 870... Loss: 1.216758... Val Loss: 0.924921 Accuracy: 0.628765 F1 Score: 0.376858
Epoch: 2/3... Step: 880... Loss: 1.299709... Val Loss: 0.920722 Accuracy: 0.623243 F1 Score: 0.356752
Epoch: 2/3... Step: 890... Loss: 0.639298... Val Loss: 0.917757 Accuracy: 0.628263 F1 Score: 0.348592
Epoch: 2/3... Step: 900... Loss: 0.800798... Val Loss: 0.912619 Accuracy: 0.630773 F1 Score: 0.377296
Epoch: 2/3... Step: 910... Loss: 0.507967... Val Loss: 0.908795 Accuracy: 0.630020 F1 Score: 0.359572
Epoch: 2/3... Step: 920... Loss: 0.675219... Val Loss: 0.910801 Accuracy: 0.636546 F1 Score: 0.372334
Epoch: 2/3... Step: 930... Loss: 1.208724... Val Loss: 0.907813 Accuracy: 0.634287 F1 Score: 0.361371
Epoch: 2/3... Step: 940... Loss: 0.949298... Val Loss: 0.905335 Accuracy: 0.639307 F1 Score: 0.353174
Epoch: 2/3... Step: 950... Loss: 0.958683... Val Loss: 0.897211 Accuracy: 0.635040 F1 Score: 0.351389
Epoch: 2/3... Step: 960... Loss: 1.192219... Val Loss: 0.892763 Accuracy: 0.628263 F1 Score: 0.348592
Epoch: 2/3... Step: 970... Loss: 0.917772... Val Loss: 0.892019 Accuracy: 0.598896 F1 Score: 0.356670
Epoch: 2/3... Step: 980... Loss: 1.108001... Val Loss: 0.884317 Accuracy: 0.623243 F1 Score: 0.356752
Epoch: 2/3... Step: 990... Loss: 1.024502... Val Loss: 0.882259 Accuracy: 0.634538 F1 Score: 0.371483
999 steps in epoch 2
```

```
Epoch: 2, Average Accuracy: 0.63453815, Average f1: 0.37148316

<Figure size 432x288 with 0 Axes>
```

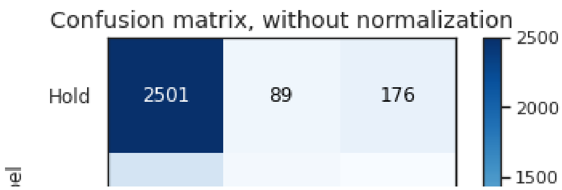



```
Starting epoch 3
Epoch: 3/3... Step: 10... Loss: 0.353822... Val Loss: 0.875092 Accuracy: 0.625502 F1 Score: 0.367705
Epoch: 3/3... Step: 20... Loss: 0.611452... Val Loss: 0.875377 Accuracy: 0.638303 F1 Score: 0.340754
Epoch: 3/3... Step: 30... Loss: 0.949969... Val Loss: 0.875212 Accuracy: 0.640813 F1 Score: 0.341821
Epoch: 3/3... Step: 40... Loss: 1.387490... Val Loss: 0.874059 Accuracy: 0.651857 F1 Score: 0.359800
Epoch: 3/3... Step: 50... Loss: 0.717128... Val Loss: 0.873170 Accuracy: 0.651857 F1 Score: 0.359800
Epoch: 3/3... Step: 60... Loss: 1.767632... Val Loss: 0.863905 Accuracy: 0.642570 F1 Score: 0.343910
Epoch: 3/3... Step: 70... Loss: 1.201651... Val Loss: 0.863213 Accuracy: 0.601908 F1 Score: 0.338165
Epoch: 3/3... Step: 80... Loss: 1.062871... Val Loss: 0.858582 Accuracy: 0.631526 F1 Score: 0.337975
Epoch: 3/3... Step: 90... Loss: 0.480357... Val Loss: 0.857031 Accuracy: 0.647590 F1 Score: 0.358033
Epoch: 3/3... Step: 100... Loss: 0.565096... Val Loss: 0.854754 Accuracy: 0.658384 F1 Score: 0.351786
Epoch: 3/3... Step: 110... Loss: 0.814757... Val Loss: 0.853603 Accuracy: 0.660141 F1 Score: 0.352817
Epoch: 3/3... Step: 120... Loss: 0.993017... Val Loss: 0.849804 Accuracy: 0.659388 F1 Score: 0.370264
Epoch: 3/3... Step: 130... Loss: 0.480129... Val Loss: 0.844035 Accuracy: 0.655622 F1 Score: 0.364692
Epoch: 3/3... Step: 140... Loss: 0.579046... Val Loss: 0.841159 Accuracy: 0.651355 F1 Score: 0.348008
Epoch: 3/3... Step: 150... Loss: 0.587690... Val Loss: 0.840256 Accuracy: 0.652610 F1 Score: 0.348800
Epoch: 3/3... Step: 160... Loss: 0.875675... Val Loss: 0.838330 Accuracy: 0.655622 F1 Score: 0.368085
Epoch: 3/3... Step: 170... Loss: 0.688598... Val Loss: 0.837048 Accuracy: 0.654618 F1 Score: 0.349617
Epoch: 3/3... Step: 180... Loss: 0.873904... Val Loss: 0.842643 Accuracy: 0.659137 F1 Score: 0.351245
Epoch: 3/3... Step: 190... Loss: 0.616247... Val Loss: 0.841903 Accuracy: 0.656376 F1 Score: 0.370152
Epoch: 3/3... Step: 200... Loss: 0.483705... Val Loss: 0.836089 Accuracy: 0.652610 F1 Score: 0.368584
Epoch: 3/3... Step: 210... Loss: 0.580328... Val Loss: 0.828677 Accuracy: 0.638303 F1 Score: 0.360083
Epoch: 3/3... Step: 220... Loss: 0.664095... Val Loss: 0.827331 Accuracy: 0.633032 F1 Score: 0.385345
```

```
Epoch: 3/3... Step: 230... Loss: 0.851288... Val Loss: 0.824253 Accuracy: 0.644829 F1 Score: 0.364117
Epoch: 3/3... Step: 240... Loss: 1.022319... Val Loss: 0.824652 Accuracy: 0.632279 F1 Score: 0.370531
Epoch: 3/3... Step: 250... Loss: 0.903802... Val Loss: 0.820531 Accuracy: 0.632279 F1 Score: 0.370531
Epoch: 3/3... Step: 260... Loss: 1.205909... Val Loss: 0.820947 Accuracy: 0.634538 F1 Score: 0.369606
Epoch: 3/3... Step: 270... Loss: 0.687946... Val Loss: 0.819281 Accuracy: 0.630020 F1 Score: 0.359572
Epoch: 3/3... Step: 280... Loss: 1.112210... Val Loss: 0.822905 Accuracy: 0.652861 F1 Score: 0.367896
Epoch: 3/3... Step: 290... Loss: 1.196929... Val Loss: 0.822720 Accuracy: 0.655120 F1 Score: 0.369341
Epoch: 3/3... Step: 300... Loss: 1.087890... Val Loss: 0.818300 Accuracy: 0.660643 F1 Score: 0.373940
Epoch: 3/3... Step: 310... Loss: 0.888015... Val Loss: 0.811114 Accuracy: 0.661647 F1 Score: 0.375273
Epoch: 3/3... Step: 320... Loss: 0.839935... Val Loss: 0.808115 Accuracy: 0.637801 F1 Score: 0.370459
Epoch: 3/3... Step: 330... Loss: 0.573175... Val Loss: 0.808086 Accuracy: 0.642319 F1 Score: 0.372783
Epoch: 3/3... Step: 340... Loss: 0.694908... Val Loss: 0.804882 Accuracy: 0.653112 F1 Score: 0.362847
Epoch: 3/3... Step: 350... Loss: 0.584025... Val Loss: 0.803149 Accuracy: 0.672942 F1 Score: 0.361656
Epoch: 3/3... Step: 360... Loss: 0.927471... Val Loss: 0.799238 Accuracy: 0.644076 F1 Score: 0.358196
Epoch: 3/3... Step: 370... Loss: 0.652007... Val Loss: 0.801465 Accuracy: 0.654869 F1 Score: 0.379406
Epoch: 3/3... Step: 380... Loss: 0.833935... Val Loss: 0.799088 Accuracy: 0.654869 F1 Score: 0.379406
Epoch: 3/3... Step: 390... Loss: 0.492992... Val Loss: 0.794753 Accuracy: 0.654367 F1 Score: 0.363461
Epoch: 3/3... Step: 400... Loss: 0.912592... Val Loss: 0.791756 Accuracy: 0.658886 F1 Score: 0.365687
Epoch: 3/3... Step: 410... Loss: 1.170504... Val Loss: 0.791341 Accuracy: 0.670432 F1 Score: 0.410922
Epoch: 3/3... Step: 420... Loss: 1.192153... Val Loss: 0.789118 Accuracy: 0.645331 F1 Score: 0.399602
Epoch: 3/3... Step: 430... Loss: 0.793181... Val Loss: 0.787331 Accuracy: 0.628765 F1 Score: 0.401643
Epoch: 3/3... Step: 440... Loss: 0.646757... Val Loss: 0.789330 Accuracy: 0.626506 F1 Score: 0.419355
Epoch: 3/3... Step: 450... Loss: 0.519722... Val Loss: 0.783634 Accuracy: 0.641064 F1 Score: 0.393910
Epoch: 3/3... Step: 460... Loss: 0.745183... Val Loss: 0.784227 Accuracy: 0.663404 F1 Score: 0.354764
Epoch: 3/3... Step: 470... Loss: 0.914445... Val Loss: 0.790069 Accuracy: 0.667671 F1 Score: 0.377753
Epoch: 3/3... Step: 480... Loss: 0.583088... Val Loss: 0.785631 Accuracy: 0.664408 F1 Score: 0.376405
Epoch: 3/3... Step: 490... Loss: 0.831235... Val Loss: 0.779554 Accuracy: 0.660643 F1 Score: 0.375121
Epoch: 3/3... Step: 500... Loss: 0.932333... Val Loss: 0.777811 Accuracy: 0.631526 F1 Score: 0.387322
Epoch: 3/3... Step: 510... Loss: 0.913283... Val Loss: 0.779836 Accuracy: 0.639307 F1 Score: 0.412454
Epoch: 3/3... Step: 520... Loss: 0.688634... Val Loss: 0.779447 Accuracy: 0.651857 F1 Score: 0.430245
Epoch: 3/3... Step: 530... Loss: 0.723709... Val Loss: 0.775806 Accuracy: 0.654116 F1 Score: 0.404458
Epoch: 3/3... Step: 540... Loss: 1.080217... Val Loss: 0.773267 Accuracy: 0.664408 F1 Score: 0.395310
Epoch: 3/3... Step: 550... Loss: 1.012102... Val Loss: 0.773219 Accuracy: 0.666165 F1 Score: 0.365897
Epoch: 3/3... Step: 560... Loss: 1.031577... Val Loss: 0.772728 Accuracy: 0.676205 F1 Score: 0.362869
Epoch: 3/3... Step: 570... Loss: 0.933771... Val Loss: 0.775451 Accuracy: 0.674699 F1 Score: 0.344560
Epoch: 3/3... Step: 580... Loss: 0.698348... Val Loss: 0.776034 Accuracy: 0.675703 F1 Score: 0.344966
Epoch: 3/3... Step: 590... Loss: 0.685248... Val Loss: 0.778548 Accuracy: 0.676707 F1 Score: 0.338854
Epoch: 3/3... Step: 600... Loss: 0.630876... Val Loss: 0.780931 Accuracy: 0.695281 F1 Score: 0.323281
Epoch: 3/3... Step: 610... Loss: 1.218565... Val Loss: 0.784405 Accuracy: 0.698293 F1 Score: 0.348080
Epoch: 3/3... Step: 620... Loss: 1.077573... Val Loss: 0.777821 Accuracy: 0.691516 F1 Score: 0.322081
Epoch: 3/3... Step: 630... Loss: 1.041182... Val Loss: 0.770932 Accuracy: 0.669679 F1 Score: 0.314315
Epoch: 3/3... Step: 640... Loss: 0.616337... Val Loss: 0.765640 Accuracy: 0.671687 F1 Score: 0.329040
Epoch: 3/3... Step: 650... Loss: 1.278681... Val Loss: 0.761769 Accuracy: 0.666165 F1 Score: 0.340257
Epoch: 3/3... Step: 660... Loss: 0.790647... Val Loss: 0.757877 Accuracy: 0.649849 F1 Score: 0.355188
Epoch: 3/3... Step: 670... Loss: 1.114346... Val Loss: 0.756072 Accuracy: 0.672440 F1 Score: 0.364119
Epoch: 3/3... Step: 680... Loss: 0.908533... Val Loss: 0.759985 Accuracy: 0.666918 F1 Score: 0.358688
Epoch: 3/3... Step: 690... Loss: 1.113330... Val Loss: 0.768243 Accuracy: 0.661647 F1 Score: 0.381831
Epoch: 3/3... Step: 700... Loss: 0.663372... Val Loss: 0.763004 Accuracy: 0.659388 F1 Score: 0.380896
Epoch: 3/3... Step: 710... Loss: 0.700064... Val Loss: 0.757679 Accuracy: 0.656627 F1 Score: 0.365936
Epoch: 3/3... Step: 720... Loss: 0.937571... Val Loss: 0.753626 Accuracy: 0.655371 F1 Score: 0.381848
Epoch: 3/3... Step: 730... Loss: 0.932740... Val Loss: 0.750774 Accuracy: 0.663906 F1 Score: 0.408128
Epoch: 3/3... Step: 740... Loss: 0.792198... Val Loss: 0.747508 Accuracy: 0.660643 F1 Score: 0.418967
Epoch: 3/3... Step: 750... Loss: 0.726024... Val Loss: 0.744527 Accuracy: 0.668675 F1 Score: 0.406924
Epoch: 3/3... Step: 760... Loss: 0.770706... Val Loss: 0.745941 Accuracy: 0.668675 F1 Score: 0.359401
Epoch: 3/3... Step: 770... Loss: 0.664386... Val Loss: 0.742779 Accuracy: 0.681476 F1 Score: 0.384813
Epoch: 3/3... Step: 780... Loss: 0.569884... Val Loss: 0.741860 Accuracy: 0.681225 F1 Score: 0.357345
Epoch: 3/3... Step: 790... Loss: 0.351886... Val Loss: 0.746873 Accuracy: 0.670683 F1 Score: 0.302976
Epoch: 3/3... Step: 800... Loss: 0.941921... Val Loss: 0.743962 Accuracy: 0.678464 F1 Score: 0.324855
Epoch: 3/3... Step: 810... Loss: 0.666099... Val Loss: 0.739311 Accuracy: 0.676707 F1 Score: 0.324009
Epoch: 3/3... Step: 820... Loss: 0.856247... Val Loss: 0.737933 Accuracy: 0.684237 F1 Score: 0.386471
```

```
Epoch: 3/3... Step: 830... Loss: 1.098079... Val Loss: 0.743801 Accuracy: 0.661647 F1 Score: 0.355097
Epoch: 3/3... Step: 840... Loss: 0.621801... Val Loss: 0.739823 Accuracy: 0.665663 F1 Score: 0.370731
Epoch: 3/3... Step: 850... Loss: 0.500689... Val Loss: 0.734995 Accuracy: 0.649849 F1 Score: 0.386021
Epoch: 3/3... Step: 860... Loss: 0.572876... Val Loss: 0.735070 Accuracy: 0.663906 F1 Score: 0.408128
Epoch: 3/3... Step: 870... Loss: 0.867050... Val Loss: 0.733405 Accuracy: 0.660894 F1 Score: 0.401587
Epoch: 3/3... Step: 880... Loss: 1.006020... Val Loss: 0.731848 Accuracy: 0.650351 F1 Score: 0.323897
Epoch: 3/3... Step: 890... Loss: 0.668066... Val Loss: 0.731475 Accuracy: 0.655622 F1 Score: 0.324690
Epoch: 3/3... Step: 900... Loss: 0.499151... Val Loss: 0.730206 Accuracy: 0.654367 F1 Score: 0.339680
Epoch: 3/3... Step: 910... Loss: 0.537363... Val Loss: 0.728216 Accuracy: 0.670432 F1 Score: 0.353017
Epoch: 3/3... Step: 920... Loss: 0.575633... Val Loss: 0.732582 Accuracy: 0.678464 F1 Score: 0.365447
Epoch: 3/3... Step: 930... Loss: 0.789711... Val Loss: 0.734631 Accuracy: 0.678966 F1 Score: 0.306944
Epoch: 3/3... Step: 940... Loss: 0.775372... Val Loss: 0.737511 Accuracy: 0.688253 F1 Score: 0.310207
Epoch: 3/3... Step: 950... Loss: 0.703375... Val Loss: 0.729880 Accuracy: 0.678464 F1 Score: 0.324855
Epoch: 3/3... Step: 960... Loss: 0.895232... Val Loss: 0.724605 Accuracy: 0.666918 F1 Score: 0.331152
Epoch: 3/3... Step: 970... Loss: 0.650106... Val Loss: 0.724962 Accuracy: 0.657129 F1 Score: 0.345674
Epoch: 3/3... Step: 980... Loss: 0.807497... Val Loss: 0.721779 Accuracy: 0.659639 F1 Score: 0.327497
Epoch: 3/3... Step: 990... Loss: 0.981043... Val Loss: 0.721813 Accuracy: 0.663906 F1 Score: 0.379693
999 steps in epoch 3

Epoch: 3, Average Accuracy: 0.66390562, Average f1: 0.37969266

<Figure size 432x288 with 0 Axes>
```




## V. BERT Model

### Configure Model

```
#from transformers import *
from transformers import BertTokenizer, BertForSequenceClassification, BertModel
```

```
class InputFeature(object):
    def __init__(self, id, input_ids, masks, segments, meta, label=None):
        self.id = id
        self.features = {
            'input_ids': input_ids,
            'input_mask': masks,
            'segment_ids': segments,
            'meta': meta
        }
        self.label = label
```

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', do_lower_case=True)

def bert_encoder(text, max_len=200):
```

```
    text_token = tokenizer.tokenize(text)
    text_token = text_token[:max_len-2]
    text_token = ["[CLS]"] + text_token + ["[SEP]"]
    text_ids = tokenizer.convert_tokens_to_ids(text_token)
    text_ids += [0] * (max_len - len(text_token))
    pad_masks = [1] * len(text_token) + [0] * (max_len - len(text_token))
    segment_ids = [0] * len(text_token) + [0] * (max_len - len(text_token))

    return text_ids, pad_masks, segment_ids
```

Downloading: 100% 232k/232k [00:00<00:00, 629kB/s]

```
train_set = []
max_seq_length = 200
meta_size = 10

for index, row in tqdm(split_train_df.iterrows(), total=split_train_df.shape[0]):
    input_ids, masks, segments = bert_encoder(row['text'], max_seq_length)
    train_set.append(InputFeature(row.index, input_ids, masks, segments, row[nontext_columns + ['tone']], int(row['target'])))

train_labels = split_train_df['target'].astype(int).values
train_valid_input_ids = np.array([data.features['input_ids'] for data in train_set])
train_valid_input_masks = np.array([data.features['input_mask'] for data in train_set])
train_valid_segment_ids =np.array([data.features['segment_ids'] for data in train_set])
train_valid_meta =np.array([data.features['meta'] for data in train_set], dtype=np.float64)
train_valid_labels = np.array([data.label for data in train_set])

oof_train = np.zeros((len(split_train_df), 3), dtype=np.float32)
```

100% 19974/19974 [39:37<00:00, 8.40it/s]

```
print(train_valid_meta[0])
print(train_valid_meta[1])
```

```
[ 1.00000000e+00  1.04316469e+00  5.58000000e+01  2.61000000e+02
  1.80763085e+00  0.00000000e+00 -1.59017884e-02  1.49014176e+01
  3.58787053e-02 -1.46710838e-01]
[ 1.00000000e+00  1.04316469e+00  5.58000000e+01  2.61000000e+02
  1.80763085e+00  0.00000000e+00 -1.59017884e-02  1.49014176e+01
  3.58787053e-02 -1.46710838e-01]
```

## Model

```
class BertTextClassifier(nn.Module):
    def __init__(self, hidden_size, dense_size, meta_size, output_size, dropout=0.1):
```

```
        super().__init__()
        self.output_size = output_size
        self.dropout = dropout

        self.bert = BertModel.from_pretrained('bert-base-uncased',
                                output_hidden_states=True,
                                output_attentions=True)
        for param in self.bert.parameters():
            param.requires_grad = True
        self.weights = nn.Parameter(torch.rand(13, 1))
        self.dropout = nn.Dropout(dropout)
        self.fc1 = nn.Linear(hidden_size, dense_size)
        self.fc2 = nn.Linear(dense_size + meta_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input_ids, nn_input_meta):
        all_hidden_states, all_attentions = self.bert(input_ids)[-2:]
        batch_size = input_ids.shape[0]
        ht_cls = torch.cat(all_hidden_states)[:, :1, :].view(13, batch_size, 1, 768)
        atten = torch.sum(ht_cls * self.weights.view(13, 1, 1, 1), dim=[1, 3])
        atten = F.softmax(atten.view(-1), dim=0)
        feature = torch.sum(ht_cls * atten.view(13, 1, 1, 1), dim=[0, 2])
        # Dense layer
        dense_out = self.fc1(self.dropout(feature))
        concat_layer = torch.cat((dense_out, nn_input_meta.float()), 1)
        # print(len(dense_out[0]))
        # print(len(nn_input_meta[0]))
        # print(len(concat_layer[0]))
        # print("dense_out: \n", dense_out)
        # print("nn_input_meta: \n", nn_input_meta)
        # print("concat_layer: \n", concat_layer)
        out = self.fc2(concat_layer)
        #logps = self.softmax(out)

        return out



# Check how BertTokenizer works
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')
torch.save(model.state_dict(), model_dir + 'bert_case_unbased')

input_ids = torch.tensor(tokenizer.encode("Hello, my dog is cute", add_special_tokens=True)).unsqueeze(0)  # Batch size 1
outputs = model(input_ids)

print(input_ids)
print(outputs) # The last hidden-state is the first element of the output tuple
```

Downloading: 100% 433/433 [00:00<00:00, 1.16kB/s]

Downloading: 100% 440M/440M [00:07<00:00, 61.4MB/s]

```
tensor([[  101,  7592,  1010,  2026,  3899,  2003, 10140,   102]])
(tensor([[[-0.1144,  0.1937,  0.1250,  ..., -0.3827,  0.2107,  0.5407],
         [ 0.5308,  0.3207,  0.3665,  ..., -0.0036,  0.7579,  0.0388],
         [-0.4877,  0.8849,  0.4256,  ..., -0.6976,  0.4458,  0.1231],
         ...,
         [-0.7003, -0.1815,  0.3297,  ..., -0.4838,  0.0680,  0.8901],
         [-1.0355, -0.2567, -0.0317,  ...,  0.3197,  0.3999,  0.1795],
         [ 0.6080,  0.2610, -0.3131,  ...,  0.0311, -0.6283, -0.1994]]],
       grad_fn=<NativeLayerNormBackward>), tensor([[-7.1946e-01, -2.1445e-01, -2.9576e-01,  3.6603e-01,  2.7968e-01,
          2.2184e-02,  5.7299e-01,  6.2331e-02,  5.9586e-02, -9.9965e-01,
          5.0146e-02,  4.4756e-01,  9.7612e-01,  3.3988e-02,  8.4494e-01,
         -3.6905e-01,  9.8649e-02, -3.7169e-01,  1.7371e-01,  1.1515e-01,
          4.4133e-01,  9.9525e-01,  3.7221e-01,  8.2881e-02,  2.1402e-01,
          6.8965e-01, -6.1042e-01,  8.7136e-01,  9.4158e-01,  5.7372e-01,
         -3.2187e-01,  8.6672e-03, -9.8611e-01, -2.0542e-02, -4.3756e-01,
         -9.8012e-01,  1.1142e-01, -6.7587e-01,  1.3499e-01,  3.1130e-01,
         -8.2997e-01,  1.9006e-01,  9.9896e-01, -3.1798e-01,  2.1517e-02,
         -1.6531e-01, -9.9943e-01,  1.0173e-01, -8.1811e-01,  3.3119e-02,
          3.6740e-01, -7.3230e-02, -1.4261e-01,  1.8907e-01,  2.6119e-01,
          4.1582e-01, -2.4427e-01, -5.9846e-02, -7.3492e-02, -3.4202e-01,
         -5.8001e-01,  2.8331e-01, -5.0513e-01, -8.1967e-01,  1.9813e-01,
          1.9108e-01,  3.7011e-02, -1.1327e-01,  1.3472e-01, -2.1614e-01,
          6.3494e-01,  2.4869e-02,  3.8287e-01, -8.1779e-01, -2.4874e-01,
          8.4982e-02, -5.2998e-01,  1.0000e+00, -5.2155e-02, -9.7052e-01,
          3.9848e-01,  2.1360e-02,  3.9035e-01,  3.5588e-01, -1.7881e-01,
         -9.9997e-01,  2.6939e-01, -3.8057e-02, -9.8657e-01,  6.9322e-02,
          3.9138e-01, -2.1884e-02, -9.6332e-02,  3.8545e-01, -3.4136e-01,
         -8.0362e-02, -3.2022e-02, -3.6328e-01, -7.8130e-02,  1.9192e-02,
         -1.3429e-01, -1.6013e-02, -5.2640e-02, -2.8006e-01,  9.3611e-02,
         -2.2885e-01, -1.2305e-01, -1.1002e-01, -3.2808e-01,  4.0356e-01,
          2.8048e-01, -2.0102e-01,  2.7685e-01, -9.4023e-01,  4.1756e-01,
         -1.5473e-01, -9.7553e-01, -4.3003e-01, -9.8546e-01,  5.9158e-01,
          3.7344e-02, -1.9320e-01,  9.1691e-01,  3.6012e-01,  1.4505e-01,
          1.5398e-01, -1.0657e-02, -1.0000e+00, -3.1573e-01, -3.1037e-01,
          1.6523e-01, -8.0330e-02, -9.6650e-01, -9.4546e-01,  3.6145e-01,
          9.0138e-01, -7.2696e-02,  9.9774e-01,  3.7289e-02,  9.3599e-01,
          2.5317e-01, -2.0185e-01,  2.9533e-02, -2.3162e-01,  3.4632e-01,
         -1.0763e-01, -2.6565e-01,  1.0874e-01,  1.2985e-01,  2.1134e-02,
         -9.6283e-02, -7.6358e-02, -6.5149e-02, -8.9277e-01, -2.3465e-01,
          9.1176e-01,  7.0429e-02, -2.1429e-01,  3.8197e-01,  3.5892e-02,
         -1.6971e-01,  7.0654e-01,  2.4045e-01,  1.5014e-01, -1.9478e-02,
          2.1369e-01, -1.7977e-01,  3.5112e-01, -6.0260e-01,  4.1683e-01,
          1.8090e-01, -3.2497e-02, -3.0137e-01, -9.7103e-01, -1.3917e-01,
          3.5130e-01,  9.8326e-01,  5.2702e-01,  4.8812e-02,  1.3991e-02,
         -6.7964e-02,  2.9718e-01, -9.4136e-01,  9.7219e-01, -2.4774e-02,
          1.5224e-01, -1.8241e-01,  5.5584e-02, -7.7306e-01, -9.9000e-02,
          4.7058e-01, -1.7022e-01, -7.7803e-01,  5.2834e-02, -3.7679e-01,
         -4.1296e-02, -4.9612e-01,  1.4171e-01, -1.1803e-01, -1.8995e-01,
          5.0384e-02,  9.0623e-01,  7.8828e-01,  5.2288e-01, -3.5274e-01,
          2.8563e-01, -8.1494e-01, -1.9622e-01, -9.2975e-02,  5.9311e-02,
          3.1902e-02,  9.8860e-01, -3.9452e-01,  1.1867e-01, -8.6977e-01,
         -9.7789e-01, -1.4859e-01, -7.7064e-01, -4.0617e-03, -4.1152e-01,
          3.2578e-01,  1.8777e-01, -2.4501e-01,  2.6668e-01, -7.9329e-01,
         -4.8133e-01,  9.3245e-02, -1.7010e-01,  2.7043e-01, -3.5880e-02,
```

```
 7.7973e-01,  4.6696e-01, -3.4636e-01,  5.5238e-02,  9.0312e-01,
-2.4115e-01, -6.4200e-01,  4.1441e-01, -9.7797e-02,  6.2983e-01,
-4.1787e-01,  9.4069e-01,  4.9285e-01,  3.6058e-01, -8.7901e-01,
-2.6726e-01, -5.4679e-01,  9.3944e-04, -1.0502e-02, -4.6837e-01,
 3.1116e-01,  3.6999e-01,  1.3306e-01,  6.4092e-01, -3.5630e-01,
 8.8549e-01, -8.9036e-01, -9.3865e-01, -8.1215e-01,  2.7362e-01,
-9.8566e-01,  4.0363e-01,  2.1223e-01, -1.4316e-01, -2.4553e-01,
-2.1144e-01, -9.4728e-01,  5.0806e-01, -9.6622e-02,  8.5571e-01,
-1.0133e-01, -6.7768e-01, -2.8500e-01, -8.9905e-01, -3.3577e-01,
 8.9155e-02,  3.2600e-01, -2.6467e-01, -9.2032e-01,  3.4629e-01,
 3.3430e-01,  2.1397e-01,  3.0630e-02,  9.3878e-01,  9.9986e-01,
 9.6385e-01,  8.3159e-01,  6.2250e-01, -9.8055e-01, -7.3623e-01,
 9.9986e-01, -7.8395e-01, -9.9998e-01, -8.7800e-01, -5.0893e-01,
 2.3399e-02, -1.0000e+00, -6.1938e-02,  1.9563e-01, -9.0552e-01,
-1.4008e-01,  9.5264e-01,  7.9837e-01, -1.0000e+00,  7.6343e-01,
 8.3670e-01, -4.5859e-01,  5.4410e-01, -2.4073e-01,  9.6085e-01,
 1.9164e-01,  3.2135e-01, -1.3064e-02,  2.4534e-01, -5.3001e-01,
-5.9538e-01,  3.7464e-01, -2.1189e-01,  8.8024e-01,  1.9648e-02,
-3.8349e-01, -8.4779e-01,  1.4676e-02, -2.8375e-02, -4.4313e-01,
-9.4966e-01, -6.5704e-02, -7.2327e-02,  6.5967e-01, -1.1504e-01,
 2.1876e-01, -5.5254e-01,  9.2219e-02, -5.0583e-01, -5.2826e-02,
 5.1425e-01, -8.9533e-01, -1.2744e-01,  9.7845e-02, -6.0145e-01,
-3.1652e-02, -9.5186e-01,  9.4685e-01, -2.2341e-01,  1.8390e-01,
 1.0000e+00,  1.1755e-01, -7.0390e-01,  3.2502e-01, -1.0898e-02,
-1.8308e-01,  9.9999e-01,  5.8376e-01, -9.7387e-01, -3.3783e-01,
 2.9640e-01, -2.7002e-01, -2.2243e-01,  9.9711e-01,  1.4422e-02,
 7.8269e-02,  3.8660e-01,  9.7787e-01, -9.8501e-01,  8.7459e-01,
-7.2276e-01, -9.5249e-01,  9.4567e-01,  9.1005e-01, -5.0722e-01,
-4.9026e-01, -1.2517e-01, -3.9076e-02,  8.8128e-02, -8.2481e-01,
 3.8301e-01,  1.8045e-01,  5.4796e-02,  8.0041e-01, -3.3501e-01,
-3.9115e-01,  1.4233e-01, -9.0141e-02,  3.4585e-01,  4.4044e-01,
 3.1044e-01, -1.3280e-01, -1.3614e-01, -3.0303e-01, -4.8794e-01,
-9.4950e-01,  1.0887e-01,  1.0000e+00,  6.0752e-02,  8.3374e-02,
-3.1301e-03,  8.5578e-02, -3.1288e-01,  2.6283e-01,  2.6870e-01,
-1.4267e-01, -7.4000e-01,  2.2856e-01, -7.9442e-01, -9.8812e-01,
 4.3592e-01,  7.7229e-02, -3.8084e-02,  9.9490e-01,  3.2616e-01,
 6.7989e-02,  8.2888e-02,  4.7391e-01, -2.1855e-01,  3.9278e-01,
 3.7665e-02,  9.6440e-01, -1.8374e-01,  3.9259e-01,  4.3319e-01,
-1.8618e-01, -2.1584e-01, -4.9610e-01, -9.7025e-02, -8.8006e-01,
 2.4995e-01, -9.3940e-01,  9.3827e-01,  3.2001e-01,  1.1919e-01,
 7.3959e-02,  3.1273e-02,  1.0000e+00, -7.5631e-01,  3.5396e-01,
 5.3290e-01,  3.2036e-01, -9.7538e-01, -4.7482e-01, -2.3322e-01,
 3.5377e-02, -4.6060e-02, -1.2863e-01,  8.3798e-02, -9.5139e-01,
 3.4662e-02,  4.5219e-03, -8.8296e-01, -9.8300e-01,  1.6468e-01,
 3.3595e-01, -1.0217e-01, -7.0275e-01, -4.3307e-01, -5.4169e-01,
 1.8884e-01, -5.5797e-02, -9.2162e-01,  4.4790e-01, -3.5256e-02,
 2.1131e-01, -4.6267e-02,  4.1688e-01,  1.9311e-01,  8.2643e-01,
 3.1897e-02,  1.8036e-02,  2.2502e-02, -5.6261e-01,  5.2690e-01,
-4.1523e-01, -2.0335e-01,  5.0975e-03,  1.0000e+00, -1.3769e-01,
 4.0090e-01,  4.8581e-01,  3.0547e-01,  1.0161e-01,  1.1372e-01,
 5.4688e-01,  1.7282e-01, -1.1611e-01,  1.1692e-01,  3.3706e-01,
-9.4995e-02,  3.3125e-01, -1.1600e-01,  5.5663e-02,  6.9017e-01,
 5.2775e-01, -7.8248e-02,  7.7874e-02, -2.5570e-01,  9.5441e-01,
 4.4725e-02,  7.5062e-02, -1.6521e-01,  9.8572e-02, -1.2673e-01,
 4.2396e-01,  9.9999e-01,  1.4012e-01, -6.5118e-02, -9.8683e-01,
-3.4659e-01, -6.9549e-01,  9.9968e-01,  7.8693e-01, -6.2560e-01,
 4.0561e-01,  5.1398e-01, -7.1926e-03,  3.7469e-01, -4.9920e-02,
-1.8379e-01,  1.0699e-01,  6.4271e-02,  9.4363e-01, -4.5982e-01,
-9.6684e-01, -4.8714e-01,  1.6233e-01, -9.2982e-01,  9.8976e-01,
-2.8241e-01, -3.9526e-02, -2.8969e-01,  2.2178e-01, -7.3322e-01,
```

```
-1.9752e-01, -9.7385e-01,  1.4625e-01,  1.7384e-02,  9.4459e-01,
 8.0070e-02, -4.1026e-01, -7.2363e-01,  6.5494e-02,  2.9531e-01,
-2.0402e-01, -9.4453e-01,  9.4867e-01, -9.6224e-01,  4.1987e-01,
 9.9992e-01,  2.0182e-01, -5.9719e-01,  6.7062e-02, -1.3560e-01,
 1.1140e-01, -7.1071e-02,  3.3843e-01, -9.1928e-01, -1.1785e-01,
 7.1903e-03,  9.3813e-02,  1.2718e-01, -4.2175e-01,  6.2383e-01,
-3.0948e-02, -3.9573e-01, -4.9911e-01,  1.9713e-01,  1.9574e-01,
 5.2774e-01, -6.4998e-02,  3.8218e-02, -1.3764e-01,  1.3114e-01,
-8.2896e-01, -6.2801e-02, -1.3077e-01, -9.9745e-01,  3.8189e-01,
-1.0000e+00, -4.9528e-02, -3.3011e-01, -9.7048e-03,  7.4032e-01,
 4.5588e-01, -4.3038e-02, -5.9485e-01,  3.5139e-02,  8.4290e-01,
 7.0024e-01,  4.9502e-03,  1.5221e-01, -4.8182e-01,  3.4912e-02,
 6.8681e-02,  5.9797e-02,  9.4147e-02,  5.7532e-01,  3.5063e-02,
 1.0000e+00, -4.4784e-03, -3.4757e-01, -7.9309e-01,  5.7241e-02,
-4.8241e-02,  9.9991e-01, -3.6963e-01, -9.2729e-01,  2.2610e-01,
-3.2602e-01, -6.5948e-01,  2.3506e-01, -6.6026e-02, -6.2875e-01,
-4.7124e-01,  8.3105e-01,  4.3462e-01, -5.2237e-01,  2.1811e-01,
-1.1176e-01, -2.7027e-01, -6.8502e-02,  5.0503e-02,  9.8319e-01,
 3.3888e-01,  5.6442e-01,  1.0517e-01,  6.1441e-02,  9.3666e-01,
 7.3988e-02, -2.4528e-01, -8.5207e-02,  9.9998e-01,  1.4210e-01,
-8.2488e-01,  2.2405e-01, -9.2098e-01, -1.0235e-01, -8.4105e-01,
 2.1140e-01, -3.4107e-02,  8.0942e-01,  4.9841e-03,  8.9624e-01,
 6.7186e-02, -1.7137e-01, -2.7561e-01,  2.6385e-01,  1.9073e-01,
-8.6307e-01, -9.8238e-01, -9.8035e-01,  2.2370e-01, -3.5154e-01,
 1.9181e-01,  8.9503e-02, -9.8139e-02,  8.3593e-02,  3.0373e-01,
-9.9998e-01,  9.0944e-01,  2.9007e-01,  4.4585e-01,  9.4631e-01,
 4.1260e-01,  1.9621e-01,  2.4693e-01, -9.7562e-01, -7.6957e-01,
-1.7996e-01, -5.8601e-02,  4.2949e-01,  3.3341e-01,  8.0548e-01,
 2.5306e-01, -4.0736e-01, -3.4586e-02,  4.1000e-01, -8.3874e-01,
-9.9092e-01,  3.0937e-01,  3.3917e-01, -6.2679e-01,  9.4565e-01,
-5.9613e-01, -1.9438e-03,  3.7971e-01, -2.2250e-01,  5.2158e-01,
 5.9324e-01, -1.8357e-02, -6.8000e-03,  2.1554e-01,  8.2484e-01,
 8.0068e-01,  9.7795e-01, -1.0868e-01,  4.3963e-01,  2.2388e-01,
 2.7078e-01,  8.5065e-01, -9.2567e-01,  4.3628e-03, -3.2062e-02,
-1.9565e-01,  1.1169e-01, -9.4711e-02, -7.2645e-01,  6.3986e-01,
-1.7955e-01,  4.2939e-01, -2.0787e-01,  2.2294e-01, -2.3857e-01,
 6.7195e-02, -5.1772e-01, -3.6389e-01,  5.3170e-01,  5.3485e-02,
 8.5309e-01,  6.4611e-01,  1.2341e-02, -2.4756e-01,  1.4718e-02,
-5.3294e-02, -9.2566e-01,  5.0771e-01,  1.2492e-01,  2.1458e-01,
-6.7959e-02, -2.7113e-01,  9.0946e-01, -1.9032e-01, -2.1274e-01,
-6.4847e-02, -4.3871e-01,  6.3752e-01, -2.1017e-01, -2.9291e-01,
-3.1616e-01,  5.4117e-01,  1.6768e-01,  9.9424e-01, -9.4508e-02,
-2.9022e-01, -2.1880e-03, -1.5720e-01,  2.8317e-01, -2.9364e-01,
-9.9998e-01,  1.4066e-01,  9.1606e-02,  1.1457e-01, -2.1965e-01,
```

## Configure Model

```
-1.0402e-01, -5.4204e-01,  8.4934e-01]], grad_fn=<TanhBackward>))
```

```
# Test Tokenizer - Own Implementation
bert_model = BertTextClassifier(768, 128, meta_size, 3, dropout=0.1)

text_ids, pad_masks, segment_ids = bert_encoder("Hello, my dog is cute")
print('text_ids: \n', text_ids)
print('text_ids (torch.tensor): \n', torch.tensor(text_ids))
text_ids = torch.tensor(text_ids).unsqueeze(0)
print('text_ids (unsqueezed): \n', text_ids)
#print('pad_masks: ',pad_masks)
#print('segment_ids: ',segment_ids)
```

```
x_meta = torch.tensor([1,2,3,4,5,6,7,8,9,10]).unsqueeze(0)
outputs = bert_model(text_ids, x_meta)
print(len(outputs))
print('outputs: \n',outputs)
print('outputs(detached): \n', outputs.detach())
```

```
text_ids:
 [101, 7592, 1010, 2026, 3899, 2003, 10140, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
text_ids (torch.tensor):
 tensor([  101,  7592,  1010,  2026,  3899,  2003, 10140,   102,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0])
text_ids (unsqueezed):
 tensor([[  101,  7592,  1010,  2026,  3899,  2003, 10140,   102,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0]])
1
outputs:
 tensor([[-0.1071,  0.9974,  0.7125]], grad_fn=<AddmmBackward>)
outputs(detached):
 tensor([[-0.1071,  0.9974,  0.7125]])
```

```
# Hyperparameters
learning_rate = 1e-5
num_epochs = 3
batch_size = 32
patience =2
file_name = 'model'
use_skf = True
bert_hidden_size = 768
bert_dense_size =128
```

## Custom Training Model Definition

```
def train_bert(fold, train_indices, valid_indices):

    # Number of folds to iterrate
    # if fold == 3:
    #     break

    logger.info('================     fold {}        ==============='.format(fold))

    # Train Data in Tensor
    train_input_ids = torch.tensor(train_valid_input_ids[train_indices], dtype=torch.long)
    train_input_mask = torch.tensor(train_valid_input_masks[train_indices], dtype=torch.long)
    train_segment_ids = torch.tensor(train_valid_segment_ids[train_indices], dtype=torch.long)
    train_label = torch.tensor(train_valid_labels[train_indices], dtype=torch.long)
    train_meta = torch.tensor(train_valid_meta[train_indices], dtype=torch.long)

    # Validation Data in Tensor
    valid_input_ids = torch.tensor(train_valid_input_ids[valid_indices], dtype=torch.long)
    valid_input_mask = torch.tensor(train_valid_input_masks[valid_indices], dtype=torch.long)
    valid_segment_ids = torch.tensor(train_valid_segment_ids[valid_indices], dtype=torch.long)
    valid_label = torch.tensor(train_valid_labels[valid_indices], dtype=torch.long)
    valid_meta = torch.tensor(train_valid_meta[valid_indices], dtype=torch.long)

    # Load data into TensorDataset
    train = torch.utils.data.TensorDataset(train_input_ids, train_input_mask, train_segment_ids, train_meta, train_label)
    valid = torch.utils.data.TensorDataset(valid_input_ids, valid_input_mask, valid_segment_ids, valid_meta, valid_label)

    # Use DataLoader to load data from Dataset in batches
    train_loader = torch.utils.data.DataLoader(train, batch_size=batch_size, shuffle=True)
    valid_loader = torch.utils.data.DataLoader(valid, batch_size=batch_size, shuffle=False)

    bert_model = BertTextClassifier(bert_hidden_size, bert_dense_size, meta_size, 3, dropout=0.1)

    # Move model to GPU/CPU device
    device = 'cuda' if torch.cuda.is_available() else 'cpu'
    bert_model = bert_model.to(device)

    # Loss Function - use Cross Entropy as binary classification
    loss_fn = torch.nn.CrossEntropyLoss()
```

```
# Optimizer - Adam with parameter groups
param_optimizer = list(model.named_parameters())
no_decay = ['bias', 'LayerNorm.bias', 'LayerNorm.weight']
optimizer_grouped_parameters = [
    {'params': [p for n, p in param_optimizer if not any(nd in n for nd in no_decay)], 'weight_decay': 0.01},
    {'params': [p for n, p in param_optimizer if any(nd in n for nd in no_decay)], 'weight_decay': 0.0}]

optimizer = AdamW(optimizer_grouped_parameters, lr=learning_rate, eps=1e-6)

# Set Train Mode
bert_model.train()

# Initialize
best_f1 = 0.
valid_best = np.zeros((valid_label.size(0), 2))
early_stop = 0
train_losses = []
valid_losses = []

for epoch in range(num_epochs):
    logger.info('================     epoch {}        ==============='.format(epoch+1))
    train_loss = 0.
    for i, batch in tqdm(enumerate(train_loader), total=len(train_loader), desc='Training'):
        # Move batch data to device
        batch = tuple(t.to(device) for t in batch)
        # Bert input features and labels from batch
        x_ids, x_mask, x_sids, x_meta, y_truth = batch

        # Feedforward prediction
        y_pred = bert_model(x_ids, x_meta)

        # Calculate Loss
        loss = loss_fn(y_pred, y_truth)

        # Reset gradient
        optimizer.zero_grad()
        # Backward Propagation
        loss.backward()
        # Update Weights
        optimizer.step()
        # Training Loss
        train_loss += loss.item() / len(train_loader)

        logger.debug('train batch: %d, train_loss: %8f\n' % (i, train_loss))

    train_losses.append(train_loss)
    # Move to Evaluation Mode
    model.eval()

    # Initialize
    val_loss = 0.
    valid_preds_fold = np.zeros((valid_label.size(0), 3))

    with torch.no_grad():
```

```
        for i, batch in tqdm(enumerate(valid_loader), total=len(valid_loader), desc='Validation'):
            batch = tuple(t.to(device) for t in batch)
            x_ids, x_mask, x_sids, x_meta, y_truth = batch
            y_pred = bert_model(x_ids, x_meta).detach()
            loss = loss_fn(y_pred, y_truth)
            val_loss += loss.item() / len(valid_loader)
            valid_preds_fold[i * batch_size:(i + 1) * batch_size] = F.softmax(y_pred, dim=1).cpu().numpy()

            logger.debug('validation batch: {}, val_loss: {}, valid_preds_fold: {}'.format(i, val_loss, valid_preds_fold[i * batch_size:(i + 1)
        valid_losses.append(val_loss)

    # Calculate metrics
    acc, f1 = metric(train_valid_labels[valid_indices], np.argmax(valid_preds_fold, axis=1))

    # If improving, save the model. If not, count up for early stopping
    if best_f1 < f1:
        early_stop = 0
        best_f1 = f1
        valid_best = valid_preds_fold
        torch.save(bert_model.state_dict(), output_dir + 'model_fold_{}.dict'.format(fold))
    else:
        early_stop += 1

    logger.info(
        'epoch: %d, train loss: %.8f, valid loss: %.8f, acc: %.8f, f1: %.8f, best_f1: %.8f\n' %
        (epoch, train_loss, val_loss, acc, f1, best_f1))

    if device == 'cuda:0':
        torch.cuda.empty_cache()

    # Early stop if it reaches patience number
    if early_stop >= patience:
        break

    model.train()

# Once all epochs are done, take the best model of the fold
valid_preds_fold = np.zeros((valid_label.size(0), 3))

# Draw training/validation losses
sns.set(font_scale=1.5)
plt.rcParams["figure.figsize"] = (15,6)
plt.plot(train_losses, 'b-o')
plt.plot(valid_losses, 'b-o')
plt.title("Training/Validation Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.show()
plt.savefig(graph_dir + 'training_validation_loss_bert_epoch_{}_fold_{}'.format(epoch,fold) + '.png', )#bbox_inches='tight')

# Load the best model
bert_model.load_state_dict(torch.load(output_dir + 'model_fold_{}.dict'.format(fold)))
# Set Evaluation Mode
bert_model.eval()
```

```
    bert_model.eval()

    # Prediction on the validation set
    with torch.no_grad():
        for i, batch in tqdm(enumerate(valid_loader), total=len(valid_loader)):
            batch = tuple(t.to(device) for t in batch)
            x_ids, x_mask, x_sids, x_meta, y_truth = batch
            y_pred = bert_model(x_ids, x_meta).detach()
            valid_preds_fold[i * batch_size:(i + 1) * batch_size] = F.softmax(y_pred, dim=1).cpu().numpy()

    # Check the metrics for the validation set
    valid_best = valid_preds_fold
    oof_train[valid_indices] = valid_best
    acc, f1 = metric(train_valid_labels[valid_indices], np.argmax(valid_best, axis=1))
    logger.info('epoch: best, acc: %.8f, f1: %.8f, best_f1: %.8f\n' % (acc, f1, best_f1))

    class_names = ['Lower', 'Hold', 'Raise']
    titles_options = [("Confusion matrix, without normalization", None), ("Normalized confusion matrix", 'true')]
    for title, normalize in titles_options:
        disp = skplt.metrics.plot_confusion_matrix(train_valid_labels[valid_indices], np.argmax(valid_best, axis=1), normalize=normalize, title=tit
    plt.show()
    plt.savefig(graph_dir + 'conf_mats_bert_final.png')#bbox_inches='tight')
```

## Train Model

```
if use_skf:
    skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

    for fold, (train_indices, valid_indices) in enumerate(skf.split(train_valid_labels, train_valid_labels)):
        train_bert(fold, train_indices, valid_indices)

else:
    train_ratio = 0.7
    train_indices = np.arange(0, int(len(train_valid_labels)*train_ratio))
    valid_indices = np.arange(int(len(train_valid_labels)*train_ratio), len(train_valid_labels))

    train_bert(0, train_indices, valid_indices)
    # print('train_indices', train_indices)
    # print('valid_indices', valid_indices)
```

```
# Execute only when all folds have been performed
logger.info(f1_score(train_labels, np.argmax(oof_train, axis=1), average='macro'))
split_train_df['pred_target'] = np.argmax(oof_train, axis=1)
split_train_df['pred_target_lower'] = oof_train[:,0]
split_train_df['pred_target_hold'] = oof_train[:,1]
split_train_df['pred_target_raise'] = oof_train[:,2]
split_train_df.head()
```

```
[2021-01-25 14:51:20,151][INFO] ## 0.15943325041593778
```

| | target | prev_decision | GDP_diff_prev | PMI_value | Employ_diff_prev | Rsales_diff_year | Unemp_diff_prev | Inertia_diff | Hsales_diff_year | Balanced_diff | presconf_sc |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 1 | 1.043165 | 55.8 | 261.0 | 1.807631 | 0.0 | -0.015902 | 14.901418 | 0.035879 | |
| **1** | 1 | 1 | 1.043165 | 55.8 | 261.0 | 1.807631 | 0.0 | -0.015902 | 14.901418 | 0.035879 | |
| **2** | 1 | 1 | 1.043165 | 55.8 | 261.0 | 1.807631 | 0.0 | -0.015902 | 14.901418 | 0.035879 | |
| **3** | 1 | 1 | 1.043165 | 55.8 | 261.0 | 1.807631 | 0.0 | -0.015902 | 14.901418 | 0.035879 | |
| **4** | 1 | 1 | 1.043165 | 55.8 | 261.0 | 1.807631 | 0.0 | -0.015902 | 14.901418 | 0.035879 | |

## Save Data

```
if IN_COLAB:
  def save_data(df, file_name, dir_name=train_dir, index_csv=True):
    if not os.path.exists(dir_name):
      os.mkdir(dir_name)
    # Save results to a picke file
    file = open(dir_name + file_name + '.pickle', 'wb')
    pickle.dump(df, file)
    file.close()
    print('Successfully saved {}.pickle. in {}'.format(file_name, dir_name + file_name + '.pickle'))
    # Save results to a csv file
    df.to_csv(dir_name + file_name + '.csv', index=index_csv)
    print('Successfully saved {}.csv. in {}'.format(file_name, dir_name + file_name + '.csv'))

else:
  def save_data(df, file_name, dir_name=train_dir, index_csv=True):
    # Save results to a .picke file
    file = open(dir_name + file_name + '.pickle', 'wb')
```

```
    pickle.dump(df, file)
    file.close()
    print('Successfully saved {}.pickle. in {}'.format(file_name, dir_name + file_name + '.pickle'))
    # Save results to a .csv file
    df.to_csv(dir_name + file_name + '.csv', index=index_csv)
    print('Successfully saved {}.csv. in {}'.format(file_name, dir_name + file_name + '.csv'))
```

```
# Save text data (very large files)
save_data(train_df, 'train_df')
save_data(text_df, 'text_df')
save_data(split_train_df, 'split_train_df')
```

```
Successfully saved train_df.pickle. in /content/drive/My Drive/Colab Notebooks/proj2/src/data/train_data/train_df.pickle
Successfully saved train_df.csv. in /content/drive/My Drive/Colab Notebooks/proj2/src/data/train_data/train_df.csv
Successfully saved text_df.pickle. in /content/drive/My Drive/Colab Notebooks/proj2/src/data/train_data/text_df.pickle
Successfully saved text_df.csv. in /content/drive/My Drive/Colab Notebooks/proj2/src/data/train_data/text_df.csv
Successfully saved split_train_df.pickle. in /content/drive/My Drive/Colab Notebooks/proj2/src/data/train_data/split_train_df.pickle
Successfully saved split_train_df.csv. in /content/drive/My Drive/Colab Notebooks/proj2/src/data/train_data/split_train_df.csv
```