DAFTAR PUSTAKA

A. Graves, J. N. (2013). Hybrid Speech Recognition With Deep Bidirectional LSTM. *Department of Computer Science 6 King's College Rd. Toronto, M5S 3G4, Canada*, 1-6.

Agustriana. (2018). Perancangan sisetem informasi penjualan pakaian bebasis web pada annoteos shop. *Program Studi Sistem Informasi (STIMIK) GICI Batam*, 1-89.

Ahmad, A. (2017). Prediksi Kunjungan Wisatawan Dengan Reccurent Neural Network Extended Kalman Filter. *Program Studi Informatika, STIMIK Bumigora Mataram*, 1-12.

Alpaydin, E. (2010). *Intoductuion to machone learning, second edition.* London, England: Massachusetts Institute of Technology.

Choi, K. d. (2017). Convotional recurrent neural networks for music classification. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1-5.

Colah. (2015, August Thusday). *Understanding LSTM Networks*. Retrieved from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Danukusumo, K. (2017). Implementasi Deep Learning Menggunakan Convolutional Neural Network Untuk Klasifikasi Citra Candi Berbasis GpU. *Fakultas Teknoloi Industri Tekni Informatika UAJY*, 1-79.

Fausett. (1994). *Fundamentals of newral networks architectures, algoritma and application.* Upper Sadle River, New Jersey: Prentice-Hall, Inc., River, NJ.

Han, H., Wang, W. Y., & Mao, B. H. (2005, August). Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning. *International conference on intelligent computing*, 878-887.

Han, J. (2011). *Data Mining Concepts and Techniques Third.* 225 Wyman Street, Waltham,: Morgan Kaufmann Publishers is an imprint of Elsevier.

Kazybek Adam, K. d. (2019). Memristive LSTM network hardware architecture for time series predictive modeling problem. *Departemen of elecrtival and computer engineering nazarbayev*, 50.

Li, L. d. (2018). Classification Based on Word2vec and Convolutional Neural Network. *International Conference on Neural Information*, 450-460.

Lila Dini, H. (2018). Komparasi Algoritma Klasifikasi Pada Analisis Review Hotel. *Ilmu Komputer STIMIK Nusa Mandiri Jakarta*, 1-6.

Luthfi, M. A. (2020). Prediksi data transaksi penjualan time series menggunakan regresi LSTM. *Jurusan Teknik Informatika, Fakultas Ilmu Komputer,Universitas Mercu Buana*, 1-10.

Manaswi, N. K. (2018). *Deep Learning With Applications Using Phyton.* Bangalore, Karnataka, India: Apress.

Moreo Alejandro, A. (2019). Word class embeddings for multiclass text classification. *Istituto di Scienza e Tecnologie dell'Informazione Consiglio Nazionale delle Ricerche*, 1-29.

Muhammad Rizky Setio Basuki, Y. A. (2020). Implementasi Deep Learning Menggunakan Arsitektur Long. *Teknik Informatika Universitas Muhammadiyah Malang*, 1-8.

Prechelt, L. (2000). Early Stopping- But When? *Fakultas fur Informatik; Universitas Karlsruhe*, 2-16.

Satyo, A. (2020). Analisis Data Time Series Menggunakan LSTM (Long Short Term Memory) Dan ARIMA (Autocorrelation Integrated Moving Average) Dalam Bahasa Python. *Universitas Gunadarma*, 1-7.

Wikipedia. (2020, Februaru 1). Kecerdasan Buatan. Retrieved from Jaringan. *https://id.wikipedia.org/wiki/Jaringan_saraf_tiruan*.

Winda, R. F. (2020). Multi-label text classification in news article usding long-term memory with word2vec. *Departemen informatika Universitas Sriwijaya*, 1-11.

Wira, G. J. (2004). *Pengenalan Konsep Pembelajaran Machine Learning Edisi 2,4.* Tokyo Japan: Inc., San Francisco, CA, USA.

Yan, Y., Wang, Y., Gao W. (2018). LSTM2: Multi Label ranking for document classification. *Neural Processing later*, 47.

LAMPIRAN 1

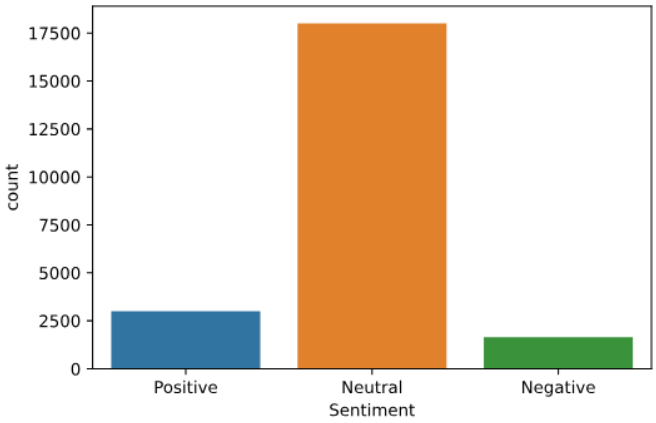
Multiclass

A. Banyak kelas Y pada data asli

Neutral : 18.004

Positive : 2.991

Negative : 1.646

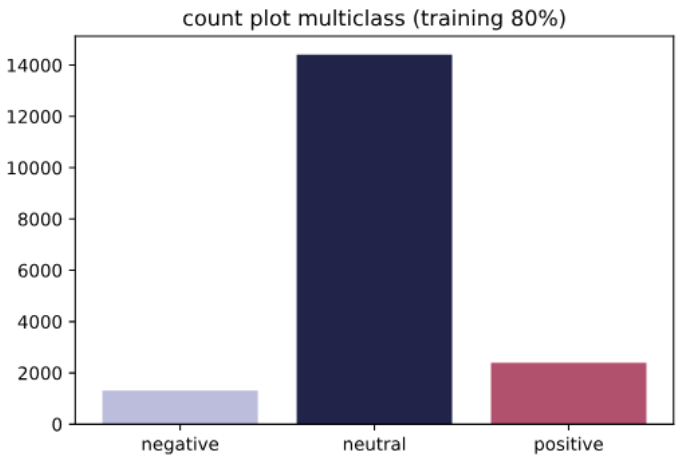


B. Banyak 3 kelas Y pada data training 80%

Neutral : 14.402

Positive : 2.393

Negative : 1.317

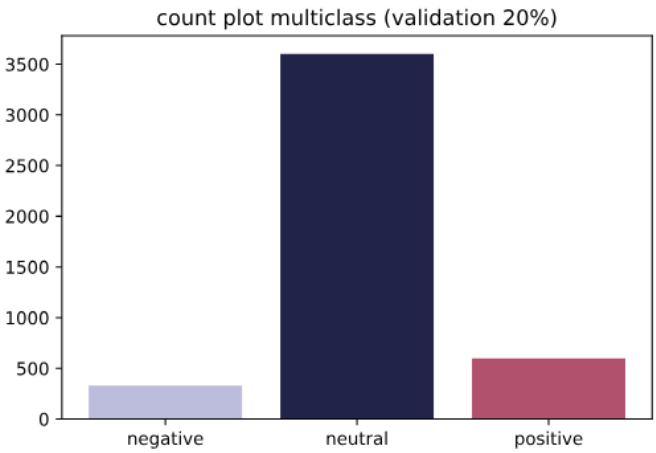


C. Banyak 3 kelas Y pada data Testing 20%

Neutral : 3.602

Positive : 598

Negative : 329

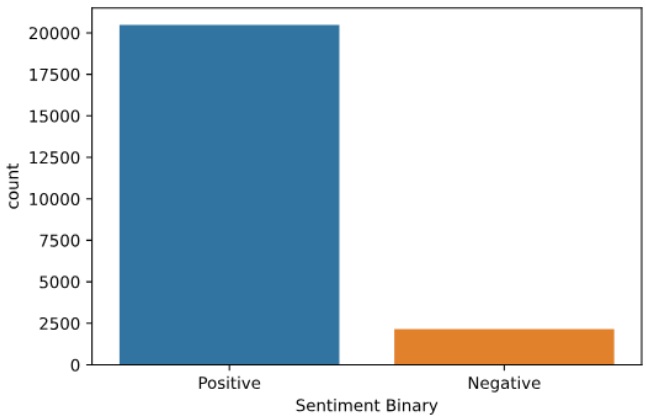


#Binary

A. Banyak y Binary pada data asli

Positive : 20.482

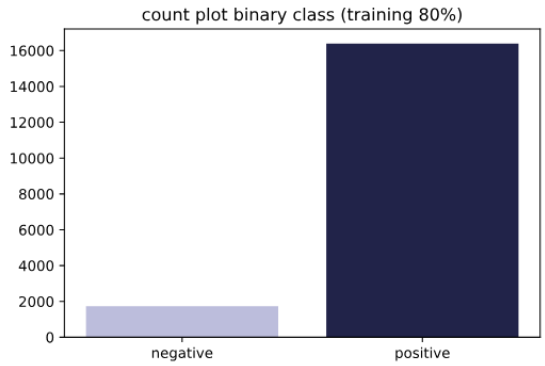
Negative : 2.159



B. Banyak kelas Y Binary pada data Training 80%

Negative : 1.727

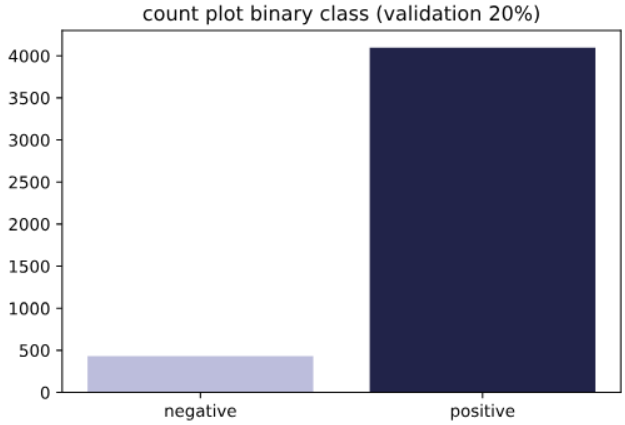
Positive : 16.385



C. Banyak kelas Y Binary pada data Training 80%

Negative : 432

Positive : 4.097



Klasifikasi Multi Kelas

 # Vanilla Training Result

Accuracy	Recall Negative	Recall Neutral	Recall Positive	Precision Negative	Precision Neutral	Precision Positive	F1 Negative	F1 Neutral	F1 Positive
0.841652	0.520881	0.970629	0.715838	0.775141	0.915096	0.878011	0.62307	0.942045	0.788674
0.894379	0.633257	0.986738	0.765566	0.916484	0.932236	0.935649	0.74899	0.958713	0.842105
0.922151	0.262718	0.974656	0.944421	0.961111	0.927698	0.862266	0.412642	0.950598	0.901476
0.942359	0.832954	0.984655	0.904304	0.953913	0.969906	0.924391	0.889339	0.977225	0.914237
0.957045	0.839787	0.994515	0.914333	0.983111	0.972171	0.970719	0.905815	0.983216	0.941683
0.965824	0.965831	0.977989	0.968659	0.915108	0.991971	0.918748	0.939786	0.984931	0.943043
0.972008	0.871678	0.994306	0.968659	0.993939	0.983584	0.966639	0.928803	0.988916	0.967648
0.979903	0.946849	0.993057	0.979106	0.995211	0.992023	0.959459	0.970428	0.99254	0.969183
0.982111	0.965831	0.991251	0.988299	0.997647	0.995468	0.947516	0.981481	0.993355	0.967478
0.982277	0.94609	0.994723	0.990389	0.997598	0.993757	0.968533	0.971161	0.99424	0.979339

 # Vanilla Validation Result

Accuracy	Recall Negative	Recall Neutral	Recall Positive	Precision Negative	Precision Neutral	Precision Positive	F1 Negative	F1 Neutral	Accuracy
0.868845	0.364742	0.958356	0.607023	0.621762	0.887404	0.813901	0.45977	0.921516	0.695402
0.873924	0.316109	0.96724	0.618729	0.684211	0.885613	0.835214	0.432432	0.924628	0.710855
0.865754	0.112462	0.945863	0.797659	0.787234	0.893054	0.715142	0.196809	0.9187	0.75415
0.870612	0.392097	0.937812	0.729097	0.635468	0.904418	0.737733	0.484962	0.920812	0.733389
0.875469	0.367781	0.950583	0.702341	0.650538	0.900105	0.779221	0.469903	0.924656	0.738786
0.856701	0.528875	0.89839	0.785953	0.491525	0.921412	0.708899	0.509517	0.909755	0.74544
0.872378	0.322188	0.945863	0.732441	0.666667	0.899894	0.75	0.434426	0.922306	0.741117
0.872157	0.395137	0.936424	0.747492	0.631068	0.907208	0.738843	0.485981	0.921585	0.743142
0.865754	0.428571	0.922265	0.765886	0.585062	0.912136	0.708978	0.494737	0.917173	0.736334
0.86752	0.370821	0.929206	0.769231	0.642105	0.908523	0.70229	0.470135	0.918748	0.734238

 # Bidirectional Training Result

Accuracy	Recall Negative	Recall Neutral	Recall Positive	Precision Negative	Precision Neutral	Precision Positive	F1 Negative	F1 Neutral	Accuracy
0.868845	0.364742	0.958356	0.607023	0.621762	0.887404	0.813901	0.45977	0.921516	0.695402
0.873924	0.316109	0.96724	0.618729	0.684211	0.885613	0.835214	0.432432	0.924628	0.710855
0.865754	0.112462	0.945863	0.797659	0.787234	0.893054	0.715142	0.196809	0.9187	0.75415
0.870612	0.392097	0.937812	0.729097	0.635468	0.904418	0.737733	0.484962	0.920812	0.733389
0.875469	0.367781	0.950583	0.702341	0.650538	0.900105	0.779221	0.469903	0.924656	0.738786
0.856701	0.528875	0.89839	0.785953	0.491525	0.921412	0.708899	0.509517	0.909755	0.74544
0.872378	0.322188	0.945863	0.732441	0.666667	0.899894	0.75	0.434426	0.922306	0.741117
0.872157	0.395137	0.936424	0.747492	0.631068	0.907208	0.738843	0.485981	0.921585	0.743142
0.865754	0.428571	0.922265	0.765886	0.585062	0.912136	0.708978	0.494737	0.917173	0.736334
0.86752	0.370821	0.929206	0.769231	0.642105	0.908523	0.70229	0.470135	0.918748	0.734238

 # Bidirectional Validation Result

Accuracy	Recall Negative	Recall Neutral	Recall Positive	Precision Negative	Precision Neutral	Precision Positive	F1 Negative	F1 Neutral	Accuracy
0.857364	0.00304	0.988062	0.540134	1	0.855735	0.875339	0.006061	0.91715	0.668046
0.869949	0.167173	0.975847	0.618729	0.679012	0.875903	0.850575	0.268293	0.923178	0.71636
0.871274	0.288754	0.936979	0.795987	0.725191	0.905797	0.708333	0.413043	0.921124	0.749606
0.87282	0.431611	0.935869	0.735786	0.617391	0.908136	0.749574	0.50805	0.921794	0.742616
0.877898	0.431611	0.951138	0.682274	0.648402	0.902291	0.795322	0.518248	0.926071	0.734473
0.867741	0.495441	0.918934	0.764214	0.543333	0.916644	0.739482	0.518283	0.917787	0.751645
0.864871	0.544073	0.920877	0.704013	0.518841	0.911765	0.771062	0.531157	0.916298	0.736014
0.872599	0.367781	0.941699	0.734114	0.664835	0.90381	0.739057	0.473581	0.922366	0.736577
0.869728	0.468085	0.925597	0.754181	0.587786	0.913174	0.732143	0.521151	0.919344	0.742998
0.864871	0.458967	0.916713	0.77592	0.626556	0.914681	0.684366	0.529825	0.915696	0.727273

A. Klasifikasi Biner

Vanilla Train Result

Accuracy	Recall Negative	Recall Positive	Precision Negative	Precision Positive	F1 Negative	F1 Positif
0.909397	0.408801	0.994446	0.885822	0.941034	0.559429	0.967003
0.942414	0.745802	0.992188	0.909605	0.973706	0.819599	0.98286
0.963118	0.884192	0.990845	0.910555	0.987831	0.89718	0.989336
0.978136	0.957151	0.989503	0.905753	0.995456	0.930743	0.992471
0.985534	0.952519	0.99762	0.976841	0.995009	0.964527	0.996312
0.989675	0.976259	0.997925	0.980233	0.997499	0.978242	0.997712
0.993209	0.987261	0.996399	0.966553	0.998654	0.976797	0.997525
0.991608	0.984366	0.998657	0.987224	0.998353	0.985793	0.998505
0.995749	0.997684	0.994324	0.948789	0.999755	0.972622	0.997032
0.997626	0.997684	0.998657	0.987393	0.999756	0.992512	0.999206

Vanilla Validation Result

Accuracy	Recall Negative	Recall Positive	Precision Negative	Precision Positive	F1 Negative	F1 Positif
0.920512	0.263889	0.989749	0.730769	0.927281	0.387755	0.957497
0.92272	0.409722	0.976812	0.650735	0.940099	0.502841	0.958104
0.923383	0.50463	0.967537	0.621083	0.948779	0.556833	0.958066
0.910797	0.5625	0.947523	0.530568	0.953574	0.546067	0.950539
0.924928	0.483796	0.971443	0.641104	0.946943	0.551451	0.959036
0.921616	0.527778	0.963144	0.601583	0.950843	0.562269	0.956954
0.917863	0.506944	0.961191	0.579365	0.948687	0.540741	0.954898
0.919629	0.476852	0.966317	0.598837	0.945998	0.530928	0.956049
0.909031	0.590278	0.942641	0.520408	0.956177	0.553145	0.949361
0.919629	0.548611	0.95875	0.583744	0.952704	0.565632	0.955718

Bidirectional Train Result

Accuracy	Recall Negative	Recall Positive	Precision Negative	Precision Positive	F1 Negative	F1 Positif
0.90487	0.233353	0.99646	0.874187	0.92499	0.368373	0.959396
0.935181	0.661841	0.989258	0.866566	0.965224	0.750492	0.977093
0.954781	0.828025	0.991395	0.910248	0.982045	0.867192	0.986697
0.968419	0.869716	0.994934	0.947634	0.986386	0.907005	0.990642
0.974934	0.92183	0.995179	0.952723	0.991789	0.937022	0.993481
0.981062	0.944991	0.995362	0.955504	0.994209	0.950218	0.994785
0.986639	0.971048	0.996094	0.96324	0.996946	0.967128	0.99652
0.985314	0.9641	0.997986	0.980565	0.996223	0.972263	0.997104
0.98824	0.984366	0.99585	0.961538	0.998348	0.972818	0.997097
0.991442	0.966416	0.99884	0.988744	0.996469	0.977452	0.997653

Bidirectional Validation Result

Accuracy	Recall Negative	Recall Positive	Precision Negative	Precision Positive	F1 Negative	F1 Positif
0.913668	0.150463	0.994142	0.730337	0.917342	0.24952	0.954199
0.919187	0.384259	0.975592	0.62406	0.937603	0.475645	0.95622
0.920954	0.435185	0.972175	0.622517	0.942276	0.512262	0.956992
0.920954	0.439815	0.971687	0.620915	0.942695	0.514905	0.956971
0.921175	0.460648	0.969734	0.616099	0.944603	0.527152	0.957003
0.912122	0.469907	0.95875	0.545699	0.944912	0.504975	0.951781
0.912564	0.502315	0.955821	0.545226	0.947954	0.522892	0.951872
0.917421	0.458333	0.965829	0.585799	0.944166	0.514286	0.954875
0.909031	0.560185	0.945814	0.521552	0.95326	0.540179	0.949522
0.922499	0.409722	0.976568	0.648352	0.940085	0.502128	0.957979

LAMPIRAN 2

1. Word Could Unigram

A. Word cloud Unigram Sentiment positive

B. Word cloud Unigram Sentiment Neutral

C. Word cloud Unigram Sentiment Negative

2. Word Could Trigram

A. Word Could Trigram Sentiment Positive

B. Word Could Trigram Sentiment Neutral

C. Word Could Trigram Sentiment Negative

LAMPIRAN 3

```
# Library yang digunakan

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import string
%matplotlib inline
import statsmodels.api as sm
import re
from itertools import islice
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matri
x, plot_confusion_matrix, precision_recall_fscore_support as score
from sklearn.utils import class_weight

import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Embedding, Flatten, Dense, Sim
pleRNN, LSTM, Bidirectional, Dropout
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import Callback, ModelCheckpoint
import nltk
import nltk.classify.util
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, RegexpTokenizer
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk.sentiment.util import *
from nltk.stem.lancaster import LancasterStemmer
from nltk.stem.porter import PorterStemmer
from wordcloud import WordCloud, STOPWORDS

seed = 101
```

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

# Membaca data
df = pd.read_csv('reviews.csv', index_col=0)
# menampilkan data
df.head()
#menampilkan info data
df.info()

# Menampilkan plot rating
sns.countplot(df.Rating)
plt.xlabel('Rating')
plt.title('Number of individual rating')

# Menampilkan data sebelum missing value
print('total rows:',len(df))
df.isna().sum()

# drop missing values
df = df.dropna(subset=['Review Text'])

#df = df[df['Rating'] != 3]
df.reset_index(inplace=True, drop=True)

# Menampilkan data setelah missing value
print('total rows:',len(df))
df.isna().sum()

# Featur Elemination
df = df.drop(['Clothing ID', 'Age', 'Title', 'Recommended IND', 'P
ositive Feedback Count', 'Division Name', 'Department Name', 'Clas
s Name'], axis=1)

# tokenized but joined with spaces
```

```
def string_unlist(strlist):
    return " ".join(strlist)
tokenizer = RegexpTokenizer(r'[a-zA-Z]{3,}')
stop_words = set(stopwords.words('english')) - set(['not', 'but',
'don\'t', 'dont'])

df['tokenized'] = df['Review Text'].astype(str).str.lower()

# Turn into lower case text
df['tokenized'] = df.apply(lambda row: tokenizer.tokenize(row['tok
enized']), axis=1)

# Apply tokenize to each row
df['tokenized'] = df['tokenized'].apply(lambda x: [w for w in x if
 not w in stop_words])

# Remove stopwords from each row
df["tokenized_unlist"] = df["tokenized"].apply(string_unlist)


samples = df['tokenized']
maxlen = 100
max_words = 10000
tokenizer2 = Tokenizer(num_words=max_words)
tokenizer2.fit_on_texts(samples)
sequences = tokenizer2.texts_to_sequences(samples)


word_index = tokenizer2.word_index
print('Found %s unique tokens.' % len(word_index))
padded = pad_sequences(sequences, maxlen=maxlen)
#df['preprocessed'] = padded

# Melakukan Preprocessing
df['preprocessed'] = ''*len(df)
for i in range(len(df)):
```

```
    df.at[i,'preprocessed'] = padded[i]

# original text
df['Review Text'].loc[3]

# after lower case
df['Review Text'].astype(str).str.lower()[3]

# after tokenization
tokenizer.tokenize(df['Review Text'].astype(str).str.lower()[3])

# after removing stop words
df['tokenized'].loc[3]

# after text to sequence
sequences[3]

# after padding (length 100)
padded[3]


def take(n, iterable):
    "Return first n items of the iterable as a list"
    return list(islice(iterable, n))




# word index
take(50, word_index)


i = 0
for key, value in word_index.items():
    if i==len(word_index.items()) - 1:
        print(key, value)
    i = i +1
```

```
# Konversi Rating menjadi sentimen
df['Sentiment']=''
df.loc[df['Rating']==5,'Sentiment'] = 'Positive'
df.loc[df['Rating']==4,'Sentiment'] = 'Positive'

#df.loc[df['Rating']==3,'Sentiment'] = 'Negative'

df.loc[df['Rating']==2,'Sentiment'] = 'Negative'
df.loc[df['Rating']==1,'Sentiment'] = 'Negative'

# nltk sentiment analyzer module
SIA = SentimentIntensityAnalyzer()

# get polarity score
df['Polarity Score']=df["tokenized_unlist"].apply(lambda x:SIA.pol
arity_scores(x)['compound'])
df['Neutral Score']=df["tokenized_unlist"].apply(lambda x:SIA.pola
rity_scores(x)['neu'])
df['Negative Score']=df["tokenized_unlist"].apply(lambda x:SIA.pol
arity_scores(x)['neg'])
df['Positive Score']=df["tokenized_unlist"].apply(lambda x:SIA.pol
arity_scores(x)['pos'])

df['Sentiment']=''
df.loc[(df['Positive Score']>df['Neutral Score']) & (df['Positive
Score']>df['Negative Score']),'Sentiment' ]='Positive'
df.loc[(df['Neutral Score']>df['Positive Score']) & (df['Neutral S
core']>df['Negative Score']),'Sentiment' ]='Neutral'
df.loc[(df['Negative Score']>df['Positive Score']) & (df['Negative
 Score']>df['Neutral Score']),'Sentiment' ]='Negative'

conditions = [
    df['Sentiment'] == "Positive",
    df['Sentiment'] == "Negative",
    df['Sentiment'] == "Neutral"
```

```
        ]
    choices = [2,0,1]
    df['label'] = np.select(conditions, choices)
    df.head(10)


rint(df['Sentiment'].value_counts())
sns.countplot(df['Sentiment'])

len(df[df['Polarity Score']<=0])

len(df[df['Sentiment']=='Negative'])

np.unique(df['Sentiment'])

df.loc[df['Sentiment']=='Negative'].head(3)

df.head(10).to_excel('positive_neutral.xlsx')
df.loc[df['Sentiment']=='Negative'].head(3).to_excel('negative.xlsx')

TRAIN TEST Plist
X = padded
y = df['label']
y = to_categorical(y)
# lstm
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.20
, stratify=y, random_state=seed)


MODELING
# callback to find metrics at epoch end
class Metrics(Callback):
    def __init__(self, x, y):
        self.x = x
        self.y = y if (y.ndim == 1 or y.shape[1] == 1) else np.argmax(
y, axis=1)
        self.reports = []
self.x = x
        self.y = y if (y.ndim == 1 or y.shape[1] == 1) else np.argmax(
y, axis=1)
        self.reports = []
        self.confusion_matrices = []
```

```
    def on_epoch_end(self, epoch, logs={}):
        y_hat = np.asarray(self.model.predict(self.x))
        y_hat = np.where(y_hat > 0.5, 1, 0) if (y_hat.ndim == 1 or y_h
at.shape[1] == 1)  else np.argmax(y_hat, axis=1)
        report = classification_report(self.y,y_hat,output_dict=True)
        self.reports.append(report)

        _confusion_matrix = confusion_matrix(self.y,y_hat)
        self.confusion_matrices.append(_confusion_matrix)
        return

    # Utility method
    def get(self, metrics, of_class):
        return [report[str(of_class)][metrics] for report in self.repo
rts]

    def get_cm(self):
        return self.confusion_matrices

class_weights = class_weight.compute_class_weight('balanced', np.uniqu
e(df['label']), y=df['label'])
class_weights = dict(enumerate(class_weights))
#class_weights = {0: 2., 1: 2., 2: 1.}

print(class_weights)
#class_weights[0] /=2
#class_weights[1] /=2
#print(class_weights)

EPOCHS = 10
BATCH_SIZE = 32

VANILLA LSTM
def baseline_model():
    model = Sequential()
    model.add(Embedding(max_words, 100, input_length=maxlen)) # input
layer
    model.add(LSTM(100)) # hidden layer
    model.add(Dropout(rate=0.5))
    model.add(Dense(3, activation='softmax'))
    model.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['acc'])
    return model

vanilla_metrics_train = Metrics(X_train, y_train)
```

```
vanilla_metrics_val = Metrics(X_val, y_val)
checkpoint1 = ModelCheckpoint('vanilla/checkpoint-
{epoch:02d}', monitor='val_loss', verbose=0, save_best_only=False, sav
e_weights_only=False, mode='auto', save_freq='epoch')
model1 = baseline_model()
model1.summary()

history = model1.fit(X_train, y_train, epochs=EPOCHS, batch_size=BATCH
_SIZE, validation_data=(X_val, y_val),#callbacks=[vanilla_metrics_trai
n, vanilla_metrics_val, checkpoint1], class_weight=class_weights)
#callbacks=[vanilla_metrics_train, vanilla_metrics_val], class_weight=
class_weights)
callbacks=[vanilla_metrics_train, vanilla_metrics_val]

vanilla_train_result = pd.DataFrame([history.history['acc'], vanilla_m
etrics_train.get('recall',0), vanilla_metrics_train.get('recall',1), v
anilla_metrics_train.get('recall',2),
vanilla_metrics_train.get('precision',0), vanilla_metrics_train.get('p
recision',1), vanilla_metrics_train.get('precision',2),
vanilla_metrics_train.get('f1-
score',0), vanilla_metrics_train.get('f1-
score',1), vanilla_metrics_train.get('f1-
score',2)],index=['accuracy','recall negative', 'recall neutral', 'rec
all positive', 'precision negative', 'precision neutral', 'precision p
ositive', 'f1 negative', 'f1 neutral', 'f1 positive']).T
vanilla_train_result.to_excel('vanilla_train_result.xlsx')
vanilla_train_result

vanilla_val_result = pd.DataFrame([history.history['val_acc'], vanilla
_metrics_val.get('recall',0), vanilla_metrics_val.get('recall',1), van
illa_metrics_val.get('recall',2),
vanilla_metrics_val.get('precision',0), vanilla_metrics_val.get('preci
sion',1), vanilla_metrics_val.get('precision',2),
vanilla_metrics_val.get('f1-score',0), vanilla_metrics_val.get('f1-
score',1), vanilla_metrics_val.get('f1-
score',2)],index=['accuracy','recall negative', 'recall neutral', 'rec
all positive', 'precision negative', 'precision neutral', 'precision p
ositive', 'f1 negative', 'f1 neutral', 'f1 positive']).T
vanilla_val_result.to_excel('vanilla_val_result.xlsx')
vanilla_val_result

# accuracy plot
fig, ax = plt.subplots(figsize=(8,6))
plt.plot(range(1, EPOCHS+1), history.history['acc'], color='#9E9AC8')
```

```
plt.plot(range(1, EPOCHS+1), history.history['val_acc'], color='#21234
9')

plt.title('Vanilla LSTM accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch', labelpad=2)
plt.xticks(np.arange(1, EPOCHS+1))
plt.legend(['train', 'val'], loc='upper left')
plt.grid(True)
plt.show()

BIDIRECTIONAL

def baseline_model2():
    model = Sequential()
    model.add(Embedding(max_words, 100, input_length=maxlen))
    model.add(Dropout(rate=0.33))
    model.add(Bidirectional(LSTM(units = 20, return_sequences= True)))
    model.add(Dropout(rate=0.5))
    model.add(Bidirectional(LSTM(units = 20, return_sequences= False))
)
    model.add(Dropout(rate=0.5))
    model.add(Dense(3, activation='softmax'))
    model.compile(optimizer='adam', loss='categorical_crossentropy', m
etrics=['acc'])
    return model

bidirectional_metrics_train = Metrics(X_train, y_train)
bidirectional_metrics_val = Metrics(X_val, y_val)
checkpoint2 = ModelCheckpoint('bidirectional/checkpoint-
{epoch:02d}', monitor='val_loss', verbose=0, save_best_only=False, sav
e_weights_only=False, mode='auto', save_freq='epoch')
model2 = baseline_model2()
model2.summary()

history2 = model2.fit(X_train, y_train, epochs=EPOCHS, batch_size=BATC
H_SIZE, validation_data=(X_val, y_val), #callbacks=[bidirectional_metr
ics_train, bidirectional_metrics_val, checkpoint2], class_weight=class
_weights )
#callbacks=[bidirectional_metrics_train, bidirectional_metrics_val], c
lass_weight=class_weights
callbacks=[bidirectional_metrics_train, bidirectional_metrics_val]
)
```

```
bidirectional_train_result = pd.DataFrame([history2.history['acc'], bi
directional_metrics_train.get('recall',0), bidirectional_metrics_train
.get('recall',1), bidirectional_metrics_train.get('recall',2),
bidirectional_metrics_train.get('precision',0), bidirectional_metrics_
train.get('precision',1), bidirectional_metrics_train.get('precision',
2),
bidirectional_metrics_train.get('f1-
score',0), bidirectional_metrics_train.get('f1-
score',1), bidirectional_metrics_train.get('f1-
score',2)],index=['accuracy','recall negative', 'recall neutral', 'rec
all positive', 'precision negative', 'precision neutral', 'precision p
ositive', 'f1 negative', 'f1 neutral', 'f1 positive']).T
bidirectional_train_result.to_excel('bidirectional_train_result.xlsx')
bidirectional_train_result

bidirectional_val_result = pd.DataFrame([history2.history['val_acc'],
bidirectional_metrics_val.get('recall',0), bidirectional_metrics_val.g
et('recall',1), bidirectional_metrics_val.get('recall',2),
bidirectional_metrics_val.get('precision',0), bidirectional_metrics_va
l.get('precision',1), bidirectional_metrics_val.get('precision',2),
bidirectional_metrics_val.get('f1-
score',0), bidirectional_metrics_val.get('f1-
score',1), bidirectional_metrics_val.get('f1-
score',2)],index=['accuracy','recall negative', 'recall neutral', 'rec
all positive', 'precision negative', 'precision neutral', 'precision p
ositive', 'f1 negative', 'f1 neutral', 'f1 positive']).T
bidirectional_val_result.to_excel
('bidirectional_val_result.xlsx')
bidirectional_val_result

# accuracy plot
fig, ax = plt.subplots(figsize=(8,6))
plt.plot(range(1, EPOCHS+1), history2.history['acc'], color='#9E9AC8')
plt.plot(range(1, EPOCHS+1), history2.history['val_acc'], color='#2123
49')

plt.title('Bidirectional LSTM accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch', labelpad=2)
plt.xticks(np.arange(1, EPOCHS+1))
plt.legend(['train', 'val'], loc='upper left')
plt.grid(True)
plt.show()

COMPARISON
```

```
# training plot
fig, ax = plt.subplots(figsize=(8,6))
plt.plot(range(1, EPOCHS+1), history.history['acc'], color='#9E9AC8')
plt.plot(range(1, EPOCHS+1), history2.history['acc'], color='#212349')

plt.title('Training Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch', labelpad=2)
plt.xticks(np.arange(1, EPOCHS+1))
plt.legend(['Vanilla', 'Bidirectional'], loc='upper left')
plt.grid(True)
plt.show()

# validation plot
fig, ax = plt.subplots(figsize=(8,6))
plt.plot(range(1, EPOCHS+1), history.history['val_acc'], color='#9E9AC
8')
plt.plot(range(1, EPOCHS+1), history2.history['val_acc'], color='#2123
49')

plt.title('Validation Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch', labelpad=2)
plt.xticks(np.arange(1, EPOCHS+1))
plt.legend(['Vanilla', 'Bidirectional'], loc='lower right')
plt.grid(True)
plt.show()

# precision plot
fig, ax = plt.subplots(figsize=(8,6))
plt.plot(range(1, EPOCHS+1), vanilla_val_result['precision negative'],
 color='#fdb462')
plt.plot(range(1, EPOCHS+1), vanilla_val_result['precision neutral'],
color='#af8dc3')
plt.plot(range(1, EPOCHS+1), vanilla_val_result['precision positive'],
 color='#f03b20')
plt.title('Vanilla Precision (Validation)')
plt.ylabel('Precision')
plt.xlabel('epoch', labelpad=2)
plt.xticks(np.arange(1, EPOCHS+1))
plt.legend(['precision negative', 'precision neutral', 'precision posi
tive'], loc='lower left')
plt.grid(True)
plt.show()
```

```
# recall plot
fig, ax = plt.subplots(figsize=(8,6))
plt.plot(range(1, EPOCHS+1), vanilla_train_result['recall negative'],
color='#fdb462')
plt.plot(range(1, EPOCHS+1), vanilla_train_result['recall neutral'], c
olor='#af8dc3')
plt.plot(range(1, EPOCHS+1), vanilla_train_result['recall positive'],
color='#f03b20')
plt.title('Vanilla Recall (Training)')
plt.ylabel('Recall')
plt.xlabel('epoch', labelpad=2)
plt.xticks(np.arange(1, EPOCHS+1))
plt.legend(['recall negative', 'recall neutral', 'recall positive'], l
oc='lower left')
plt.grid(True)
plt.show()

# recall plot
fig, ax = plt.subplots(figsize=(8,6))
plt.plot(range(1, EPOCHS+1), vanilla_val_result['recall negative'], co
lor='#fdb462')
plt.plot(range(1, EPOCHS+1), vanilla_val_result['recall neutral'], col
or='#af8dc3')
plt.plot(range(1, EPOCHS+1), vanilla_val_result['recall positive'], co
lor='#f03b20')
plt.title('Vanilla Recall (Validation)')
plt.ylabel('Recall')
plt.xlabel('epoch', labelpad=2)
plt.xticks(np.arange(1, EPOCHS+1))
plt.legend(['recall negative', 'recall neutral', 'recall positive'], l
oc='lower left')
plt.grid(True)
plt.show()

# F1 Training plot
fig, ax = plt.subplots(figsize=(8,6))
plt.plot(range(1, EPOCHS+1), vanilla_train_result['f1 negative'], colo
r='#fdb462')
plt.plot(range(1, EPOCHS+1), vanilla_train_result['f1 neutral'], color
='#af8dc3')
plt.plot(range(1, EPOCHS+1), vanilla_train_result['f1 positive'], colo
r='#f03b20')
plt.title('Vanilla f1 (Training)')
plt.ylabel('f1')
```

```
plt.xlabel('epoch', labelpad=2)
plt.xticks(np.arange(1, EPOCHS+1))
plt.legend(['f1 negative', 'f1 neutral', 'f1 positive'], loc='lower le
ft')
plt.grid(True)
plt.show()

# F1 Testing plot
fig, ax = plt.subplots(figsize=(8,6))
plt.plot(range(1, EPOCHS+1), vanilla_val_result['f1 negative'], color=
'#fdb462')
plt.plot(range(1, EPOCHS+1), vanilla_val_result['f1 neutral'], color='
#af8dc3')
plt.plot(range(1, EPOCHS+1), vanilla_val_result['f1 positive'], color=
'#f03b20')
plt.title('Vanilla f1 (Validation)')
plt.ylabel('f1')
plt.xlabel('epoch', labelpad=2)
plt.xticks(np.arange(1, EPOCHS+1))
plt.legend(['f1 negative', 'f1 neutral', 'f1 positive'], loc='lower le
ft')
plt.grid(True)
plt.show()

# precision plot
fig, ax = plt.subplots(figsize=(8,6))
plt.plot(range(1, EPOCHS+1), bidirectional_train_result['precision neg
ative'], color='#fdb462')
plt.plot(range(1, EPOCHS+1), bidirectional_train_result['precision neu
tral'], color='#af8dc3')
plt.plot(range(1, EPOCHS+1), bidirectional_train_result['precision pos
itive'], color='#f03b20')
plt.title('Bidirectional Precision (Training)')
plt.ylabel('Precision')
plt.xlabel('epoch', labelpad=2)
plt.xticks(np.arange(1, EPOCHS+1))
plt.legend(['precision negative', 'precision neutral', 'precision posi
tive'], loc='lower left')
plt.grid(True)
plt.show()

# precision plot
fig, ax = plt.subplots(figsize=(8,6))
plt.plot(range(1, EPOCHS+1), bidirectional_val_result['precision negat
ive'], color='#fdb462')
```

```
plt.plot(range(1, EPOCHS+1), bidirectional_val_result['precision neutr
al'], color='#af8dc3')
plt.plot(range(1, EPOCHS+1), bidirectional_val_result['precision posit
ive'], color='#f03b20')
plt.title('Bidirectional Precision (Validation)')
plt.ylabel('Precision')
plt.xlabel('epoch', labelpad=2)
plt.xticks(np.arange(1, EPOCHS+1))
plt.legend(['precision negative', 'precision neutral', 'precision posi
tive'], loc='lower left')
plt.grid(True)

# recall plot
fig, ax = plt.subplots(figsize=(8,6))
plt.plot(range(1, EPOCHS+1), bidirectional_train_result['recall negati
ve'], color='#fdb462')
plt.plot(range(1, EPOCHS+1), bidirectional_train_result['recall neutra
l'], color='#af8dc3')
plt.plot(range(1, EPOCHS+1), bidirectional_train_result['recall positi
ve'], color='#f03b20')
plt.title('Bidirectional Recall (Training)')
plt.ylabel('Recall')
plt.xlabel('epoch', labelpad=2)
plt.xticks(np.arange(1, EPOCHS+1))
plt.legend(['recall negative', 'recall neutral', 'recall positive'], l
oc='lower left')
plt.grid(True)
plt.show()

# recall plot
fig, ax = plt.subplots(figsize=(8,6))
plt.plot(range(1, EPOCHS+1), bidirectional_val_result['recall negative
'], color='#fdb462')
plt.plot(range(1, EPOCHS+1), bidirectional_val_result['recall neutral'
], color='#af8dc3')
plt.plot(range(1, EPOCHS+1), bidirectional_val_result['recall positive
'], color='#f03b20')
plt.title('Bidirectional Recall (Validation)')
plt.ylabel('Recall')
plt.xlabel('epoch', labelpad=2)
plt.xticks(np.arange(1, EPOCHS+1))
plt.legend(['recall negative', 'recall neutral', 'recall positive'], l
oc='lower left')
plt.grid(True)
plt.show()
```

```
# f1 plot
fig, ax = plt.subplots(figsize=(8,6))
plt.plot(range(1, EPOCHS+1), bidirectional_train_result['f1 negative']
, color='#fdb462')
plt.plot(range(1, EPOCHS+1), bidirectional_train_result['f1 neutral'],
 color='#af8dc3')
plt.plot(range(1, EPOCHS+1), bidirectional_train_result['f1 positive']
, color='#f03b20')
plt.title('Bidirectional f1 (Training)')
plt.ylabel('f1')
plt.xlabel('epoch', labelpad=2)
plt.xticks(np.arange(1, EPOCHS+1))
plt.legend(['f1 negative', 'f1 neutral', 'f1 positive'], loc='lower le
ft')
plt.grid(True)
plt.show()

# f1 plot
fig, ax = plt.subplots(figsize=(8,6))
plt.plot(range(1, EPOCHS+1), bidirectional_val_result['f1 negative'],
color='#fdb462')
plt.plot(range(1, EPOCHS+1), bidirectional_val_result['f1 neutral'], c
olor='#af8dc3')
plt.plot(range(1, EPOCHS+1), bidirectional_val_result['f1 positive'],
color='#f03b20')
plt.title('Bidirectional f1 (Validation)')
plt.ylabel('f1')
plt.xlabel('epoch', labelpad=2)
plt.xticks(np.arange(1, EPOCHS+1))
plt.legend(['f1 negative', 'f1 neutral', 'f1 positive'], loc='lower le
ft')
plt.grid(True)
plt.show()

SIDE BY SIDE COMPARISON

#BEST_EPOCH_VANILLA = np.argmax(vanilla_val_result['accuracy'])+1
#BEST_EPOCH_BIDIRECTIONAL = np.argmax(bidirectional_val_result['accura
cy'])+1

BEST_EPOCH_VANILLA = 5
BEST_EPOCH_BIDIRECTIONAL = 6

def autolabel(rects):
```

```
    """Attach a text label above each bar in *rects*, displaying its h
eight."""
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{:.1f}%'.format(height*100),
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')
labels = ['acc', 'rec neg', 'rec neu', 'rec pos', 'pre neg' ,'pre neu'
, 'pre pos', 'f1 neg', 'f1 neu', 'f1 pos']
x = np.arange(len(labels))  # the label locations
width = 0.35  # the width of the bars

fig, ax = plt.subplots(figsize=(16,10))
rects1 = ax.bar(x - 0.02 - width/2, vanilla_train_result.loc[BEST_EPOC
H_VANILLA-
1], width, label='Vanilla', color='#BCBDDC', edgecolor='#9E9AC8')
rects2 = ax.bar(x + 0.02 + width/2, bidirectional_train_result.loc[BES
T_EPOCH_BIDIRECTIONAL-
1], width, label='Bidirectional', color='#212349', edgecolor='#7C7DBD'
)
# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('Scores')
ax.set_title('Training Scores')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend(loc='lower right')
autolabel(rects1)
autolabel(rects2)

labels = ['acc', 'rec neg', 'rec neu', 'rec pos', 'pre neg' ,'pre neu'
, 'pre pos', 'f1 neg', 'f1 neu', 'f1 pos']
x = np.arange(len(labels))  # the label locations
width = 0.35  # the width of the bars

fig, ax = plt.subplots(figsize=(16,10))
rects1 = ax.bar(x - 0.02 - width/2, vanilla_val_result.loc[BEST_EPOCH_
VANILLA-
1], width, label='Vanilla', color='#BCBDDC', edgecolor='#9E9BCBDDCAC8'
)
rects2 = ax.bar(x + 0.02 + width/2, bidirectional_val_result.loc[BEST_
EPOCH_BIDIRECTIONAL-
1], width, label='Bidirectional', color='#212349', edgecolor='#7C7DBD'
)
```

```
# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_ylabel('Scores')
ax.set_title('Validation Scores')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend(loc='lower right')
autolabel(rects1)
autolabel(rects2)

ax =  plt.subplot()
cmap = sns.light_palette('#212349', as_cmap=True, reverse=True)
sns.heatmap(vanilla_metrics_train.get_cm()[BEST_EPOCH_VANILLA-
1], annot=True, fmt='g', ax=ax, cmap=cmap)
# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Vanilla Training (Epoch '+str(BEST_EPOCH_VANILLA)+')');
ax.xaxis.set_ticklabels(['negative', 'neutral', 'positive']); ax.yaxis
.set_ticklabels(['negative', 'neutral', 'positive']);

ax =  plt.subplot()
sns.heatmap(vanilla_metrics_val.get_cm()[BEST_EPOCH_VANILLA-
1], annot=True, fmt='g', ax=ax, cmap=cmap)
# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Vanilla Testing (Epoch '+str(BEST_EPOCH_VANILLA)+')');
ax.xaxis.set_ticklabels(['negative', 'neutral', 'positive']); ax.yaxis
.set_ticklabels(['negative', 'neutral', 'positive']);

sns.heatmap(bidirectional_metrics_train.get_cm()[BEST_EPOCH_BIDIRECTIO
NAL-1], annot=True, fmt='g', ax=ax, cmap=cmap)
# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Bidirectional Training (Epoch '+str(BEST_EPOCH_BIDIRECTI
ONAL)+')');
ax.xaxis.set_ticklabels(['negative', 'neutral', 'positive']); ax.yaxis
.set_ticklabels(['negative', 'neutral', 'positive']);

ax =  plt.subplot()
sns.heatmap(bidirectional_metrics_val.get_cm()[BEST_EPOCH_BIDIRECTIONA
L-1], annot=True, fmt='g', ax=ax, cmap=cmap)
# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Bidirectional Testing (Epoch '+str(BEST_EPOCH_BIDIRECTIO
NAL)+')');
```

```
ax.xaxis.set_ticklabels(['negative', 'neutral', 'positive']); ax.yaxis
.set_ticklabels(['negative', 'neutral', 'positive']);

WORD COULD

    # Set figure size
    plt.figure(figsize=(40, 30))
    # Display image
    plt.imshow(wordcloud)
    # No axis details
    plt.title(title, fontsize=16)
    plt.axis("off");
sw = ['dress', 'fabric', 'shirt', 'sweater', 'skirt', 'also', 'size',
'one', 'look', 'ordered','color','would','top', 'fit', 'looks', 'looke
d']

all_wordcloud = WordCloud(width = 800, height = 600, stopwords = sw).g
enerate(' '.join(df['tokenized_unlist']))
positive_wordcloud = WordCloud(width = 800, height = 600, stopwords =
sw).generate(' '.join(df[df['Sentiment']=='Positive']['tokenized_unlis
t']))
neutral_wordcloud = WordCloud(width = 800, height = 600, stopwords = s
w).generate(' '.join(df[df['Sentiment']=='Neutral']['tokenized_unlist'
]))
negative_wordcloud = WordCloud(width = 800, height = 600, stopwords =
sw).generate(' '.join(df[df['Sentiment']=='Negative']['tokenized_unlis
t']))

plot_cloud(all_wordcloud, 'All reviews WordCloud')
plot_cloud(positive_wordcloud, 'Positive reviews WordCloud')
plot_cloud(neutral_wordcloud, 'Neutral reviews WordCloud')
plot_cloud(negative_wordcloud, 'Negative reviews WordCloud')

def get_top_n_gram(corpus,ngram_range,n=None):
    vec = CountVectorizer(ngram_range=ngram_range).fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocab
ulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]

bigrams_positive_list = get_top_n_gram(df[df['Sentiment']=='Positive']
['tokenized_unlist'],(2,2),100)
```

```
bigrams_neutral_list = get_top_n_gram(df[df['Sentiment']=='Neutral']['
tokenized_unlist'],(2,2),100)
bigrams_negative_list = get_top_n_gram(df[df['Sentiment']=='Negative']
['tokenized_unlist'],(2,2),100)

bigrams_positive = {}
for a, x in bigrams_positive_list:
    bigrams_positive[a] = x

bigrams_neutral = {}
for a, x in bigrams_neutral_list:
    bigrams_neutral[a] = x

bigrams_negative = {}
for a, x in bigrams_negative_list:
    bigrams_negative[a] = x

dir = path.dirname(__file__) if "__file__" in locals() else os.getcwd(
)
positive_mask = np.array(Image.open(path.join(dir, "positive.jpg")))
wc1 = WordCloud(width = 800, height = 600, background_color="white", c
ontour_width=3, contour_color='steelblue')
wc1.generate_from_frequencies(frequencies=bigrams_positive)
plot_cloud(wc1, 'Positive Bigrams')
#plt.imshow(wc1)
#lt.imshow(positive_mask, cmap=plt.cm.gray, interpolation='bilinear')
```