COURSEWORK TEXT-AS-DATA

STUDENT: ELISSAIOS BAXEVANIS

STUDENT ID: 25889922B

STUDENT MAIL: 2588922B@STUDENT.GLA.AC.UK

DATE: 04-06-2021

In [1]:
```
import pandas as pd
import numpy as np

train_data =
pd.read_json("https://raw.githubusercontent.com/rpsoft/tad_course/main/reddit_sentiment_train.json


validation_data =
pd.read_json("https://raw.githubusercontent.com/rpsoft/tad_course/main/reddit_sentiment_validation


test_data =
pd.read_json("https://raw.githubusercontent.com/rpsoft/tad_course/main/reddit_sentiment_test.json"
```

In [2]:
```
import sys
print(sys.version)
import importlib

!pip install -U yellowbrick
%matplotlib inline
"""
!python -m spacy download en_core_web_md

import spacy
nlp = spacy.load('en_core_web_sm')

"""
!pip install spacy==3.0.6
!python -m spacy download en_core_web_sm
import spacy
nlp = spacy.load('en_core_web_sm', disable=['ner'])
!pip install eli5
from eli5.sklearn import explain_linear_classifier_weights
from eli5 import show_prediction
import matplotlib.pyplot as plt
from sklearn.preprocessing import OrdinalEncoder
from sklearn.metrics import precision_recall_fscore_support as score
# Downloa a stopword list
import nltk
nltk.download('stopwords')
#SET STOP_WORDS from NLTK
stop_words = (nltk.corpus.stopwords.words('english'))
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
import collections
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.base import BaseEstimator, TransformerMixin
!pip install vaderSentiment
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
import warnings
warnings.filterwarnings('ignore')
```

```
3.7.10 (default, May  3 2021, 02:48:31)
[GCC 7.5.0]
Requirement already up-to-date: yellowbrick in /usr/local/lib/python3.7/dist-packages (1.3.post1)
Requirement already satisfied, skipping upgrade: numpy<1.20,>=1.16.0 in /usr/local/lib/python3.7/dist-packages (from yellowbrick) (1.19.5)
Requirement already satisfied, skipping upgrade: matplotlib!=3.0.0,>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from yellowbrick) (3.2.2)
Requirement already satisfied, skipping upgrade: scikit-learn>=0.20 in /usr/local/lib/python3.7/dist-packages (from yellowbrick) (0.22.2.post1)
Requirement already satisfied, skipping upgrade: cycler>=0.10.0 in /usr/local/lib/python3.7/dist-packages (from yellowbrick) (0.10.0)
Requirement already satisfied, skipping upgrade: scipy>=1.0.0 in /usr/local/lib/python3.7/dist-packages (from yellowbrick) (1.4.1)
Requirement already satisfied, skipping upgrade: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (1.3.1)
Requirement already satisfied, skipping upgrade: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (2.4.7)
Requirement already satisfied, skipping upgrade: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib!=3.0.0,>=2.0.2->yellowbrick) (2.8.1)
Requirement already satisfied, skipping upgrade: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.20->yellowbrick) (1.0.1)
Requirement already satisfied, skipping upgrade: six in /usr/local/lib/python3.7/dist-packages (from cycler>=0.10.0->yellowbrick) (1.15.0)
Requirement already satisfied: spacy==3.0.6 in /usr/local/lib/python3.7/dist-packages (3.0.6)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.7/dist-packages (from spacy==3.0.6) (20.9)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /usr/local/lib/python3.7/dist-packages (from spacy==3.0.6) (4.41.1)
Requirement already satisfied: wasabi<1.1.0,>=0.8.1 in /usr/local/lib/python3.7/dist-packages (from spacy==3.0.6) (0.8.2)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from spacy==3.0.6) (2.0.5)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.7/dist-packages (from spacy==3.0.6) (2.11.3)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.4 in /usr/local/lib/python3.7/dist-packages (from spacy==3.0.6) (3.0.5)
Requirement already satisfied: catalogue<2.1.0,>=2.0.3 in /usr/local/lib/python3.7/dist-packages (from spacy==3.0.6) (2.0.4)
Requirement already satisfied: pydantic<1.8.0,>=1.7.1 in /usr/local/lib/python3.7/dist-packages (from spacy==3.0.6) (1.7.4)
Requirement already satisfied: numpy>=1.15.0 in /usr/local/lib/python3.7/dist-packages (from spacy==3.0.6) (1.19.5)
Requirement already satisfied: typing-extensions<4.0.0.0,>=3.7.4; python_version < "3.8" in /usr/local/lib/python3.7/dist-packages (from spacy==3.0.6) (3.7.4.3)
Requirement already satisfied: blis<0.8.0,>=0.4.0 in /usr/local/lib/python3.7/dist-packages (from spacy==3.0.6) (0.4.1)
Requirement already satisfied: pathy>=0.3.5 in /usr/local/lib/python3.7/dist-packages (from spacy==3.0.6) (0.5.2)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3.7/dist-packages (from spacy==3.0.6) (1.0.5)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from spacy==3.0.6) (3.0.5)
Requirement already satisfied: srsly<3.0.0,>=2.4.1 in /usr/local/lib/python3.7/dist-packages (from spacy==3.0.6) (2.4.1)
Requirement already satisfied: typer<0.4.0,>=0.3.0 in /usr/local/lib/python3.7/dist-packages (from spacy==3.0.6) (0.3.2)
Requirement already satisfied: thinc<8.1.0,>=8.0.3 in /usr/local/lib/python3.7/dist-packages (from spacy==3.0.6) (8.0.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from spacy==3.0.6) (57.0.0)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.7/dist-packages (from spacy==3.0.6) (2.23.0)
Requirement already satisfied: pyparsing>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from packaging>=20.0->spacy==3.0.6) (2.4.7)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-packages (from jinja2->spacy==3.0.6) (2.0.1)
Requirement already satisfied: zipp>=0.5; python_version < "3.8" in /usr/local/lib/python3.7/dist-packages (from catalogue<2.1.0,>=2.0.3->spacy==3.0.6) (3.4.1)
Requirement already satisfied: smart-open<4.0.0,>=2.2.0 in /usr/local/lib/python3.7/dist-packages (from pathy>=0.3.5->spacy==3.0.6) (3.0.0)
Requirement already satisfied: click<7.2.0,>=7.1.1 in /usr/local/lib/python3.7/dist-packages (from typer<0.4.0,>=0.3.0->spacy==3.0.6) (7.1.2)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests<3.0.0,>=2.13.0->spacy==3.0.6) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests<3.0.0,>=2.13.0->spacy==3.0.6) (2020.12.5)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests<3.0.0,>=2.13.0->spacy==3.0.6) (2.10)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests<3.0.0,>=2.13.0->spacy==3.0.6) (1.24.3)
2021-06-04 14:56:54.509045: I tensorflow/stream_executor/platform/default/dso_loader.cc:53] Successfully opened dynamic library libcudart.so.11.0
Requirement already satisfied: en-core-web-sm==3.0.0 from https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-3.0.0/en_core_web_sm-3.0.0-py3-none-any.whl#egg=en_core_web_sm==3.0.0 in /usr/local/lib/python3.7/dist-packages (3.0.0)
Requirement already satisfied: spacy<3.1.0,>=3.0.0 in /usr/local/lib/python3.7/dist-packages (from en-core-web-sm==3.0.0) (3.0.6)
Requirement already satisfied: spacy-legacy<3.1.0,>=3.0.4 in /usr/local/lib/python3.7/dist-packages (from spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (3.0.5)
Requirement already satisfied: preshed<3.1.0,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (3.0.5)
Requirement already satisfied: murmurhash<1.1.0,>=0.28.0 in /usr/local/lib/python3.7/dist-packages (from spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (1.0.5)
Requirement already satisfied: wasabi<1.1.0,>=0.8.1 in /usr/local/lib/python3.7/dist-packages (from spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (0.8.2)
Requirement already satisfied: srsly<3.0.0,>=2.4.1 in /usr/local/lib/python3.7/dist-packages (from spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (2.4.1)
Requirement already satisfied: thinc<8.1.0,>=8.0.3 in /usr/local/lib/python3.7/dist-packages (from spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (8.0.3)
Requirement already satisfied: tqdm<5.0.0,>=4.38.0 in /usr/local/lib/python3.7/dist-packages (from spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (4.41.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.7/dist-packages (from spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (2.11.3)
Requirement already satisfied: catalogue<2.1.0,>=2.0.3 in /usr/local/lib/python3.7/dist-packages (from spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (2.0.4)
Requirement already satisfied: pathy>=0.3.5 in /usr/local/lib/python3.7/dist-packages (from spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (0.5.2)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (57.0.0)
Requirement already satisfied: typer<0.4.0,>=0.3.0 in /usr/local/lib/python3.7/dist-packages (from spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (0.3.2)
Requirement already satisfied: typing-extensions<4.0.0.0,>=3.7.4; python_version < "3.8" in /usr/local/lib/python3.7/dist-packages (from spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (3.7.4.3)
Requirement already satisfied: requests<3.0.0,>=2.13.0 in /usr/local/lib/python3.7/dist-packages (from spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (2.23.0)
Requirement already satisfied: numpy>=1.15.0 in /usr/local/lib/python3.7/dist-packages (from spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (1.19.5)
Requirement already satisfied: pydantic<1.8.0,>=1.7.1 in /usr/local/lib/python3.7/dist-packages (from spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (1.7.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.7/dist-packages (from spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (20.9)
Requirement already satisfied: cymem<2.1.0,>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (2.0.5)
Requirement already satisfied: blis<0.8.0,>=0.4.0 in /usr/local/lib/python3.7/dist-packages (from spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (0.4.1)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-packages (from jinja2->spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (2.0.1)
Requirement already satisfied: zipp>=0.5; python_version < "3.8" in /usr/local/lib/python3.7/dist-packages (from catalogue<2.1.0,>=2.0.3->spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (3.4.1)
Requirement already satisfied: smart-open<4.0.0,>=2.2.0 in /usr/local/lib/python3.7/dist-packages (from pathy>=0.3.5->spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (3.0.0)
Requirement already satisfied: click<7.2.0,>=7.1.1 in /usr/local/lib/python3.7/dist-packages (from typer<0.4.0,>=0.3.0->spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (7.1.2)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests<3.0.0,>=2.13.0->spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests<3.0.0,>=2.13.0->spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (1.24.3)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests<3.0.0,>=2.13.0->spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests<3.0.0,>=2.13.0->spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (2020.12.5)
Requirement already satisfied: pyparsing>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from packaging>=20.0->spacy<3.1.0,>=3.0.0->en-core-web-sm==3.0.0) (2.4.7)
✔ Download and installation successful
You can now load the package via spacy.load('en_core_web_sm')
Requirement already satisfied: eli5 in /usr/local/lib/python3.7/dist-packages (0.11.0)
Requirement already satisfied: numpy>=1.9.0 in /usr/local/lib/python3.7/dist-packages (from eli5) (1.19.5)
Requirement already satisfied: tabulate>=0.7.7 in /usr/local/lib/python3.7/dist-packages (from eli5) (0.8.9)
Requirement already satisfied: scikit-learn>=0.20 in /usr/local/lib/python3.7/dist-packages (from eli5) (0.22.2.post1)
Requirement already satisfied: attrs>16.0.0 in /usr/local/lib/python3.7/dist-packages (from eli5) (21.2.0)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from eli5) (1.15.0)
Requirement already satisfied: graphviz in /usr/local/lib/python3.7/dist-packages (from eli5) (0.10.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.7/dist-packages (from eli5) (2.11.3)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from eli5) (1.4.1)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.20->eli5) (1.0.1)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-packages (from jinja2->eli5) (2.0.1)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:144: FutureWarning: The sklearn.feature_selection.base module is  deprecated in version 0.22 and will be removed in version 0.24. The corresponding classes / functions should instead be imported from sklearn.feature_selection. Anything that cannot be imported from sklearn.feature_selection is now part of the private API.
  warnings.warn(message, FutureWarning)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
Requirement already satisfied: vaderSentiment in /usr/local/lib/python3.7/dist-packages (3.3.2)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from vaderSentiment) (2.23.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->vaderSentiment) (2020.12.5)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests->vaderSentiment) (1.24.3)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->vaderSentiment) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->vaderSentiment) (2.10)
```

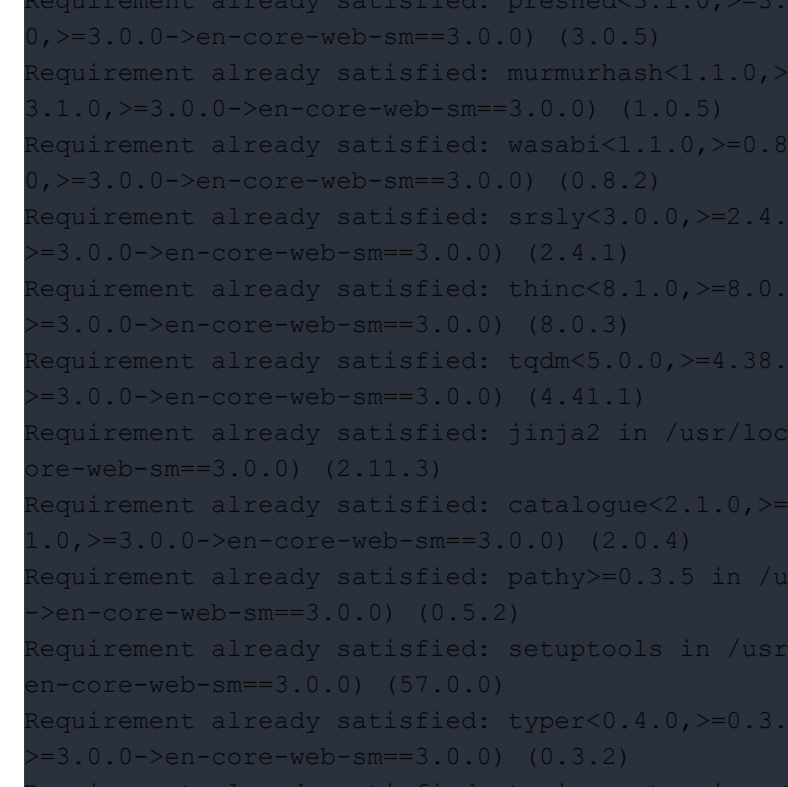
Distribution of classes per data set (train-valid-test)

In [3]:
```
#check for missing values
print("Missing labels in train: ",train_data['sentiment.polarity'].isnull().values.any())
print("Missing labels in validation:
",validation_data['sentiment.polarity'].isnull().values.any())
print("Missing labels in test: ",test_data['sentiment.polarity'].isnull().values.any())
```

```
Missing labels in train:  False
Missing labels in validation:  False
Missing labels in test:  False
```

In [4]:
```
train_polar = pd.DataFrame(data=train_data['sentiment.polarity']).value_counts()
valid_polar = pd.DataFrame(data=validation_data['sentiment.polarity']).value_counts()
test_polar = pd.DataFrame(data=test_data['sentiment.polarity']).value_counts()
columns = ['Train','Validation','Test']
distribution = pd.concat([train_polar,valid_polar,test_polar],axis=1)
distribution.columns=columns

distribution.plot(kind='bar',stacked=True)
plt.title("Distributions of Labels in Train-Validation-Test sets")
plt.xlabel("Occurencies per class per set")
plt.ylabel("Class")
plt.grid(axis='y')
plt.legend(fontsize=11,shadow=True)
plt.show()
```



In [5]:
```
distribution_percentages = distribution.copy()
distribution_percentages['Train Class %'] =
(distribution_percentages['Train']/(distribution_percentages['Train'].sum()))*100
distribution_percentages['Validation Class %'] =
(distribution_percentages['Validation']/(distribution_percentages['Validation'].sum()))*100
distribution_percentages['Test Class %'] =
(distribution_percentages['Test']/( distribution_percentages['Test'].sum()))*100
distribution_percentages = distribution_percentages.drop('Train',axis=1)
distribution_percentages = distribution_percentages.drop('Validation',axis=1)
distribution_percentages = distribution_percentages.drop('Test',axis=1)
distribution_percentages.round(3)
```

Out[5]:

	Train Class %	Validation Class %	Test Class %
neutral	63.264	63.075	62.600
positive	26.619	27.179	27.440
negative	7.233	6.915	7.022
very positive	2.084	2.348	2.141
very negative	0.799	0.482	0.797

In [6]:
```
#transform labels from categorical to numerical

encoder = OrdinalEncoder(dtype=int)
train_labels = train_data['sentiment.polarity']
validation_labels = validation_data['sentiment.polarity']
test_labels = test_data['sentiment.polarity']
train_labels_enc = encoder.fit_transform(np.array(train_labels).reshape(-1,1)).ravel()
validation_labels_enc = encoder.fit_transform(np.array(validation_labels).reshape(-1,1)).ravel()
test_labels_enc = encoder.fit_transform(np.array(test_labels).reshape(-1,1)).ravel()
```

In [7]:
```
#The following functions are the ones used during labs 1 and 2 in TaD
#Tokenize
def spacy_tokenize(string):
  tokens = list()
  doc = nlp(string)
  for token in doc:
    tokens.append(token)
  return tokens

#Normalize
def normalize(tokens):
  normalized_tokens = list()
  for token in tokens:
    normalized = token.text.lower().strip()
    if ((token.is_alpha or token.is_digit)):
      normalized_tokens.append(normalized)
  return normalized_tokens
  #return normalized_tokens

#Tokenize and normalize
def tokenize_normalize(string):
  return normalize(spacy_tokenize(string))
  #Normalize + lemmatize
```

In [8]:
```
#Based on labs 1 and 2 in TaD
#functions to provide metrics for the classifiers
def evaluation_summary(description, predictions, true_labels):
  accuracy = accuracy_score(predictions, true_labels)

  precision_macro = precision_score(predictions, true_labels, average = 'macro',zero_division=0)
  recall_macro= recall_score(predictions, true_labels, average = 'macro',zero_division=0)
  f1_macro = f1_score(predictions, true_labels,average = 'macro')

  precision_weighted = precision_score(predictions, true_labels, average =
'weighted',zero_division=0)
  recall_weighted= recall_score(predictions, true_labels, average = 'weighted',zero_division=0)
  f1_weighted = f1_score(predictions, true_labels,average = 'weighted')

  print("Classifier '%s' has Acc=%0.3f P=%0.3f R=%0.3f F1=%0.3f" %
(description,accuracy,precision_macro,recall_macro,f1_macro))
  return precision_macro, recall_macro,accuracy, f1_macro,
precision_weighted,recall_weighted,f1_weighted
def evaluation_summary_q2q3(description, predictions, true_labels,flag=0):
  accuracy = accuracy_score(predictions, true_labels)

  precision_macro = precision_score(predictions, true_labels, average = 'macro',zero_division=0)
  recall_macro= recall_score(predictions, true_labels, average = 'macro',zero_division=0)
  f1_macro = f1_score(predictions, true_labels,average = 'macro')

  print("'%s' has Acc=%0.3f P=%0.3f R=%0.3f F1=%0.3f" %
(description,accuracy,precision_macro,recall_macro,f1_macro))
  if flag == 1:
    return precision_macro, recall_macro,accuracy, f1_macro
```

One-Hot & TF-IDF vectorization across train-val-test sets

In [9]:
```
one_hot_vectorizer = CountVectorizer(tokenizer = tokenize_normalize,binary = True)

one_hot_vectorizer.fit(train_data['body'])

train_features_ohv = one_hot_vectorizer.transform(train_data['body'])
validation_features_ohv = one_hot_vectorizer.transform(validation_data['body'])
test_features_ohv = one_hot_vectorizer.transform(test_data['body'])

tfidf_vectorizer = TfidfVectorizer(tokenizer = tokenize_normalize)

tfidf_vectorizer.fit(train_data['body'])

train_features_tfidf = tfidf_vectorizer.transform(train_data['body'])
validation_features_tfidf = tfidf_vectorizer.transform(validation_data['body'])
test_features_tfidf = tfidf_vectorizer.transform(test_data['body'])
```

Illustration examples

In [10]:
```
sample = np.random.randint(len(train_data['body']))
print("Text: ",train_data['body'][sample],"\nPolarity: ",train_data['sentiment.polarity']
[sample])
```

```
Text:  Her services aren't cheap it seems.
Polarity:  positive
```

In [11]:
```
print(train_data['body'][sample])
tokenize_normalize(train_data['body'][sample])
```

```
Her services aren't cheap it seems.
```

Out[11]:
```
['her', 'services', 'are', 'cheap', 'it', 'seems']
```

QUESTION 1

In [12]:
```
#DUMMY CLASSIFIER
from sklearn.dummy import DummyClassifier
#WITH STRATEGY : MOST_FREQUENT
dummy_mostfrequent = DummyClassifier(strategy='most_frequent')
dummy_mostfrequent.fit(train_features_ohv, train_labels_enc)
dummmy_mf_train_pr,dummy_mf_train_rec,dummy_mf_train_acc,
dummy_mf_train_f1,dummy_mf_train_p_wtd,dummy_mf_train_r_wtd,dummy_mf_train_f1_wtd =
evaluation_summary("DummyClassifier Most_Frequent:
Train",dummy_mostfrequent.predict(train_features_ohv),train_labels_enc)
dummmy_mf_test_pr,dummy_mf_test_rec,dummy_mf_test_acc,
dummy_mf_test_f1,dummy_mf_test_p_wtd,dummy_mf_test_r_wtd,dummy_mf_test_f1_wtd=
evaluation_summary("DummyClassifier Most_Frequent:
Test",dummy_mostfrequent.predict(test_features_ohv),test_labels_enc)
#WITH STRATEGY : STRATIFIED
dummy_stratified = DummyClassifier(strategy='stratified')
dummy_stratified.fit(train_features_ohv, train_labels_enc)
dummy_strf_train_pr,
dummy_strf_train_rec,dummy_strf_train_acc,dummy_strf_train_f1,dummy_strf_train_p_wtd,dummy_strf_t
 evaluation_summary("DummyClassifier Stratified:
Train",dummy_stratified.predict(train_features_ohv),train_labels_enc)
dummy_strf_test_pr,
dummy_strf_test_rec,dummy_strf_test_acc,dummy_strf_test_f1,dummy_strf_test_p_wtd,dummy_strf_test_
 = evaluation_summary("DummyClassifier Stratified:
Test",dummy_stratified.predict(test_features_ohv),test_labels_enc)

#LOGISTIC REGRESSION
from sklearn.linear_model import LogisticRegression
#WITH ONE-HOT VECTORIZATION
lr = LogisticRegression()
lr_model = lr.fit(train_features_ohv,train_labels_enc)
lr_ohv_train_pr, lr_ohv_train_rec,lr_ohv_train_acc ,
lr_ohv_train_f1,lr_ohv_train_p_wtd,lr_ohv_train_r_wtd,lr_ohv_train_f1_wtd =
evaluation_summary("LR one-hot: Train",lr_model.predict(train_features_ohv), train_labels_enc)
lr_ohv_test_pr, lr_ohv_test_rec,lr_ohv_test_acc,
lr_ohv_test_f1,lr_ohv_test_p_wtd,lr_ohv_test_r_wtd,lr_ohv_test_f1_wtd = evaluation_summary("LR
one-hot: Test",lr_model.predict(test_features_ohv), test_labels_enc)
#WITH TF-IDF VECTORIZATION
lr_tfidf = LogisticRegression()
lr_tfidf_model = lr_tfidf.fit(train_features_tfidf,train_labels_enc)
lr_tfidf_train_pr, lr_tfidf_train_rec, lr_tfidf_train_acc ,lr_tfidf_train_f1,
lr_tfidf_train_p_wtd,lr_tfidf_train_r_wtd,lr_tfidf_train_f1_wtd = evaluation_summary("LR tfidf:
Train",lr_tfidf_model.predict(train_features_tfidf), train_labels_enc)
lr_tfidf_test_pr, lr_tfidf_test_rec,lr_tfidf_test_acc ,lr_tfidf_test_f1,
lr_tfidf_test_p_wtd,lr_tfidf_test_r_wtd,lr_tfidf_test_f1_wtd = evaluation_summary("LR
tfidf:Test",lr_tfidf_model.predict(test_features_tfidf), test_labels_enc)

#SVM CLASSIFIER
from sklearn.svm import SVC
#WITH RBF KERNEL (SVC)
svc = SVC()
svc_model = svc.fit(train_features_ohv, train_labels_enc)
svc_train_pr, svc_train_rec,svc_train_acc ,svc_train_f1,
svc_train_p_wtd,svc_train_r_wtd,svc_train_f1_wtd = evaluation_summary("SVC one-hot:
Train",svc_model.predict(train_features_ohv) , train_labels_enc)
svc_test_pr, svc_test_rec,svc_test_acc,
svc_test_f1,svc_test_p_wtd,svc_test_r_wtd,svc_test_f1_wtd= evaluation_summary("SVC one-hot:
Test",svc_model.predict(test_features_ohv), test_labels_enc)

#DECISION TREE CLASSIFIER
from sklearn.naive_bayes import MultinomialNB
#WITH ONE-HOT VECTORIZATION
mnb_ohv = MultinomialNB()
mnb_ohv_model = mnb_ohv.fit(train_features_ohv,train_labels_enc)
mnb_ohv_train_pr, mnb_ohv_train_rec,mnb_ohv_train_acc,
mnb_ohv_train_f1,mnb_train_p_wtd,mnb_train_r_wtd,mnb_train_f1_wtd = evaluation_summary("mnb one-
hot: Train", mnb_ohv_model.predict(train_features_ohv),train_labels_enc)
mnb_ohv_test_pr, mnb_ohv_test_rec,mnb_ohv_test_acc
,mnb_ohv_test_f1,mnb_test_p_wtd,mnb_test_r_wtd,mnb_test_f1_wtd =evaluation_summary("mnb one-hot:
Train", mnb_ohv_model.predict(test_features_ohv),test_labels_enc)
```

```
Classifier 'DummyClassifier Most_Frequent: Train' has Acc=0.633 P=0.200 R=0.127 F1=0.155
Classifier 'DummyClassifier Most_Frequent: Test' has Acc=0.626 P=0.200 R=0.125 F1=0.154
Classifier 'DummyClassifier Stratified: Train' has Acc=0.477 P=0.201 R=0.201 F1=0.201
Classifier 'DummyClassifier Stratified: Test' has Acc=0.487 P=0.217 R=0.218 F1=0.217
Classifier 'LR one-hot: Train' has Acc=0.963 P=0.845 R=0.982 F1=0.901
Classifier 'LR one-hot: Test' has Acc=0.748 P=0.441 R=0.635 F1=0.489
Classifier 'LR tfidf: Train' has Acc=0.817 P=0.418 R=0.909 F1=0.473
Classifier 'LR tfidf:Test' has Acc=0.740 P=0.330 R=0.574 F1=0.356
Classifier 'SVC one-hot: Train' has Acc=0.862 P=0.423 R=0.750 F1=0.454
Classifier 'SVC one-hot: Test' has Acc=0.729 P=0.288 R=0.458 F1=0.287
Classifier 'mnb one-hot: Train' has Acc=0.818 P=0.384 R=0.730 F1=0.407
Classifier 'mnb one-hot: Train' has Acc=0.684 P=0.264 R=0.337 F1=0.259
```

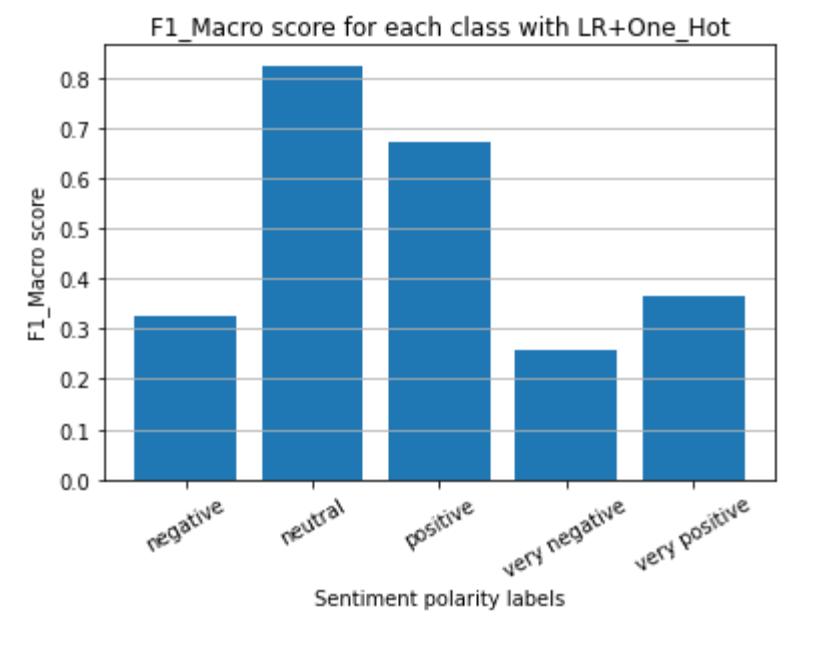
In [13]:
```
train_metrics = {'Precision_macro':[dummmy_mf_train_pr,
                                   dummy_strf_train_pr,
                                   lr_ohv_train_pr,
                                   lr_tfidf_train_pr,
                                   svc_train_pr,
                                   mnb_ohv_train_pr],
            'Recall_macro':[dummy_mf_train_rec,
                                   dummy_strf_train_rec,
                                   lr_ohv_train_rec,
                                   lr_tfidf_train_rec,
                                   svc_train_rec,
                                   mnb_ohv_train_rec],
                'F1_macro':[dummy_mf_train_f1,
                                   dummy_strf_train_f1,
                                   lr_ohv_train_f1,
                                   lr_tfidf_train_f1,
                                   svc_train_f1,
                                   mnb_ohv_train_f1],
       'Precision_weighted':[dummy_mf_train_p_wtd,
                                   dummy_strf_train_p_wtd,
                                   lr_ohv_train_p_wtd,
                                   lr_tfidf_train_p_wtd,
                                   svc_train_p_wtd,
                                   mnb_train_p_wtd],
          'Recall_weighted': [dummy_mf_train_r_wtd,
                                 dummy_strf_train_r_wtd,
                                 lr_ohv_train_r_wtd,
                                 lr_tfidf_train_r_wtd,
                                 svc_train_r_wtd,
                                 mnb_train_r_wtd],
             'F1_weighted': [dummy_mf_train_f1_wtd,
                                 dummy_strf_train_f1_wtd,
                                 lr_ohv_train_f1_wtd,
                                 lr_tfidf_train_f1_wtd,
                                 svc_train_f1_wtd,
                                 mnb_train_f1_wtd
                                 ],
               'Accuracy':[dummy_mf_train_acc,
                                 dummy_strf_train_acc,
                                 lr_ohv_train_acc,
                                 lr_tfidf_train_acc,
                                 svc_train_acc,
                                 mnb_ohv_train_acc]}
           }
test_metrics = {'Precision_macro':[dummmy_mf_test_pr,
                                   dummy_strf_test_pr,
                                   lr_ohv_test_pr,
                                   lr_tfidf_test_pr,
                                   svc_test_pr,
                                   mnb_ohv_test_pr],
            'Recall_macro':[dummy_mf_test_rec,
                                   dummy_strf_test_rec,
                                   lr_ohv_test_rec,
                                   lr_tfidf_test_rec,
                                   svc_test_rec,
                                   mnb_ohv_test_rec],
                'F1_macro':[dummy_mf_test_f1,
                                   dummy_strf_test_f1,
                                   lr_ohv_test_f1,
                                   lr_tfidf_test_f1,
                                   svc_test_f1,
                                   mnb_ohv_test_f1],
       'Precision_weighted':[dummy_mf_test_p_wtd,
                                   dummy_strf_test_p_wtd,
                                   lr_ohv_test_p_wtd,
                                   lr_tfidf_test_p_wtd,
                                   svc_test_p_wtd,
                                   mnb_test_p_wtd],
          'Recall_weighted': [dummy_mf_test_r_wtd,
                                 dummy_strf_test_r_wtd,
                                 lr_ohv_test_r_wtd,
                                 lr_tfidf_test_r_wtd,
                                 svc_test_r_wtd,
                                 mnb_test_r_wtd],
             'F1_weighted':[dummy_mf_test_f1_wtd ,
                                 dummy_strf_test_f1_wtd,
                                 lr_ohv_test_f1_wtd ,
                                 lr_tfidf_test_f1_wtd ,
                                 svc_test_f1_wtd,
                                 mnb_test_f1_wtd
                                 ],
               'Accuracy':[dummy_mf_test_acc,
                                 dummy_strf_test_acc,
                                 lr_ohv_test_acc,
                                 lr_tfidf_test_acc ,
                                 svc_test_acc,
                                 mnb_ohv_test_acc ]}
           }


train_index = ['DummyClassifier_MostFrequent_Train',
         'DummyClassifier_Stratified_Train',
         'LogisticRegression_OneHot_Train',
         'LogisticRegression_TF-IDF_Train',
         'SVC_OneHot_Train',
         'MultinomialNB_OneHot_Train',
         ]
test_index = [
      'DummyClassifier_MostFrequent_Test',
      'DummyClassifier_Stratified_Test',
      'LogisticRegression_OneHot_Test',
      'LogisticRegression_TF-IDF_Test',
      'SVC_OneHot_Test',
      'MultinomialNB_OneHot_Test',
      ]
train_metrics_df = pd.DataFrame(data=train_metrics).round(3)
train_metrics_df.index = train_index
display(train_metrics_df.style.highlight_max(color = 'Red', axis = 0,subset=['F1_macro']))
#print("-----------------------------------------------------------------------------------------
--------------------------------")
#print("-----------------------------------------------------------------------------------------
--------------------------------")
test_metrics_df = pd.DataFrame(data=test_metrics).round(3)
test_metrics_df.index = test_index
test_metrics_df
display(test_metrics_df.style.highlight_max(color = 'Red', axis = 0,subset=['F1_macro']))
```

	Precision_macro	Recall_macro	F1_macro	Precision_weighted	Recall_weighted	F1_weighted	Ac
DummyClassifier_MostFrequent_Train	0.200000	0.127000	0.155000	1.000000	0.633000	0.775000	0.6
DummyClassifier_Stratified_Train	0.201000	0.201000	0.201000	0.479000	0.477000	0.478000	0.4
LogisticRegression_OneHot_Train	0.845000	0.982000	0.901000	0.967000	0.963000	0.964000	0.9
LogisticRegression_TF-IDF_Train	0.418000	0.909000	0.473000	0.901000	0.817000	0.844000	0.
SVC_OneHot_Train	0.423000	0.750000	0.454000	0.931000	0.862000	0.887000	0.8
MultinomialNB_OneHot_Train	0.384000	0.730000	0.407000	0.893000	0.818000	0.847000	0.

	Precision_macro	Recall_macro	F1_macro	Precision_weighted	Recall_weighted	F1_weighted	Acc
DummyClassifier_MostFrequent_Test	0.200000	0.125000	0.154000	1.000000	0.626000	0.770000	0.6
DummyClassifier_Stratified_Test	0.217000	0.218000	0.217000	0.495000	0.487000	0.491000	0.4
LogisticRegression_OneHot_Test	0.441000	0.635000	0.489000	0.786000	0.748000	0.762000	0.7
LogisticRegression_TF-IDF_Test	0.330000	0.574000	0.356000	0.852000	0.740000	0.779000	0.7
SVC_OneHot_Test	0.288000	0.458000	0.287000	0.875000	0.729000	0.782000	0.7
MultinomialNB_OneHot_Test	0.264000	0.337000	0.259000	0.830000	0.684000	0.738000	0.6

In [14]:
```
#best classifier : bar plot production
labels = ['negative', 'neutral', 'positive', 'very negative',
       'very positive']
f1_macro = f1_score(lr_model.predict(test_features_ohv), test_labels_enc, labels =
[0,1,2,3,4],average=None)
plt.bar(np.arange(5),f1_macro)
plt.xticks(np.arange(5),labels, rotation = 30)
plt.title("F1_Macro score for each class with LR+One_Hot")
plt.xlabel("Sentiment polarity labels")
plt.ylabel("F1_Macro score")
plt.grid(axis='y')
```



QUESTION 2

Baseline model from Q1

In [15]:
```
tfidf = TfidfVectorizer(tokenizer=tokenize_normalize)

tfidf.fit(train_data['body'])

train_ft_q2 = tfidf.transform(train_data['body'])
valid_ft_q2 = tfidf.transform(validation_data['body'])
test_ft_q2 = tfidf.transform(test_data['body'])

lr = LogisticRegression()
lr.fit(train_ft_q2,train_labels_enc)
evaluation_summary_q2q3("Baseline(TFIDF + LR) on Train:",lr.predict(train_ft_q2),train_labels_enc)
evaluation_summary_q2q3("Baseline(TFIDF + LR) on Validation:",lr.predict(valid_ft_q2),validation_labels_enc)
```

```
'Baseline(TFIDF + LR) on Train:' has Acc=0.817 P=0.418 R=0.909 F1=0.473
'Baseline(TFIDF + LR) on Validation:' has Acc=0.733 P=0.325 R=0.572 F1=0.351
```

Range of hyperparameter set to perform GridSearch

Create custom CV with the combination of Train and Valid set

In [16]:
```
sublinear = {'tfidf__sublinear_tf': [True, True]}
max_features ={'tfidf__max_features' : [100,500,1000,1000,10000]}
lr_c = {'logreg__C' : [1e-3,1e+1,1e+2,1e+3,1e+4]}
max_iter = {'logreg__max_iter':[100,500,1000,2500,5000]}

train_val = train_data['body'].append(validation_data['body'])
train_val_labels= np.append(train_labels_enc,validation_labels_enc)
idx_train = [i for i in range(len(train_labels_enc))]
idx_train_val = [i for i in range(len(train_labels_enc),len(train_val_labels))]
custom_cv = [(np.asarray(idx_train),np.asarray(idx_train_val))]
```

LogisticRegression OPTIMISATION for C

In [17]:
```
from sklearn.model_selection import GridSearchCV
pred_pipeline= Pipeline([
                ('tfidf',TfidfVectorizer(tokenizer = tokenize_normalize)),
                ('logreg', LogisticRegression())
])

grid_search_clf_c=
GridSearchCV(pred_pipeline,param_grid=lr_c,cv=custom_cv,n_jobs=-1,verbose=2,scoring='f1_macro')

grid_search_clf_c.fit(train_val,train_val_labels)
lr_c_score = grid_search_clf_c.best_score_
lr_c_bp = grid_search_clf_c.best_params_
print("Best score: %0.3f" % lr_c_score)
print("Best parameters set:",lr_c_bp )
```

```
Fitting 1 folds for each of 5 candidates, totalling 5 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:  7.9min remaining:    0.0s
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:  7.9min finished
Best score: 0.510
Best parameters set: {'logreg__C': 10000.0}
```

LogisticRegression OPTIMISATION for max_iter (WITH C OPTIMISED)

In [18]:
```
pred_pipeline = Pipeline([
                ('tfidf',TfidfVectorizer(tokenizer = tokenize_normalize)),
                ('logreg', LogisticRegression(C=10000.0))
])

grid_search_clf_maxiter=
GridSearchCV(pred_pipeline,param_grid=max_iter,cv=custom_cv,n_jobs=-1,verbose=2,scoring='f1_macro'


grid_search_clf_maxiter.fit(train_val,train_val_labels)
lr_maxiter_score = grid_search_clf_maxiter.best_score_
lr_maxiter_bp = grid_search_clf_maxiter.best_params_
print("Best score: %0.3f" % lr_maxiter_score)
print("Best parameters set:",lr_maxiter_bp )
```

```
Fitting 1 folds for each of 5 candidates, totalling 5 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:  9.1min remaining:    0.0s
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:  9.1min finished
Best score: 0.510
Best parameters set: {'logreg__max_iter': 100}
```

TFidf Vectorizer OPTIMISATION for sublinear (WITH C+MAX_ITER OPTIMISED)

In [19]:
```
pred_pipeline = Pipeline([
                ('tfidf',TfidfVectorizer(tokenizer=tokenize_normalize)),
                ('logreg', LogisticRegression(max_iter=100, C=10000.0))
                ])
grid_search_vec_sub = GridSearchCV(pred_pipeline,param_grid = sublinear, cv=custom_cv,
n_jobs=-1,verbose=2 ,scoring = 'f1_macro')
grid_search_vec_sub.fit(train_val, train_val_labels)
tfidf_sub_score = grid_search_vec_sub.best_score_
tfidf_sub_bp = grid_search_vec_sub.best_params_

print("Best score: %0.3f" % tfidf_sub_score)
print("Best parameters set:",tfidf_sub_bp)
```

```
Fitting 1 folds for each of 2 candidates, totalling 2 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 out of   2 | elapsed:  3.0min finished
Best score: 0.528
Best parameters set: {'tfidf__sublinear_tf': True}
```

Tfidf Vectorizer OPTIMISATION for max_features (WITH C+MAX_ITER+SUBLINEAR_TF OPTIMISED)

In [20]:
```
pred_pipeline = Pipeline([
                ('tfidf',TfidfVectorizer(tokenizer=tokenize_normalize,sublinear_tf=True)),
                ('logreg', LogisticRegression(max_iter=100, C=10000.0))
                ])
grid_search_vec_mf = GridSearchCV(pred_pipeline,param_grid = max_features, cv=custom_cv,
n_jobs=-1,verbose=2 ,scoring = 'f1_macro')
grid_search_vec_mf.fit(train_val, train_val_labels)
tfidf_mf_score = grid_search_vec_mf.best_score_
tfidf_mf_bp = grid_search_vec_mf.best_params_

print("Best score: %0.3f" % tfidf_mf_score)
print("Best parameters set:",tfidf_mf_bp)
```

```
Fitting 1 folds for each of 5 candidates, totalling 5 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:  7.8min remaining:    0.0s
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:  7.8min finished
Best score: 0.539
Best parameters set: {'tfidf__max_features': 1000}
```

In [21]:
```
tune_results = [[lr_c_bp.values(), lr_c_score],
                [lr_maxiter_bp.values(), lr_maxiter_score],
                [tfidf_sub_bp.values() ,tfidf_sub_score],
                [tfidf_mf_bp.values() ,tfidf_mf_score],]

index= ['LogReg-C',
        'LogReg-max_iter & (C tuned)',
        'Tfidf-sublinear_tf & (max_iter & C tuned)',
        'Tfidf-max_features & (max_iter & C & sublinear_tf tuned)',

      ]


tune_results_df = pd.DataFrame(data=tune_results, columns=['Tuned Values',
'F1_score'])#.set_index(index)#,lr_c_score])#, columns=['Tuned Parameters', 'F1 score'])
tune_results_df.index = index
tune_results_df
```

Out[21]:

	Tuned Values	F1_score
LogReg-C	(10000.0)	0.509566
LogReg-max_iter & (C tuned)	(100)	0.509566
Tfidf-sublinear_tf & (max_iter & C tuned)	(True)	0.527930
Tfidf-max_features & (max_iter & C & sublinear_tf tuned)	(1000)	0.539068

In [22]:
```
prec_bl, rec_bl,acc_bl, f1_bl = evaluation_summary_q2q3("Baseline(TFIDF + LR) on Test:",lr.predict(test_ft_q2),test_labels_enc,flag=1)
pipeline_optimised = Pipeline([
                              ('tfidf',TfidfVectorizer(tokenizer=tokenize_normalize,
                                                       sublinear_tf=True, max_features=1000)),
                              ('logreg', LogisticRegression(max_iter=100, C=10000.0))
])
pipeline_optimised.fit(train_data['body'],train_labels_enc)
test_predictions_final = pipeline_optimised.predict(test_data['body'])
prec_opt, rec_opt,acc_opt, f1_opt = evaluation_summary_q2q3("Optimised(TFIDF + LR) on Test:",test_predictions_final,test_labels_enc,flag=1)

test_results = {'Precision(macro)':[prec_bl,prec_opt,],
                'Recall(macro)':[rec_bl,rec_opt],
                'Accuracy':[acc_bl,acc_opt],
                'F1_score(macro)':[f1_bl,f1_opt]}
index=['Baseline', 'Optimized']
test_results_df = pd.DataFrame(data=test_results).round(3)
test_results_df.index=index
test_results_df
```

```
'Baseline(TFIDF + LR) on Test:' has Acc=0.740 P=0.330 R=0.574 F1=0.356
'Optimised(TFIDF + LR) on Test:' has Acc=0.759 P=0.485 R=0.550 F1=0.512
```

Out[22]:

	Precision(macro)	Recall(macro)	Accuracy	F1_score(macro)
Baseline	0.330	0.574	0.740	0.356
Optimized	0.485	0.550	0.759	0.512

Q2 - Error Analysis

In [23]:
```
#confusion matrix
conf_matrix = confusion_matrix(test_labels_enc,test_predictions_final,labels=[0,1,2,3,4])
pd.DataFrame(data=conf_matrix, columns = encoder.categories_, index=encoder.categories_)
```

Out[23]:

	negative	neutral	positive	very negative	very positive
negative	102	166	7	4	3
neutral	103	2181	205	8	17
positive	7	345	729	1	20
very negative	9	17	0	6	0
very positive	0	18	38	0	30

Eli5 library investigation

In [24]:
```
print(encoder.categories_)
explain_linear_classifier_weights(pipeline_optimised[1],pipeline_optimised[0])
```

```
[array(['negative', 'neutral', 'positive', 'very negative',
       'very positive'], dtype=object)]
```

Out[24]:

y=0 top features		y=1 top features		y=2 top features		y=3 top features		y=4 top features	
Weight?	Feature	Weight?	Feature	Weight?	Feature	Weight?	Feature	Weight?	Feature
+19.611	wrong	+11.064	due	+13.119	kind	+15.757	hate	+22.694	best
+17.540	bad	+8.318	total	+13.022	good	+15.670	stupid	+18.080	great
+15.592	game	+7.776	itself	+12.237	many	+14.734	worst	+17.187	lol
+14.216	crazy	+7.104	new	+11.984	thanks	+12.601	terrible	+13.257	perfect
+13.955	mean	+6.868	back	+11.828	most	+8.238	base	+12.129	win
+13.740	sorry	+6.741	corsair	+11.538	first	+7.138	stuff	+11.783	good
+13.385	expensive	+6.464	half	… 569 more positive …		+7.004	fucking	+11.522	happy
+12.952	weird	+6.096	wow	… 412 more negative …		+6.077	removed	+11.088	d
+11.983	worse	+6.048	personal	-11.351	fucking	+5.876	doing	+11.023	awesome
… 458 more positive …		+6.040	spent	-11.648	game	+5.873	movies	+10.493	nice
… 523 more negative …		+6.001	short	-13.537	small	+5.771	state	+8.025	better
-11.909	more	+5.992	other	-13.973	base	+5.513	oh	+7.784	more
-12.911	love	+5.926	slightly	-14.388	crazy	+5.422	job	+7.634	love
-13.940	kind	… 580 more positive …		-14.540	expensive	+5.327	comments	… 277 more positive …	
-14.188	better	… 401 more negative …		-14.628	worse	+5.219	budget	… 704 more negative …	
-14.720	many	-5.933	favorite	-14.666	bad	+5.209	night	-7.449	they
-14.803	nice	-6.442	great	-16.018	terrible	… 205 more positive …		-7.604	other
-16.270	good	-6.682	win	-16.474	sorry	… 776 more negative …		-8.272	game
-17.296	most	-7.175	lol	-16.518	stupid	-5.260	actually	-8.449	pretty
-17.468	lol	-8.063	happy	-18.045	worst	-5.603	good	-8.614	far
-17.591	great	-8.579	perfect	-18.750	hate	-6.105	really	-8.702	bad
-20.517	best	-10.516	best	-18.762	wrong	-6.566	think	-10.193	mean

In [25]:
```
test_predictions_final[0] #predictions on test set
test_data_copy = test_data.copy()
test_data_copy['Predictions'] =
(encoder.inverse_transform(test_predictions_final.reshape(-1,1)))
miss_clf = test_data_copy[(test_data_copy['Predictions'] !=
test_data_copy['sentiment.polarity'])]
```

In [26]:
```
print("body: ",miss_clf['body'][0])
print("\nsentiment.polarity: ",miss_clf['sentiment.polarity'][0])
print("\nprediction: ",miss_clf['Predictions'][0])
show_prediction(pipeline_optimised[1],miss_clf['body']
[0],vec=pipeline_optimised[0],show_feature_values=True)
```

```
body:  Was watching a VOD from last years DreamHack Winter. The first map in the finals (Naama vs Mana) was Lost Temple.. spawning close ground positions..

It's a great sign that shows how much the game is evolving imo. Has anyone else noticed stuff like this?

sentiment.polarity:  neutral

prediction:  positive
```

Out[26]:

y=0 (probability 0.000, score -3.337) top features			y=1 (probability 0.380, score 3.691) top features			y=2 (probability 0.615, score 4.171) top features			y=3 (probability 0.004, score -0.788) top features		
Contribution?	Feature	Value	Contribution?	Feature	Value	Contribution?	Feature	Value	Contribution?	Feature	Value
+2.687	game	0.172	+3.944	<BIAS>	1.000	+2.045	first	0.177	+1.546	stuff	0.217
+0.886	ground	0.274	+1.010	last	0.207	+1.879	great	0.191	+1.136	watching	0.252
+0.821	else	0.209	+0.858	first	0.177	+1.183	much	0.157	+0.917	close	0.228
+0.672	lost	0.256	+0.446	game	0.172	+1.149	<BIAS>	1.000	+0.322	that	0.092
+0.545	last	0.207	+0.403	much	0.157	+0.540	shows	0.265	+0.299	game	0.172
+0.292	a	0.132	+0.283	shows	0.265	+0.392	mana	0.243	+0.257	how	0.152
+0.218	in	0.098	+0.228	lost	0.256	+0.262	anyone	0.210	+0.248	lost	0.256
+0.122	watching	0.252	+0.103	like	0.126	+0.188	a	0.132	+0.236	this	0.116
+0.116	has	0.150	+0.092	else	0.209	+0.163	else	0.209	+0.197	mana	0.243
+0.106	is	0.092	+0.088	from	0.141	+0.159	has	0.150	+0.187	from	0.141
+0.054	it	0.083	+0.084	the	0.149	+0.129	the	0.149	+0.161	was	0.212
+0.038	the	0.149	+0.063	in	0.098	+0.123	years	0.191	+0.087	anyone	0.210
-0.010	anyone	0.210	+0.058	a	0.132	+0.090	last	0.207	+0.059	in	0.098
-0.011	mana	0.243	+0.044	has	0.150	+0.061	in	0.098	+0.031	is	0.092
-0.023	like	0.126	+0.031	vs	0.256	+0.053	like	0.126	+0.023	has	0.150
-0.040	that	0.092	-0.028	it	0.083	+0.031	from	0.141	-0.006	the	0.149
-0.055	how	0.152	-0.030	close	0.228	+0.010	this	0.116	-0.086	shows	0.265
-0.059	from	0.141	-0.073	this	0.116	-0.010	it	0.083	-0.098	ground	0.274
-0.083	vs	0.256	-0.085	years	0.191	-0.062	is	0.092	-0.126	vs	0.256
-0.173	this	0.116	-0.092	how	0.152	-0.094	vs	0.256	-0.137	like	0.126
-0.278	<BIAS>	1.000	-0.136	anyone	0.210	-0.115	was	0.212	-0.161	much	0.157
-0.319	was	0.212	-0.150	is	0.092	-0.116	close	0.228	-0.199	it	0.083
-0.354	years	0.191	-0.162	that	0.092	-0.131	stuff	0.217	-0.318	a	0.132
-0.357	shows	0.265	-0.278	was	0.212	-0.136	watching	0.252	-0.334	years	0.191
-0.387	stuff	0.217	-0.300	ground	0.274	-0.185	that	0.092	-0.342	last	0.207
-0.851	close	0.228	-0.331	mana	0.243	-0.193	how	0.152	-0.352	first	0.177
-1.582	much	0.157	-0.478	stuff	0.217	-0.395	ground	0.274	-0.400	else	0.209
-1.950	first	0.177	-0.670	watching	0.252	-0.842	lost	0.256	-0.741	great	0.191
-3.362	great	0.191	-1.231	great	0.191	-2.007	game	0.172	-3.192	<BIAS>	1.000

In [27]:
```
#find missclasifications
miss_clf,index = [],[]
idx = 0
for i in zip(test_predictions_final,test_labels_enc):
  if (i[0]!=i[1]):
      miss_clf.append(i)
      index.append(idx)
  idx+=1
miss_clf = np.array(miss_clf)
miss_clf_df = (pd.DataFrame({'Index':index,
                              'Predictions':miss_clf[:,0],
                              'True Labels':miss_clf[:,1]})).set_index('Index')
#function to print  missclasied samples
def confusion_prints(indeces):
  c=0
  for i in indeces:
    print(c,": ",test_data['body'][i],"\n")
    c+=1
```

In [28]:
```
print("Error Analysis for (true) class: negative\nHighest confusion with (predicted) class:neutral")
neg_neut = miss_clf_df[(miss_clf_df['True Labels']==0)&
(miss_clf_df['Predictions']==1)].sample(5).index.values
confusion_prints(neg_neut)
print("--------------------------------------------------")

print("Error Analysis for (true) class: neutral\nHighest confusion with (predicted) class:positive")
neut_pos = miss_clf_df[(miss_clf_df['True Labels']==1 )&
(miss_clf_df['Predictions']==2)].sample(5).index.values
confusion_prints(neut_pos)
print("--------------------------------------------------")

print("Error Analysis for (true) class: neutral\nHighest confusion with (predicted) class:negative")
neut_neg = miss_clf_df[(miss_clf_df['True Labels']==1 )&
(miss_clf_df['Predictions']==0)].sample(5).index.values
confusion_prints(neut_neg)
print("--------------------------------------------------")

print("Error Analysis for (true) class: positive\nHighest confusion with (predicted) class:neutral")
pos_neut = miss_clf_df[(miss_clf_df['True Labels']==2)&
(miss_clf_df['Predictions']==1)].sample(5).index.values
confusion_prints(pos_neut)
print("--------------------------------------------------")

print("Error Analysis for (true) class: positive\nHighest confusion with (predicted) class:very_positive")
pos_vpos = miss_clf_df[(miss_clf_df['True Labels']==2)&
(miss_clf_df['Predictions']==4)].sample(5).index.values
confusion_prints(pos_vpos)
print("--------------------------------------------------")

print("Error Analysis for (true) class: very negative\nHighest confusion with (predicted) class:neutral")
vneg_neut = miss_clf_df[(miss_clf_df['True Labels']==3)&
(miss_clf_df['Predictions']==1)].sample(3).index.values
confusion_prints(vneg_neut)
print("--------------------------------------------------")

print("Error Analysis for (true) class: very negative\nHighest confusion with (predicted) class:neutral")
vneg_neg = miss_clf_df[(miss_clf_df['True Labels']==3)&
(miss_clf_df['Predictions']==0)].sample(2).index.values
confusion_prints(vneg_neg)
print("--------------------------------------------------")

print("Error Analysis for (true) class: very positive\nHighest confusion with (predicted) class:positive")
vpos_pos = miss_clf_df[(miss_clf_df['True Labels']==4 )&
(miss_clf_df['Predictions']==2)].sample(5).index.values
confusion_prints(vpos_pos)
print("--------------------------------------------------")

print("Error Analysis for (true) class: very positive\nHighest confusion with (predicted) class:neutral")
vpos_neut = miss_clf_df[(miss_clf_df['True Labels']==4 )&
(miss_clf_df['Predictions']==1)].sample(5).index.values
confusion_prints(vpos_neut)
print("--------------------------------------------------")
```

```
Error Analysis for (true) class: negative
Highest confusion with (predicted) class:neutral
0 :  Sadly I'm in the UK :(

1 :  Hey! I'm really sorry to hear about this. I'll ask around tonight and hopefully I can get you some information.

2 :  When a society establishes laws that enforce evil, then the members of that society perpetuate evil.

#Ergo, society has made you a worse person.

So, is religion really any worse than any social institution?

3 :  Yeah, that happens a lot. While you're undetected, Ellie will also remain undetected which is how it should be! It might be kind of annoying when she awkwardly runs out but at least she's not able to give away your position.

4 :  Subreddits that are racist, which is far worse than hating fatties, such as /kiketown haven't been banned yet, merely quarantined.

--------------------------------------------------
Error Analysis for (true) class: neutral
Highest confusion with (predicted) class:positive
0 :  That is one awesome explanation, many thanks, I'll read the article later tonight since I'm on my phone now, wish I had gold to give you :(

1 :  I snapped a bit at some poor bloke in a recent post who seems to be following a trend of posts lately that advertise the good deeds they have done.

I remember studying various outlooks on this at uni in Philosophy but i'm curious as to what YOU think - does it matter if someone does a good deed because they like the praise they get from people? Is a good act intrinsically better if you don't seek recognition from it or is it all the same in the end?

2 :  Yeah I can see how that would get complicated quickly. Thanks for you help!

3 :  False reports are worth nothing in the system. You're horrendously overreacting.

4 :  Personally I love metal Gear, even the hugely laboured cut scenes.  COD, it depends on the actual game.  Gears I enjoyed, once I got used to the fact that the controls were kinda clunky.  I'd played Uncharted on the ps3, before playing it, and there were a lot of similarities in the cover and shooting mechanics, but uncharted just handled so much more smoothly.  I never got Halo, though.  What makes it your choice?

--------------------------------------------------
Error Analysis for (true) class: neutral
Highest confusion with (predicted) class:negative
0 :  Depends on whether certain people's dead bodies are...ahem..available.

You won't be there to judge me.

1 :  My question is who thought it would be a good idea to make every CC remove his stars? Diana doesn't have her Moonsilver Blade disabled for being silenced, Singed doesn't lose his poison for being stunned.

It feels absolutely terrible to be hit by something random, even if you weren't the target, and be completely shut down because of it.

2 :  A face mage with ice block?

3 :  I am not even able to fukin reconnect into my bot game QQ


4 :  Why hello there. I didn't see you come in. I appear to have a conundrum.

I'm trying to make an instant messaging type of program using Java. I have the client set up and the server almost set up. I'm just having trouble getting the clients IP address. This is throwing the exception:

    MyStreamSocket myDataSocket = new
        MyStreamSocket(myConnectionSocket.accept());

    String ip = myDataSocket.getInetAddress().getHostName();
    //it's declaring the "ip" String that hits the error

Also, could this have anything to do with it being "localhost" as I only have one machine at the moment?

****EDIT****
Had the wrong code in. Was just experimenting with some stuff just to try to stumble upon the answer:
    //String ip = myDataSocket.getInetAddress()..getHostName();
***********

The error is a NullPointerException

--------------------------------------------------
Error Analysis for (true) class: positive
Highest confusion with (predicted) class:neutral
0 :  priest needs 1 more card in hand

1 :  No op just wants to do his The Meek Shall Inherit quest.

2 :  Probably.  Heh.  You can't blame me for trying to be a ray of sunshine though.  :)

3 :  One more question, I think I'm going to get the z313 (I found some for $30 new on Ebay), and I was wondering what would be the best location for the sub. I have a little shelf under my desk, about a foot off the ground, do you think that would be a good place or should I just put it on the desk behind/next to my monitor?

4 :  Hello, I need some advice. I am vaping on .2 sub ohm coils on a mod that goes up to 54 watts. I have been vaping on 30 watts. I have noticed the juice spits into my mouth a lot and will build up brown gunk on my top and bottom 2 front teeth. It is very embarrassing! I feel like I am drinking the juice some times. Should I turn up the wattage? By the way, I clean my tank very often.

--------------------------------------------------
Error Analysis for (true) class: positive
Highest confusion with (predicted) class:very_positive
0 :  i play all the game and i win it gaves me 15 lp:D

1 :  Welcome to the wonderful world of Tame Impala. Fucking incredible band. My favorite.

2 :  Depends on where you live, but a retail store like Best Buy or Fry's should have them kicking around.

3 :  Looks as good as can be for the price. Good luck, brother.

4 :  Is Zhonyas a good defensive for adc corki? It give him more ap,armor, and a good active against assassins. What do you think?

--------------------------------------------------
Error Analysis for (true) class: very negative
Highest confusion with (predicted) class:neutral
0 :  Energyzer Byunny ran out of steam:(

1 :  ....I thought something was up, figured it was just in my head that they were sitting there for too long...Well, now I feel like a moron.

2 :  That was indeed insane. Mass vikings, tanks, turrets, pdd's, seekers with a dash of bc's and nukes.

--------------------------------------------------
Error Analysis for (true) class: very negative
Highest confusion with (predicted) class:neutral
0 :  Oh, so it doesnt? My life is a lie.
 Well, another reason to ban this stupid champ.

1 :  Shutter Island is one of the worst, non-scary soundtracks I have ever heard.

--------------------------------------------------
Error Analysis for (true) class: very positive
Highest confusion with (predicted) class:positive
0 :  I would get a modular psu and a Samsung 840 ssd IMO of course happy building!


1 :  Religion isn't about doing good. It is about moneyz while pretending to do good!

2 :  My best friend today just said he thought the movie was amazing I'm not joking

3 :  The irony. If you dont talk.. you get reported for refusing to communicate with the team. LOL

4 :  lol, isnt the Model M buckle springs?

--------------------------------------------------
Error Analysis for (true) class: very positive
Highest confusion with (predicted) class:neutral
0 :  Is Spartacus in it? If so start with that. Still one of the greatest sword and sandal epics ever made.

1 :  ["So brave!"](http://i.imgur.com/LRvdW.gif)

2 :  /u/Chainsaw__Monkey

I have you tagged on RES as Nightwing Expert. :P

3 :  This. Graphics that were breathtaking in their day will always be outdone by improvments in the field.

4 :  Ur silver aswell :P

--------------------------------------------------
```

QUESTION III

In [29]:
```
#selectors used in FeatureUnion, similar to lab 2 and creation of
#new one (numerical) in order to use numerical features in FeatureUnion
class ItemSelector(BaseEstimator, TransformerMixin):
    """For data grouped by feature, select subset of data at a provided key.    """

    def __init__(self, key):
        self.key = key

    def fit(self, x, y=None):
        return self

    def transform(self, data_dict):
        return data_dict[self.key]

class NumericalItemSelector(BaseEstimator, TransformerMixin):
    """For data grouped by feature, select subset of data at a provided key.    """

    def __init__(self, key):
        self.key = key
    def fit(self, x, y=None):
        return self

    def transform(self, data_dict):
        return data_dict.loc[:,[self.key]]
```

Vectorizer investigation

In [30]:
```
vecs = [
        CountVectorizer(tokenizer=tokenize_normalize,binary=True),
        CountVectorizer(tokenizer=tokenize_normalize),
        CountVectorizer(tokenizer=tokenize_normalize,stop_words = stop_words,binary=True),
        CountVectorizer(tokenizer=tokenize_normalize,stop_words = stop_words),
        TfidfVectorizer(tokenizer=tokenize_normalize),
        TfidfVectorizer(tokenizer=tokenize_normalize,binary=True),
        TfidfVectorizer(tokenizer=tokenize_normalize,stop_words = stop_words),
        TfidfVectorizer(tokenizer=tokenize_normalize,stop_words = stop_words,binary=True)
]
scores=[]
for v in vecs:
  prediction_pipeline = Pipeline([
                                  ('Selector',ItemSelector(key='body')),
                                  ('Vectorizer',v),
                                  ('Logistic Regression',LogisticRegression(C=10000)) #use the optimised LR
  ])
  prediction_pipeline.fit(train_data,train_labels_enc)
  preds = prediction_pipeline.predict(validation_data)
  scores.append(evaluation_summary_q2q3("",preds,validation_labels_enc,flag=1))
```

```
'' has Acc=0.732 P=0.555 R=0.558 F1=0.550
'' has Acc=0.749 P=0.541 R=0.592 F1=0.562
'' has Acc=0.709 P=0.536 R=0.519 F1=0.517
'' has Acc=0.708 P=0.540 R=0.520 F1=0.522
'' has Acc=0.736 P=0.493 R=0.552 F1=0.512
'' has Acc=0.743 P=0.524 R=0.568 F1=0.529
'' has Acc=0.719 P=0.506 R=0.520 F1=0.506
'' has Acc=0.712 P=0.495 R=0.518 F1=0.497
```

In [34]:
```
columns=['Precision(macro)','Recall(macro)','Accuracy','F1(macro)']
index = ['CV-OH','CV','CV-OH-StopWords','CV-StopWords','TFIDF','TFIDF-OH','TFIDF-StopWords','TFIDF-OH-StopWords']
vecs_df = pd.DataFrame(data=np.array(scores),columns=columns,index=index).round(3)
print('\t\tDIFFERENT VECTORISATION APPROACHES')
display(vecs_df.style.highlight_max(color = 'Red', axis = 0,subset=['F1(macro)']))
```

```
		DIFFERENT VECTORISATION APPROACHES
```

	Precision(macro)	Recall(macro)	Accuracy	F1(macro)
CV-OH	0.555000	0.558000	0.732000	0.550000
CV	0.541000	0.592000	0.749000	0.562000
CV-OH-StopWords	0.536000	0.519000	0.709000	0.517000
CV-StopWords	0.540000	0.520000	0.708000	0.522000
TFIDF	0.493000	0.552000	0.736000	0.512000
TFIDF-OH	0.524000	0.568000	0.743000	0.529000
TFIDF-StopWords	0.506000	0.520000	0.719000	0.506000
TFIDF-OH-StopWords	0.495000	0.518000	0.712000	0.497000

Optimisation for the best performing vectorizer for the proposed params in Q2 (max_features)

In [35]:
```
vec_optim_pipeline = Pipeline([
                              ('CV',CountVectorizer(tokenizer=tokenize_normalize)),
                              ('lr',LogisticRegression(C=10000))])

params_vec = {'CV__max_features':[100,500,1000,1000,10000]}
gs = GridSearchCV(vec_optim_pipeline,param_grid=params_vec,cv=custom_cv,n_jobs=-1,scoring
='f1_macro',verbose=2 )
gs.fit(train_val,train_val_labels)
gs_score = gs.best_score_
gs_sub_bp = gs.best_params_

print("Best score: %0.3f" % gs_score)
print("Best parameters set:",gs_sub_bp)
```

```
Fitting 1 folds for each of 5 candidates, totalling 5 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:  7.9min remaining:    0.0s
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:  7.9min finished
Best score: 0.559
Best parameters set: {'CV__max_features': 10000}
```

BASELINES COMPARISSON

In [36]:
```
q2_bl = Pipeline([
                ('tfidf',TfidfVectorizer(tokenizer=tokenize_normalize,
                                         sublinear_tf=True, max_features=1000)),
                ('logreg', LogisticRegression(max_iter=100, C=10000.0))])

q2_bl.fit(train_data['body'],train_labels_enc)
p_bl, r_bl, acc_bl, f1_bl = evaluation_summary_q2q3("Q2 Baseline on Validation:",q2_bl.predict(validation_data['body']),validation_labels_enc,flag=1)

vec_opt = Pipeline([
              ('CV',CountVectorizer(tokenizer=tokenize_normalize, max_features=10000)),
              ('logreg', LogisticRegression(max_iter=100, C=10000.0))])

vec_opt.fit(train_data['body'],train_labels_enc)
p_vo, r_vo, acc_vo, f1_vo = evaluation_summary_q2q3("Vectorizer Optimised on Validation:",vec_opt.predict(validation_data['body']),validation_labels_enc,flag=1)
```

```
'Q2 Baseline on Validation:' has Acc=0.757 P=0.507 R=0.584 F1=0.539
'Vectorizer Optimised on Validation:' has Acc=0.744 P=0.540 R=0.589 F1=0.561
```

SINGLE FEATURE: Subreddit

In [37]:
```python
#new vectorizer + subreddit
ppl1= Pipeline([
        ('union', FeatureUnion(
          transformer_list=[
            ('body', Pipeline([
                ('selector', ItemSelector(key='body')),
                ('CV', CountVectorizer(tokenizer = tokenize_normalize,max_features=10000)),
                  ])),
            ('subreddit', Pipeline([
              ('selector', ItemSelector(key='subreddit')),
              ('CV',CountVectorizer(tokenizer=tokenize_normalize, max_features=10000)),
              ])),
        ])),
        ('logreg',LogisticRegression(C=10000.0, max_iter=100))
        ])
ppl1.fit(train_data,train_labels_enc)
pred1 = ppl1.predict(validation_data)
p1, r1,acc1, f11 = evaluation_summary_q2q3("Body + Subreddit
",pred1,validation_labels_enc,flag=1)
```

```
'Body + Subreddit ' has Acc=0.739 P=0.506 R=0.552 F1=0.524
```

SINGLE FEATURE: Majority_type

In [38]:
```python
#new vectorizer + majority_type
#new vectorizer + subreddit
ppl2= Pipeline([
        ('union', FeatureUnion(
          transformer_list=[
            ('body', Pipeline([
                ('selector', ItemSelector(key='body')),
                ('CV', CountVectorizer(tokenizer = tokenize_normalize,max_features=10000)),
                  ])),
            ('majority_type', Pipeline([
              ('selector', ItemSelector(key='majority_type')),
             ('CV',CountVectorizer(tokenizer=tokenize_normalize, max_features=10000)),
              ])),
        ])),
        ('logreg',LogisticRegression(C=10000.0, max_iter=100))
        ])
ppl2.fit(train_data,train_labels_enc)
pred2 = ppl2.predict(validation_data)
p2, r2,acc2, f12 = evaluation_summary_q2q3("Body + Majority_type
",pred2,validation_labels_enc,flag=1)
```

```
'Body + Majority_type ' has Acc=0.746 P=0.522 R=0.559 F1=0.537
```

SINGLE FEATURE: Title

In [39]:
```python
#new vectorizer + title
#new vectorizer + subreddit
ppl3= Pipeline([
        ('union', FeatureUnion(
          transformer_list=[
            ('body', Pipeline([
                ('selector', ItemSelector(key='body')),
                ('CV', CountVectorizer(tokenizer = tokenize_normalize,max_features=10000)),
                  ])),
            ('title', Pipeline([
              ('selector', ItemSelector(key='title')),
             ('CV',CountVectorizer(tokenizer=tokenize_normalize, max_features=10000)),
              ])),
        ])),
        ('logreg',LogisticRegression(C=10000.0, max_iter=100))
        ])
ppl3.fit(train_data,train_labels_enc)
pred3 = ppl3.predict(validation_data)
p3, r3,acc3, f13 = evaluation_summary_q2q3("Body + Title",pred3,validation_labels_enc,flag=1)
```

```
'Body + Title' has Acc=0.731 P=0.429 R=0.494 F1=0.454
```

In [39]:

Create new features with Vader

In [41]:
```python
vader = SentimentIntensityAnalyzer()

vs_train = []
for i in range(len(train_data['body'])):
  scores = vader.polarity_scores(train_data['body'][i])
  vs_train.append(scores['compound'])

vs_valid = []
for i in range(len(validation_data['body'])):
  scores = vader.polarity_scores(validation_data['body'][i])
  vs_valid.append(scores['compound'])

vs_test = []
for i in range(len(test_data['body'])):
  scores = vader.polarity_scores(test_data['body'][i])
  vs_test.append(scores['compound'])

train_data_new = train_data.copy()
valid_data_new = validation_data.copy()
test_data_new = test_data.copy()
train_data_new['vader'] = np.zeros_like(len(vs_train))
valid_data_new['vader'] = np.zeros_like(len(vs_valid))
test_data_new['vader'] = np.zeros_like(len(vs_test))

vs_train_df = pd.DataFrame(vs_train)
vs_valid_df = pd.DataFrame(vs_valid)
vs_test_df = pd.DataFrame(vs_test)
```

In [42]:
```python
#features have values in range [-1,1], thus i split the range in 5 parts (For all sets)
train_data_new['vader'][(vs_train_df[0]<=(-0.6))] =0
train_data_new['vader'][((vs_train_df[0]>(-0.6)) & (vs_train_df[0]<=(-0.2)))] = 1
train_data_new['vader'][((vs_train_df[0]>(-0.2)) & (vs_train_df[0]<=(0.2)))] = 2
train_data_new['vader'][((vs_train_df[0]>(0.2)) & (vs_train_df[0]<=(0.6)))]= 3
train_data_new['vader'][(vs_train_df[0]>(0.6))] = 4

test_data_new['vader'][(vs_test_df[0]<=(-0.6))] =0
test_data_new['vader'][((vs_test_df[0]>(-0.6)) & (vs_test_df[0]<=(-0.2)))] = 1
test_data_new['vader'][((vs_test_df[0]>(-0.2)) & (vs_test_df[0]<=(0.2)))] = 2
test_data_new['vader'][((vs_test_df[0]>(0.2)) & (vs_test_df[0]<=(0.6)))]= 3
test_data_new['vader'][(vs_test_df[0]>(0.6))] = 4

valid_data_new['vader'][(vs_valid_df[0]<=(-0.6))] =0
valid_data_new['vader'][((vs_valid_df[0]>(-0.6)) & (vs_valid_df[0]<=(-0.2)))] = 1
valid_data_new['vader'][((vs_valid_df[0]>(-0.2)) & (vs_valid_df[0]<=(0.2)))] = 2
valid_data_new['vader'][((vs_valid_df[0]>(0.2)) & (vs_valid_df[0]<=(0.6)))]= 3
valid_data_new['vader'][(vs_valid_df[0]>(0.6))] = 4
```

SINGLE FEATURE: Vader

In [43]:
```python
#new vectorizer + vader
ppl4= Pipeline([
        ('union', FeatureUnion(
          transformer_list=[
            ('body', Pipeline([
                ('selector', ItemSelector(key='body')),
                ('CV', CountVectorizer(tokenizer = tokenize_normalize,max_features=10000)),
                  ])),
            ('vader', Pipeline([
              ('selector', NumericalItemSelector(key='vader')),
              ])),
        ])),
        ('logreg',LogisticRegression(C=10000.0, max_iter=100))
        ])
ppl4.fit(train_data_new, train_labels_enc)
pred4 = ppl4.predict(valid_data_new)
p4, r4,acc4, f14 = evaluation_summary_q2q3("Body + Vader",pred4,validation_labels_enc,flag=1)
```

```
'Body + Vader' has Acc=0.758 P=0.521 R=0.563 F1=0.539
```

Create new feature: body length

In [44]:
```python
#measure the len of each post, get statistics of it to split it (repeated for all sets)
train_body_length = train_data_new['body'].apply(len)
train_data_new['body_length'] =  train_body_length
train_descr= train_data_new['body_length'].describe()

valid_body_length = valid_data_new['body'].apply(len)
valid_data_new['body_length'] = valid_body_length
valid_descr = valid_data_new['body_length'].describe()

test_body_length = test_data_new['body'].apply(len)
test_data_new['body_length'] =  test_body_length
test_descr = test_data_new['body_length'].describe()

columns = ["Train Body Length","Test Body Length","Valid Body Length"]
descr = pd.concat((train_descr ,valid_descr,test_descr),axis=1)
descr.columns = columns
descr
```

Out[44]:

	Train Body Length	Test Body Length	Valid Body Length
count	12138.000000	3109.000000	4016.000000
mean	207.551738	205.288196	209.552789
std	374.641221	450.566575	355.041213
min	0.000000	0.000000	0.000000
25%	42.250000	44.000000	43.000000
50%	101.000000	103.000000	101.000000
75%	227.000000	222.000000	230.000000
max	11747.000000	10605.000000	5031.000000

In [45]:
```python
#split all sets based on individual the statistics showabove at 4 parts & assign categorize them
(0-1-2-3)
train_data_new['body_length'][train_data_new.body_length<=42] = 0
train_data_new['body_length'][((train_data_new.body_length>42) &
(train_data_new.body_length<=101))] = 1
train_data_new['body_length'][((train_data_new.body_length>101) &
(train_data_new.body_length<=227))] = 2
train_data_new['body_length'][train_data_new.body_length>227] = 3
train_data_new['body_length'] = ((np.array(train_data_new['body_length'])).reshape(-1,1))


valid_data_new['body_length'][valid_data_new.body_length<=44] = 0
valid_data_new['body_length'][((valid_data_new.body_length>44) &
(valid_data_new.body_length<=103))] = 1
valid_data_new['body_length'][((valid_data_new.body_length>103) &
(valid_data_new.body_length<=223))] = 2
valid_data_new['body_length'][valid_data_new.body_length>223] = 3
valid_data_new['body_length'] = (np.array(valid_data_new['body_length'])).reshape(-1,1)


test_data_new['body_length'][test_data_new.body_length<=43] =0
test_data_new['body_length'][((test_data_new.body_length>43) &
(test_data_new.body_length<=101))] = 1
test_data_new['body_length'][((test_data_new.body_length>101) &
(test_data_new.body_length<=230))] = 2
test_data_new['body_length'][test_data_new.body_length>230] =3
test_data_new['body_length'] = (np.array(test_data_new['body_length'])).reshape(-1,1)
```

SINGLE FEATURE: Body_length

In [46]:
```python
#new vectorizer + body_length
ppl5= Pipeline([
        ('union', FeatureUnion(
          transformer_list=[
            ('body', Pipeline([
                ('selector', ItemSelector(key='body')),
                ('CV', CountVectorizer(tokenizer = tokenize_normalize,max_features=10000)),
                  ])),
            ('body_length', Pipeline([
              ('selector', NumericalItemSelector(key='body_length')),
              ])),
        ])),
        ('logreg',LogisticRegression(C=10000.0, max_iter=100))
        ])
ppl5.fit(train_data_new,train_labels_enc)
pred5 = ppl5.predict(valid_data_new)
p5, r5,acc5, f15 = evaluation_summary_q2q3("Body +
Body_length",pred5,validation_labels_enc,flag=1)
```

```
'Body + Body_length' has Acc=0.751 P=0.542 R=0.613 F1=0.572
```

In [47]:
```python
#vec+ is_first_post
ppl6= Pipeline([
        ('union', FeatureUnion(
          transformer_list=[
            ('body', Pipeline([
                ('selector', ItemSelector(key='body')),
                ('CV', CountVectorizer(tokenizer = tokenize_normalize,max_features=10000)),
                  ])),
            ('is_first_post', Pipeline([
              ('selector', NumericalItemSelector(key='is_first_post')),
              ])),
        ])),
        ('logreg',LogisticRegression(C=10000.0, max_iter=100))
        ])
ppl6.fit(train_data,train_labels_enc)
pred6 = ppl6.predict(validation_data)
p6, r6,acc6, f16 = evaluation_summary_q2q3("Body +
is_first_post",pred6,validation_labels_enc,flag=1)
```

```
'Body + is_first_post' has Acc=0.747 P=0.537 R=0.588 F1=0.557
```

In [48]:
```python
#vec+ author
ppl7= Pipeline([
        ('union', FeatureUnion(
          transformer_list=[
            ('body', Pipeline([
                ('selector', ItemSelector(key='body')),
                ('CV', CountVectorizer(tokenizer = tokenize_normalize,max_features=10000)),
                  ])),
            ('author', Pipeline([
              ('selector', ItemSelector(key='author')),
              ('CV',CountVectorizer(tokenizer=tokenize_normalize, max_features=10000)),
              ])),
        ])),
        ('logreg',LogisticRegression(C=10000.0, max_iter=100))
        ])
ppl7.fit(train_data,train_labels_enc)
pred7 = ppl7.predict(validation_data)
p7, r7,acc7, f17 = evaluation_summary_q2q3("Body + author",pred7,validation_labels_enc,flag=1)
```

```
'Body + author' has Acc=0.752 P=0.516 R=0.582 F1=0.542
```

In [49]:
```python
pp=np.array((p_vo,p1,p2,p3,p4,p5,p6,p7))
rr=np.array((r_vo,r1,r2,r3,r4,r5,r6,p7))
accacc=np.array((acc_vo,acc1,acc2,acc3,acc4,acc5,acc6,acc7))
f1f1=np.array((f1_vo,f11,f12,f13,f14,f15,f16,f17))
columns=['Precision(macro)','Recall(macro)','F1(macro)','Accuracy']
index = ['Baseline: body','Add: subreddit', 'Add: majority_type','Add: title',
        'Add:body_length','Add:vader', 'Add: is_first_post','Add: author']
df_single_ft_vec =
pd.DataFrame(data=np.array((pp,rr,f1f1,accacc)).T,columns=columns,index=index).round(3)
print('\tCOMBINATION OF BODY WITH ONE EXTRA FEATURE , AND NEW VECTORIZER')
df_single_ft_vec
```

```
        COMBINATION OF BODY WITH ONE EXTRA FEATURE , AND NEW VECTORIZER
```

Out[49]:

	Precision(macro)	Recall(macro)	F1(macro)	Accuracy
Baseline: body	0.540	0.589	0.561	0.744
Add: subreddit	0.506	0.552	0.524	0.739
Add: majority_type	0.522	0.559	0.537	0.746
Add: title	0.429	0.494	0.454	0.731
Add:body_length	0.521	0.563	0.539	0.758
Add:vader	0.542	0.613	0.572	0.751
Add: is_first_post	0.537	0.588	0.557	0.747
Add: author	0.516	0.516	0.542	0.752

NEW VECTORIZER, WITH Body, Body_length and 1 More Feature

title: performed worse than others and dropped from further investigation

In [50]:
```python
#new vectorizer + vader + subreddit
pipe1= Pipeline([
        ('union', FeatureUnion(
          transformer_list=[
            ('body', Pipeline([
                ('selector', ItemSelector(key='body')),
                ('CV', CountVectorizer(tokenizer = tokenize_normalize,max_features=10000)),
                  ])),
            ('body_length', Pipeline([
              ('selector', NumericalItemSelector(key='body_length')),
              ])),
           ('subreddit', Pipeline([
              ('selector', ItemSelector(key='subreddit')),
              ('CV',CountVectorizer(tokenizer=tokenize_normalize, max_features=10000)),
              ]))
        ])),
        ('logreg',LogisticRegression(C=10000.0, max_iter=100))
        ])
pipe1.fit(train_data_new, train_labels_enc)
predp1 = pipe1.predict(valid_data_new)
p1n, r1n,acc1n, f1n = evaluation_summary_q2q3("Body + body_length+
Subreddit",predp1,validation_labels_enc,flag=1)
```

```
'Body + body_length+ Subreddit' has Acc=0.744 P=0.532 R=0.567 F1=0.547
```

In [51]:
```python
#new vectorizer + body_length + majority_type
pipe2= Pipeline([
        ('union', FeatureUnion(
          transformer_list=[
            ('body', Pipeline([
                ('selector', ItemSelector(key='body')),
                ('CV', CountVectorizer(tokenizer = tokenize_normalize,max_features=10000)),
                  ])),
            ('body_length', Pipeline([
              ('selector', NumericalItemSelector(key='body_length')),
              ])),
           ('majority_type', Pipeline([
              ('selector', ItemSelector(key='majority_type')),
              ('CV',CountVectorizer(tokenizer=tokenize_normalize, max_features=10000)),
              ]))
        ])),
        ('logreg',LogisticRegression(C=10000.0, max_iter=100))
        ])
pipe2.fit(train_data_new, train_labels_enc)
predp2 = pipe2.predict(valid_data_new)
p2n, r2n,acc2n, f2n = evaluation_summary_q2q3("Body + body_length+
Majority_type",predp2,validation_labels_enc,flag=1)
```

```
'Body + body_length+ Majority_type' has Acc=0.746 P=0.526 R=0.572 F1=0.546
```

In [52]:
```python
#new vectorizer + vader + body_length
pipe3= Pipeline([
        ('union', FeatureUnion(
          transformer_list=[
            ('body', Pipeline([
                ('selector', ItemSelector(key='body')),
                ('CV', CountVectorizer(tokenizer = tokenize_normalize,max_features=10000)),
                  ])),
            ('vader', Pipeline([
              ('selector', NumericalItemSelector(key='vader')),
              ])),
           ('body_length', Pipeline([
              ('selector', NumericalItemSelector(key='body_length')),
              ]))
        ])),
        ('logreg',LogisticRegression(C=10000.0, max_iter=100))
        ])
pipe3.fit(train_data_new, train_labels_enc)
predp3 = pipe3.predict(valid_data_new)
p3n, r3n,acc3n, f3n = evaluation_summary_q2q3("Body + Vader+
body_length",predp3,validation_labels_enc,flag=1)
```

```
'Body + Vader+ body_length' has Acc=0.760 P=0.514 R=0.546 F1=0.528
```

In [53]:
```python
#new vectorizer + body_length + is_first_post
pipe4= Pipeline([
        ('union', FeatureUnion(
          transformer_list=[
            ('body', Pipeline([
                ('selector', ItemSelector(key='body')),
                ('CV', CountVectorizer(tokenizer = tokenize_normalize,max_features=10000)),
                  ])),
            ('body_length', Pipeline([
              ('selector', NumericalItemSelector(key='body_length')),
              ])),
           ('is_first_post', Pipeline([
              ('selector', NumericalItemSelector(key='is_first_post')),
              ]))
        ])),
        ('logreg',LogisticRegression(C=10000.0, max_iter=100))
        ])
pipe4.fit(train_data_new, train_labels_enc)
predp4 = pipe4.predict(valid_data_new)
p4n, r4n,acc4n, f4n = evaluation_summary_q2q3("Body + body_length+
is_first_post",predp4,validation_labels_enc,flag=1)
```

```
'Body + body_length+ is_first_post' has Acc=0.752 P=0.552 R=0.607 F1=0.576
```

In [54]:
```python
#new vectorizer + vader + author
pipe5= Pipeline([
        ('union', FeatureUnion(
          transformer_list=[
            ('body', Pipeline([
                ('selector', ItemSelector(key='body')),
                ('CV', CountVectorizer(tokenizer = tokenize_normalize,max_features=10000)),
                  ])),
            ('body_length', Pipeline([
              ('selector', NumericalItemSelector(key='body_length')),
              ])),
           ('author', Pipeline([
              ('selector', ItemSelector(key='author')),
              ('CV',CountVectorizer(tokenizer=tokenize_normalize, max_features=10000)),
              ]))
        ])),
        ('logreg',LogisticRegression(C=10000.0, max_iter=100))
        ])
pipe5.fit(train_data_new, train_labels_enc)
predp5 = pipe5.predict(valid_data_new)
p5n, r5n,acc5n, f5n = evaluation_summary_q2q3("Body + body_length+
author",predp5,validation_labels_enc,flag=1)
```

```
'Body + body_length+ author' has Acc=0.750 P=0.546 R=0.577 F1=0.559
```

In [55]:
```python
pp1=np.array((p_vo,p1n,p2n,p3n,p4n,p5n))
rr1=np.array((r_vo,r1n,r2n,r3n,r4n,r5n))
accacc1=np.array((acc_vo,acc1n,acc2n,acc3n,acc4n,acc5n))
f1f11=np.array((f1_vo,f1n,f2n,f3n,f4n,f5n))
columns=['Precision(macro)','Recall(macro)','F1(macro)','Accuracy']
index = ['Baseline: body','Add: body_length & subreddit', 'Add: body_length & majority_type',
        'Add:vader & body_length', 'Add: body_length & is_first_post','Add:body_length &
author']
df_double_ft_vec =
pd.DataFrame(data=np.array((pp1,rr1,f1f11,accacc1)).T,columns=columns,index=index).round(3)
print('\tCOMBINATION OF BODY WITH Body_length & 1 MORE FEATURE & NEW VECTORIZER')
df_double_ft_vec
```

```
        COMBINATION OF BODY WITH Body_length & 1 MORE FEATURE & NEW VECTORIZER
```

Out[55]:

	Precision(macro)	Recall(macro)	F1(macro)	Accuracy
Baseline: body	0.540	0.589	0.561	0.744
Add: body_length & subreddit	0.532	0.567	0.547	0.744
Add: body_length & majority_type	0.526	0.572	0.546	0.746
Add:vader & body_length	0.514	0.546	0.528	0.760
Add: body_length & is_first_post	0.552	0.607	0.576	0.752
Add:body_length & author	0.546	0.577	0.559	0.750

Final Model, evaluation on Test set & confusion matrix

Occam's razor "rule", the simplest model was choosen among the 2 that peformed better than baseline, as their performance difference was not that huge to choose the more complex model

In [56]:
```python
best_model1 = Pipeline([
        ('union', FeatureUnion(
          transformer_list=[
            ('body', Pipeline([
                ('selector', ItemSelector(key='body')),
                ('cv', CountVectorizer(tokenizer = tokenize_normalize, max_features=10000)),
                  ])),
            ('body_lenth', Pipeline([
              ('selector', NumericalItemSelector(key='body_length')),
              ]))
        ])),
        ('logreg',LogisticRegression(C=10000, max_iter=100))
        ])
best_model1.fit(train_data_new,train_labels_enc)
preds_best_model1 = best_model1.predict(valid_data_new)
p_bm1, r_bm1,acc_bm1, f1_bm1 = evaluation_summary_q2q3("Body+body_length, & new vectorisation:
",preds_best_model1,validation_labels_enc,flag=1)
```

```
'Body+body_length, & new vectorisation: ' has Acc=0.751 P=0.542 R=0.613 F1=0.572
```

In [57]:
```python
test_preds_best_model1 = best_model1.predict(test_data_new)
p_bm_test1, r_bm_test1,acc_bm_test1, f1_bm_test1 = evaluation_summary_q2q3("Body+body_length, &
new vectorisation: ",test_preds_best_model1,test_labels_enc,flag=1)
```

```
'Body+body_length, & new vectorisation: ' has Acc=0.757 P=0.532 R=0.571 F1=0.549
```

In [58]:
```python
p_final=np.array((prec_opt,p_bm_test1))
r_final=np.array((rec_opt,r_bm_test1))
acc_final=np.array((acc_opt,acc_bm_test1))
f_final=np.array((f1_opt,f1_bm_test1))
columns=['Precision(macro)','Recall(macro)','F1(macro)','Accuracy']
index = ['Baseline-Optimised on Q2',
        'Best Model']

df_final =
pd.DataFrame(data=np.array((p_final,r_final,f_final,acc_final)).T,columns=columns,index=index).ro

print("\t\t\tTEST SET COMPARISON")
df_final.style.highlight_max(color='red')
```

```
                        TEST SET COMPARISON
```

Out[58]:

	Precision(macro)	Recall(macro)	F1(macro)	Accuracy
Baseline-Optimised on Q2	0.485000	0.550000	0.512000	0.759000
Best Model	0.532000	0.571000	0.549000	0.757000

In [59]:
```python
conf_matrix_final1= confusion_matrix(test_labels_enc,test_preds_best_model1,labels=[0,1,2,3,4])
conf_matrix_final_df1 =
pd.DataFrame(data=conf_matrix_final1,columns=encoder.categories_,index=encoder.categories_)
conf_matrix_final_df1
```

Out[59]:

	negative	neutral	positive	very negative	very positive
negative	88	172	14	6	2
neutral	102	2116	270	9	17
positive	6	282	791	1	22
very negative	6	12	1	13	0
very positive	0	11	42	0	33