In [1]:

```
import os
import pandas as pd
import numpy as np
import spacy
import matplotlib.pyplot as plt
import seaborn as sns
import timeit
import scattertext as st
import collections
from IPython.display import HTML, IFrame
from textblob import TextBlob
from w3lib.html import remove_tags
from wordcloud import WordCloud
from tqdm import tqdm_notebook
from sklearn.preprocessing import MinMaxScaler
from sklearn.manifold import TSNE
from sklearn.decomposition import KernelPCA
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
import logging
logger = logging.getLogger("spacy")
logger.setLevel(logging.ERROR)
```

In [3]:

```
data1 = pd.read_csv("drugsComTrain_raw.tsv",sep='\t',index_col=0)
data2 = pd.read_csv("drugsComTest_raw.tsv",sep='\t',index_col=0)
data = pd.concat([data1,data2])
data.head()
```

Out[3]:

	drugName	condition	review	rating	date	usefulCount
206461	Valsartan	Left Ventricular Dysfunction	"It has no side effect, I take it in combinati...	9.0	May 20, 2012	27
95260	Guanfacine	ADHD	"My son is halfway through his fourth week of ...	8.0	April 27, 2010	192
92703	Lybrel	Birth Control	"I used to take another oral contraceptive, wh...	5.0	December 14, 2009	17
138000	Ortho Evra	Birth Control	"This is my first time using any form of birth...	8.0	November 3, 2015	10
35696	Buprenorphine / naloxone	Opiate Dependence	"Suboxone has completely turned my life around...	9.0	November 27, 2016	37

In [4]:

```
df = data[['review','rating']]
df.head()
```

Out[4]:

	review	rating
206461	"It has no side effect, I take it in combinati...	9.0
95260	"My son is halfway through his fourth week of ...	8.0
92703	"I used to take another oral contraceptive, wh...	5.0
138000	"This is my first time using any form of birth...	8.0
35696	"Suboxone has completely turned my life around...	9.0

In [5]:

```
def train_val_test_split(df, val_size, test_size, random_state=0):
    """Split data frame into 3 (train/val/test) sets or into 2 (train/val) se

    If you want to split into two datasets, set test_size = 0.

    Parameters
    ----------
    df : pandas.DataFrame
        Pandas.DataFrame to split.
    val_size : float
        Fraction of dataset to include in validation set. Should be from rang
    test_size : float
        Fraction of dataset to include in test set. Should be from range <0.0
    random_state: int, optional (default=0)
        The seed used by the random number generator.

    Returns
    -------
    train: pandas.DataFrame
       Training set.
    val: pandas.DataFrame
       Validation set.
    test: pandas.DataFrame
       Test set.

    Raises
    ------
    AssertionError
        If the val_size and test_size sum is greater or equal 1 or the negati

    """
    assert (val_size + test_size) < 1, 'Validation size and test size sum is 
    assert val_size >= 0 and test_size >= 0, 'Negative size is not accepted'
    train, val, test = np.split(df.sample(frac=1, random_state=random_state),
                                [int((1-(val_size+test_size))*len(df)), int((
    return train, val, test
```

In [6]:

```
train1_set, dataset, test_set = train_val_test_split(df, val_size=0.30, test_
```

In [7]:

```
print(dataset.shape)
print(test_set.shape)
```

```
(64519, 2)
(21507, 2)
```

In [8]:

```
dataset.rating.value_counts()
```

Out[8]:

```
10.0    20529
9.0     10964
1.0      8621
8.0      7416
7.0      3751
5.0      3223
2.0      2802
3.0      2594
6.0      2585
4.0      2034
Name: rating, dtype: int64
```

In [9]:

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 64519 entries, 171080 to 91547
Data columns (total 2 columns):
review    64519 non-null object
rating    64519 non-null float64
dtypes: float64(1), object(1)
memory usage: 1.5+ MB
```

In [10]:

```
# Get indices of duplicate data (excluding first occurrence)
duplicate_indices = dataset.loc[dataset.duplicated(keep='first')].index

# Count and print the number of duplicates
print('Number of duplicates in the dataset: {}'.format(dataset.loc[duplicate_
```

```
Number of duplicates in the dataset: 7835
```

In [11]:

```
dataset.loc[duplicate_indices, :].head()
```

Out[11]:

	review	rating
100335	"I took this BC for a year. My main reason for...	10.0
51961	"I&#039;m 28 with two kids I just started adip...	10.0
159453	"I started taking this drug for chronic daily ...	10.0
72	"I have been on the depo shot for about 5 year...	8.0
48359	"If I hadn&#039;t made the decision to begin m...	10.0

In [12]:

```
# Drop duplicates
dataset.drop_duplicates(keep='first', inplace=True)
```

In [13]:

```
# Print the shape of dataset after removing duplicate rows
print('Dataset shape after removing duplicates: {}'.format(dataset.shape))
```

```
Dataset shape after removing duplicates: (56684, 2)
```

In [14]:

```
dataset = dataset.dropna()
```

In [15]:

```
# Save raw dataset as a CSV file
dataset.to_csv(os.path.join('dataset/drugreview_raw.csv'), index=False)
```

In [16]:

```
path = 'dataset/'

# Load raw dataset from CSV file
dataset = pd.read_csv(os.path.join(path, 'drugreview_raw.csv'))
```

In [17]:

```
def polarity(text):
    """Calculate the polarity score of the input text.

    """
    return TextBlob(text).sentiment.polarity
```

In [18]:

```
def subjectivity(text):
    """Calculate the subjectivity score of the input text.

    """
    return TextBlob(text).sentiment.subjectivity
```

In [19]:

```
def pos(df, batch_size, n_threads, required_tags):
    """Count the number of peculiar POS tags in data series of strings.

    Parameters
    ----------
    df : pandas.Series
        Pandas.Series containing strings to process.
    batch_size: int
        Size of text batch (recommended to be the power of 2).
    n_threads: int
        Number of threads in multiprocessing.
    required_tags: list
        List containing spacy's POS tags to count.

    Returns
    -------
    pandas.DataFrame
       DataFrame of a shape (index, len(required_tags)).

    """
    # Add index column to reviews frame and change column order
    reviews = df.reset_index(drop=False)[['review', 'index']]
    # Convert dataframe to list of tuples (review, index)
    review_list = list(zip(*[reviews[c].values.tolist() for c in reviews]))
    # Create empty dictionary
    review_dict = collections.defaultdict(dict)

    for doc, context in list(nlp.pipe(review_list, as_tuples=True, batch_size
        review_dict[context] = {}
        for token in doc:
            pos = token.pos_
            if pos in required_tags:
                review_dict[context].setdefault(pos, 0)
                review_dict[context][pos] = review_dict[context][pos] + 1
    # Transpose data frame to shape (index, tags)
    return pd.DataFrame(review_dict).transpose()
```

In [20]:

```
def pos2(df, batch_size, n_threads, required_tags):
    """Count the number of peculiar POS tags in data series of strings.

    Parameters
    ----------
    df : pandas.Series
        Pandas.Series containing strings to process.
    batch_size: int
        Size of text batch (recommended to be the power of 2).
    n_threads: int
        Number of threads in multiprocessing.
    required_tags: list
        List containing spacy's POS tags to count.

    Returns
    -------
    pandas.DataFrame
       DataFrame of a shape (index, len(required_tags)).

    """
    # Create empty dictionary
    review_dict = collections.defaultdict(dict)
    for i, doc in enumerate(nlp.pipe(df, batch_size=batch_size)):
         for token in doc:
            pos = token.pos_
            if pos in required_tags:
                review_dict[i].setdefault(pos, 0)
                review_dict[i][pos] = review_dict[i][pos] + 1
    # Transpose data frame to shape (index, tags)
    return pd.DataFrame(review_dict).transpose()
```

In [21]:

```
def pos3(df, required_tags):
    """Count the number of peculiar POS tags in data series of strings.

    Parameters
    ----------
    df : pandas.Series
        Pandas.Series containing strings to process.
    required_tags: list
        List containing spacy's POS tags to count.

    Returns
    -------
    pandas.DataFrame
       DataFrame of a shape (index, len(required_tags)).

    """
    pos_list = []
    for i in range(df.shape[0]):
        doc = nlp(df[i])
        pos_dict = {}
        for token in doc:
            pos = token.pos_
            if pos in required_tags:
                pos_dict.setdefault(pos, 0)
                pos_dict[pos] = pos_dict[pos] + 1
        pos_list.append(pos_dict)
    return pd.DataFrame(pos_list)
```

In [22]:

```
# Load language model and disable unnecessary components of processing pipeli
nlp = spacy.load('en_core_web_sm', disable = ['ner', 'parser', 'textcat', '..
required_tags = ['PROPN', 'PUNCT', 'NOUN', 'ADJ', 'VERB']

# Define batch_size and n_threads
batch_size = 512
n_threads = 2

# Test the processing time on a part of the dataset, given batch_size and n_t
start_time = timeit.default_timer()
print('Start processing 1000 examples using batch_size: {} and n_threads: {}'
pos(dataset.loc[:1000, 'review'], required_tags=required_tags, batch_size=bat
print('Function 1 processing time: {:.2f} sec'.format(timeit.default_timer()
```

```
Start processing 1000 examples using batch_size: 512 and n_threads: 2
Function 1 processing time: 5.62 sec
```

In [23]:

```
# Define batch_size and n_threads
batch_size = 512
n_threads = 2

# Test the processing time on a part of the dataset, given batch_size and n_t
start_time = timeit.default_timer()
print('Start processing 1000 examples using batch_size: {} and n_threads: {}'
pos2(dataset.loc[:1000, 'review'],  batch_size=batch_size, n_threads=2,requir
print('Function 2 processing time: {:.2f} sec'.format(timeit.default_timer()
```

```
Start processing 1000 examples using batch_size: 512 and n_threads: 2
Function 2 processing time: 4.54 sec
```

In [24]:

```
# Test the processing time on a part of the dataset, given batch_size and n_t
start_time = timeit.default_timer()
print('Start processing 1000 examples')
pos3(dataset.loc[:1000, 'review'], required_tags=required_tags)
print('Function 3 processing time: {:.2f} sec'.format(timeit.default_timer()
```

```
Start processing 1000 examples
Function 3 processing time: 5.00 sec
```

In [25]:

```python
def extract_features(df, batch_size, n_threads, required_tags):
    """Extract the following features from the data frame's 'review' column:
    polarity, subjectivity, word_count, UPPERCASE, DIGITS, and POS tags speci

    Convert extracted features to int16 or float16 data types.

    Parameters
    ----------
    df : pandas.DataFrame
        Pandas.DataFrame containing 'review' column to which extraction will
    batch_size: int
        Size of text batch (recommended to be the power of 2).
    n_threads: int
        Number of threads in multiprocessing.
    required_tags: list
        List containing spacy's POS tags to count.

    Returns
    -------
    pandas.DataFrame
       Concatenation of the original data frame and data frame containing ext

    """
    # Calculate polarity
    df['polarity'] = df.review.apply(polarity).astype('float16')
    # Calculate subjectivity
    df['subjectivity'] = df.review.apply(subjectivity).astype('float16')
    # Calculate number of words in review
    df['word_count'] = df.review.apply(lambda text: len(text.split())).astype
    # Count number of uppercase words, then divide by word_count
    df['UPPERCASE'] = df.review.apply(lambda text: len([word for word in text
                                                        if word.isupper()]))/
    # Change data type to float16
    df.UPPERCASE = df.UPPERCASE.astype('float16')
    # Count number of digits, then divide by word_count
    df['DIGITS'] = df.review.apply(lambda text: len([word for word in text.sp
                                                     if word.isdigit()]))/df.
    # Change data type to float16
    df.DIGITS = df.DIGITS.astype('float16')
    # Perform part-of-speech taging
    pos_data = pos2(df.review, batch_size=batch_size, n_threads=n_threads, re
    # Divide POS tags count by word_count
    pos_data = pos_data.div(df.word_count, axis=0).astype('float16')
    # Concatenate pandas data frames horizontaly
    return pd.concat([df, pos_data], axis=1)
```

In [26]:

```
# Load language model and disable unnecessary components of processing pipeli
nlp = spacy.load('en_core_web_sm', disable = ['ner', 'parser', 'textcat', '..
required_tags = ['PROPN', 'PUNCT', 'NOUN', 'ADJ', 'VERB']

batch_size = 512
n_threads = 2

# Test the processing time on a part of the trainig set, given batch_size and
start_time = timeit.default_timer()
print('Start processing 1000 examples using batch_size: {} and n_threads: {}'
extract_features(dataset.loc[:1000, :], batch_size=batch_size, n_threads=n_th
print('Feature extraction function processing time: {:.2f} sec'.format(timeit
```

```
Start processing 1000 examples using batch_size: 512 and n_threads: 2
Feature extraction function processing time: 6.18 sec
```

In [27]:

```
from tqdm.notebook import tqdm
def split_extract_save(df, name, path, part_size, batch_size, n_threads, requ
    """Split data frame into chunks of size equal: part_size and perform feat
    Extract the following features from the data frame part's 'review' column
    UPPERCASE, DIGITS, and POS tags specified by required_tags.

    Parameters
    ----------
    df : pandas.DataFrame
        Pandas.DataFrame containing 'review' column to which extraction will 
    name : str
        Name of the CSV file to which export the data.
    path : str
        Absolute or relative path to directory where to save the data.
    part_size: int
        Size of the chunk to process (number of strings it contains).
    batch_size: int
        Size of text batch (recommended to be the power of 2).
    n_threads: int
        Number of threads in multiprocessing.
    required_tags: list
        List containing spacy's POS tags to count.
    nlp: spacy.lang.<language>
        Spacy language model (for example spacy.lang.en.English)

    Returns
    -------
    pandas.DataFrame
       Concatenation of the original data frame and data frame containing ext

    """
    if name not in os.listdir(path):
        dataset_parts = []
        N = int(len(df)/part_size)
        # Create list of dataframe chunks
        data_frames = [df.iloc[i*part_size:(i+1)*part_size].copy() for i in r
        # Process dataset partialy
        i = 0
        for frame in tqdm(data_frames):
            print(i)
            i += 1
            # Extract features from dataset chunk
            dataset_part = extract_features(frame, batch_size=batch_size, n_t
                                            required_tags=required_tags)
            dataset_parts.append(dataset_part)
            # Reload nlp
            nlp = spacy.load('en_core_web_sm', disable = ['ner', 'parser', 't

        # Concatenate all parts into one dataset
        dataset_feat = pd.concat(dataset_parts, axis=0, sort=False)
        # Replace missing values NaN with 0
        dataset_feat.fillna(0, inplace=True)
        # Convert label values to int16
        dataset_feat.rating = dataset_feat.rating.astype('int16')
        # Export data frame to CSV file
        dataset_feat.to_csv(path + name, index=False)
    else:
        print('File {} already exists in given directory.'.format(name))
```

In [29]:

```
# Define all required variables
nlp = spacy.load('en_core_web_sm', disable = ['ner', 'parser', 'textcat', '..
required_tags = ['PROPN', 'PUNCT', 'NOUN', 'ADJ', 'VERB']
batch_size = 512
n_threads = 2
part_size = 5000
path = os.path.join(os.getcwd(), 'dataset/datasets_feat/')
name = 'drugreview_feat.csv'

# Perform feature extraction and export resulted file into CSV
split_extract_save(dataset, name, path, part_size, batch_size, n_threads, req
```

```
0
1
2
3
4
5
6
7
8
9
10
11
```

In [30]:

```
# Dictionary of {column: dtype} pairs
col_types = {'review': str,'rating': np.int16, 'polarity': np.float16, 'subje
             'word_count': np.int16, 'UPPERCASE': np.float16, 'DIGITS': np.fl
             'VERB': np.float16, 'NOUN': np.float16, 'PUNCT': np.float16, 'AD

# Import dataset from the CSV file
dataset_feat = pd.read_csv('dataset/datasets_feat/drugreview_feat.csv', dtype
```

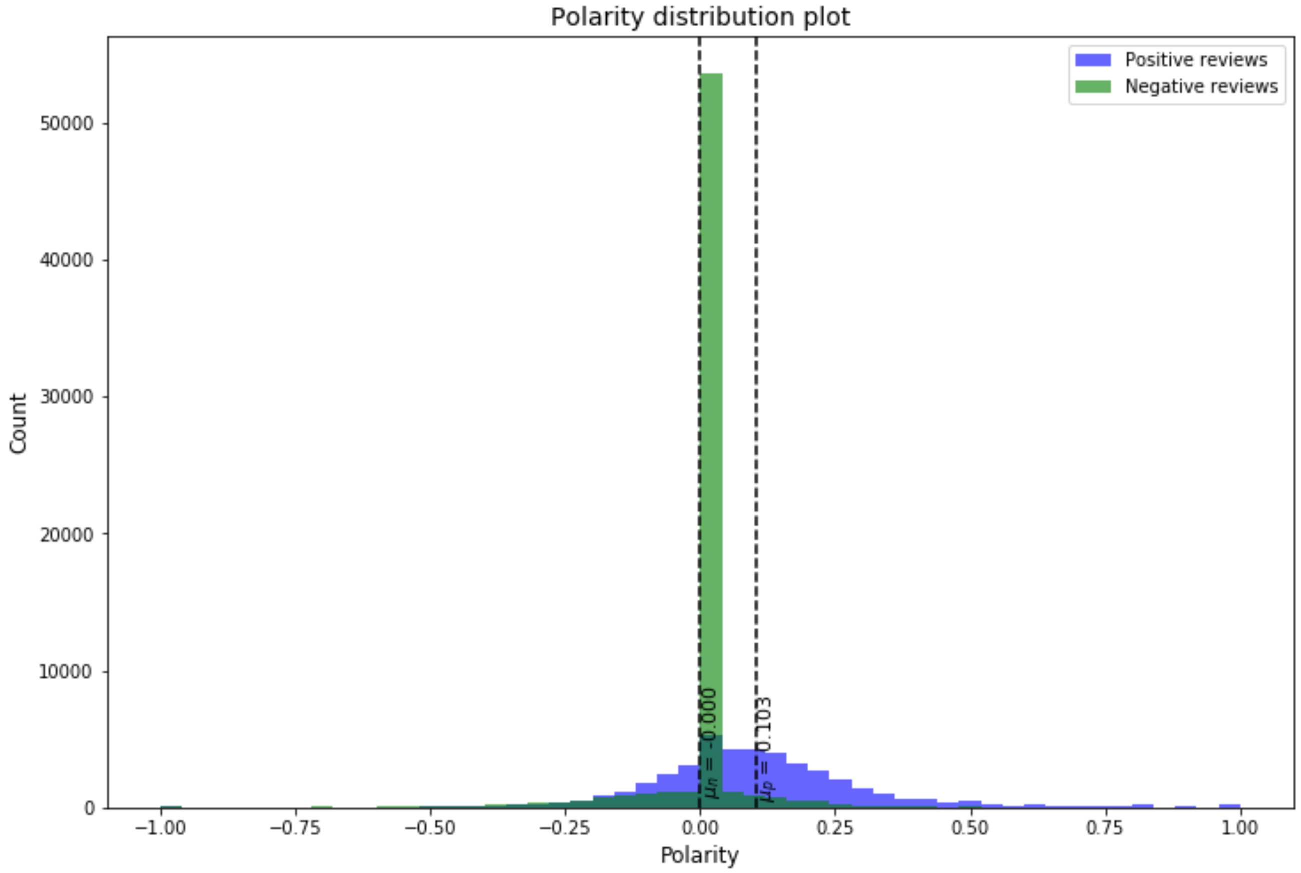
In [31]:

```
dataset_feat.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 108368 entries, 0 to 108367
Data columns (total 12 columns):
review          108368 non-null object
rating          108368 non-null int16
polarity        108368 non-null float16
subjectivity    108368 non-null float16
word_count      108368 non-null int16
UPPERCASE       108368 non-null float16
DIGITS          108368 non-null float16
PUNCT           108368 non-null float16
VERB            108368 non-null float16
PROPN           108368 non-null float16
NOUN            108368 non-null float16
ADJ             108368 non-null float16
dtypes: float16(9), int16(2), object(1)
memory usage: 3.1+ MB
```

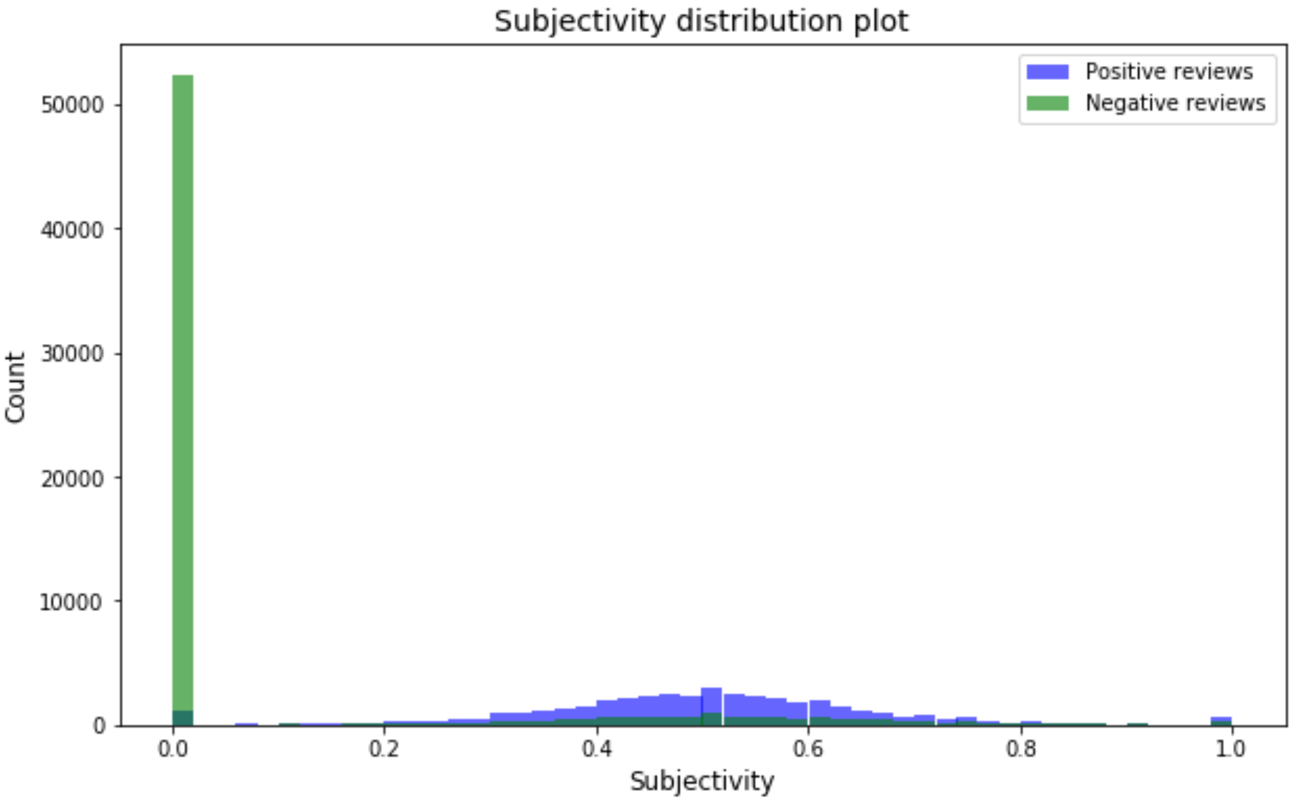
In [32]:

```
# Separate polarity score for positive and nagative reviews
pos_reviews_pol = dataset_feat.loc[dataset_feat.rating >= 5, 'polarity']
neg_reviews_pol = dataset_feat.loc[dataset_feat.rating < 5, 'polarity']

# Create a new figure
plt.figure(figsize=(12,8))

# Create a histogram of polarity for positive reviews (color=blue, transparen
plt.hist(pos_reviews_pol, bins=50, label='Positive reviews', alpha=0.6, color
# Create a histogram of polarity for negative reviews (color=green, transpare
plt.hist(neg_reviews_pol, bins=50, label='Negative reviews', alpha=0.6, color
# Create the title, horizontal axis label, vertical axis label and legend for
plt.title('Polarity distribution plot', size=14)
plt.xlabel('Polarity', size=12)
plt.ylabel('Count', size=12)
plt.legend(loc='upper right')

# Calculate the mean value of polarity for positive and negative reviews
pos_pol_mean = pos_reviews_pol.mean()
neg_pol_mean = neg_reviews_pol.mean()

# Add vertical lines that represent the average polarity of each class
plt.axvline(pos_pol_mean, c='k', linestyle='--', linewidth=1.5)
plt.axvline(neg_pol_mean, c='k', linestyle='--', linewidth=1.5)
# Add annotations
plt.text(pos_pol_mean, 1200, r'$\mu_p$ = {:.3f}'.format(pos_pol_mean), rotati
plt.text(neg_pol_mean, 1200, r'$\mu_n$ = {:.3f}'.format(neg_pol_mean), rotati

plt.show()
```



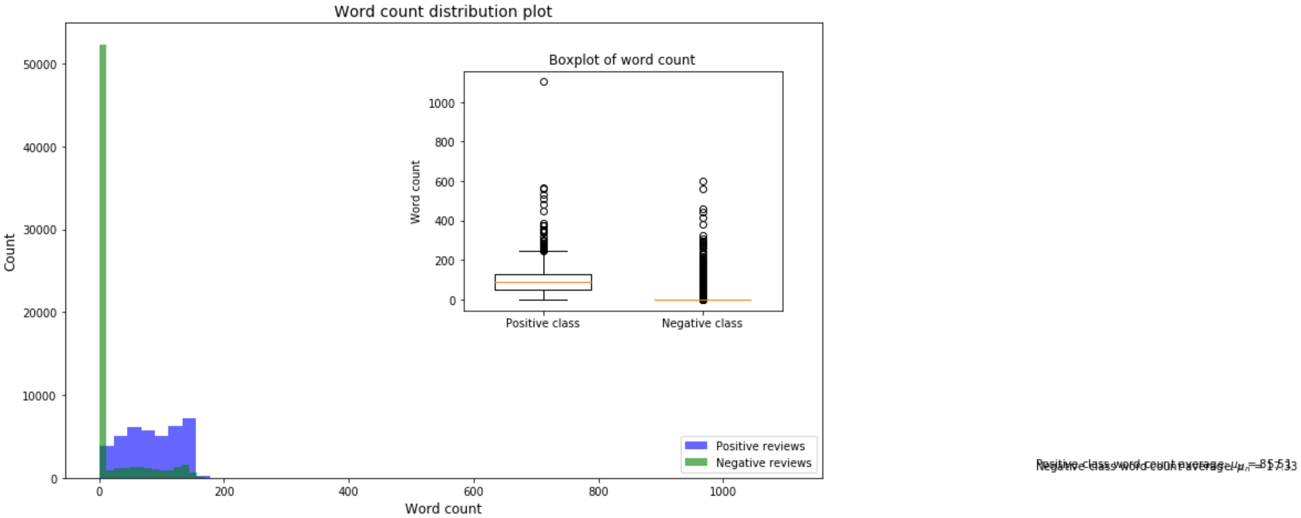
In [33]:

```
# Separate subjectivity score for positive and nagative reviews
pos_reviews_subj = dataset_feat.loc[dataset_feat.rating >=5, 'subjectivity']
neg_reviews_subj = dataset_feat.loc[dataset_feat.rating < 5, 'subjectivity']

# Create a new figure
plt.figure(figsize=(10,6))

# Create histograms of subjectivity for positive and negative reviews
plt.hist(pos_reviews_subj, bins=50, label='Positive reviews', alpha=0.6, colo
plt.hist(neg_reviews_subj, bins=50, label='Negative reviews', alpha=0.6, colo
plt.title('Subjectivity distribution plot', size=14)
plt.xlabel('Subjectivity', size=12)
plt.ylabel('Count', size=12)
plt.legend(loc='upper right')

plt.show()
```


Subjectivity distribution plot
Positive reviews
Negative reviews
Count
50000
40000
30000
20000
10000
0
0.0
0.2
0.4
0.6
0.8
1.0
Subjectivity

In [34]:

```
# Separate word count distributions for positive and nagative reviews
# Violinplots or boxplots better deal with numpy arrays
pos_reviews_w_count = np.array(dataset_feat.loc[dataset_feat.rating >+ 5, 'wo
neg_reviews_w_count = np.array(dataset_feat.loc[dataset_feat.rating < 5, 'wor

# Create a new figure instance
fig = plt.figure(figsize=(10,6))

# Add axes to the figure. Create the first main window
ax1 = fig.add_axes([0, 0, 0.95, 0.95]) # window coord: (left, bottom, width, 
ax1.hist(pos_reviews_w_count, bins=50, label='Positive reviews', alpha=0.6, c
ax1.hist(neg_reviews_w_count, bins=50, label='Negative reviews', alpha=0.6, c
# Create the title, horizontal axis label, vertical axis label and legend for
ax1.set_title('Word count distribution plot', size=14)
ax1.set_xlabel('Word count', size=12)
ax1.set_ylabel('Count', size=12)
ax1.legend(loc='lower right')
# Add descriptions
ax1.text(1500, 1200, r'Positive class word count average: $\mu_p$ = {:.2f}'.f
ax1.text(1500, 1000, r'Negative class word count average: $\mu_n$ = {:.2f}'.f

# Add axes to the figure. Create the second boxplots window
ax2 = fig.add_axes([0.5, 0.35, 0.40, 0.50]) # window coord: (left, bottom, wi
# Create boxplots
ax2.boxplot([pos_reviews_w_count, neg_reviews_w_count], widths=0.6)
ax2.set_title('Boxplot of word count')
ax2.set_ylabel('Word count')
# Set the x axis labels
ax2.set_xticks([1, 2])
ax2.set_xticklabels(['Positive class', 'Negative class'])

plt.show()
```



In [35]:

```
dataset_feat.groupby(by='rating').word_count.describe()
```

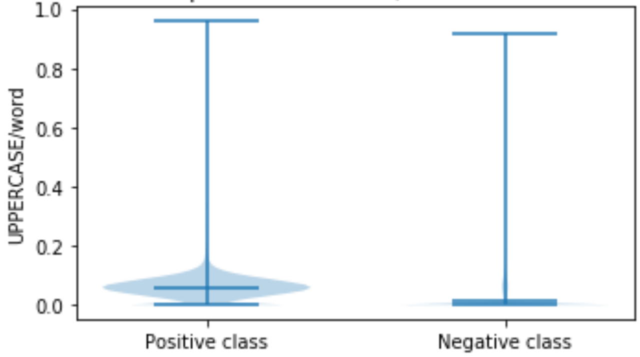
Out[35]:

	count	mean	std	min	25%	50%	75%	max
rating								
0	51684.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0
1	7592.0	78.323235	45.445657	1.0	41.0	73.0	119.0	601.0
2	2423.0	82.860091	42.835462	2.0	47.0	82.0	122.5	298.0
3	2281.0	83.859272	42.044149	2.0	49.0	83.0	124.0	216.0
4	1788.0	85.553691	42.035119	1.0	51.0	85.0	125.0	287.0
5	2825.0	88.042832	42.066123	2.0	53.0	89.0	127.0	375.0
6	2259.0	86.830899	42.652547	1.0	52.0	87.0	127.0	260.0
7	3330.0	89.367267	43.465768	1.0	54.0	91.0	129.0	566.0
8	6515.0	88.872141	43.677508	1.0	54.0	91.0	129.0	558.0
9	9633.0	88.069553	45.389299	1.0	52.0	90.0	129.0	1103.0
10	18038.0	82.061759	44.933570	1.0	45.0	80.0	124.0	532.0

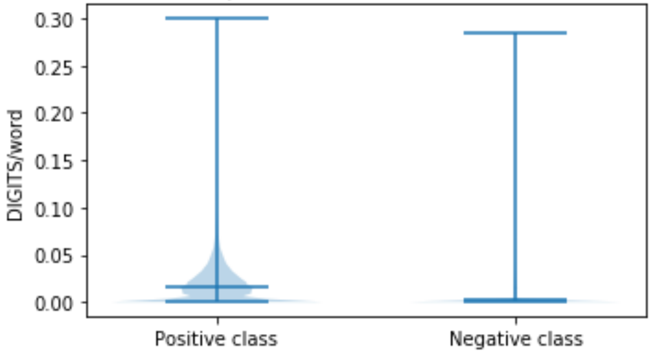
In [36]:

```python
# Create the figure and axes instances
fig, axes = plt.subplots(4, 2, figsize=(10,12), sharex=False)
# Take the last 7 columns labels from the data frame
data_labels = dataset_feat.columns[-7:]
data_idx = 0

# Iterate through the plots rows and columns
for row in range(4):
    for col in range(2):
        if data_idx <= 6:
            # Create the violinplot of given feature for positive and negativ
            axes[row, col].violinplot([np.array(dataset_feat.loc[dataset_feat
                                      np.array(dataset_feat.loc[dataset_feat.
                                      widths=0.7, showmeans=True)
            # Set the title and vertical axis labels
            axes[row, col].set_title('Violin plot of {}/word distribution'.fo
            axes[row, col].set_ylabel('{}/word'.format(data_labels[data_idx])
            # Set the x axis labels
            axes[row, col].set_xticks([1, 2])
            axes[row, col].set_xticklabels(['Positive class', 'Negative class
        else:
            # Delete unnecessary axes
            fig.delaxes(axes[row, col])
        data_idx += 1

# Automatically adjusts subplot params to fit in figure
plt.tight_layout()
```

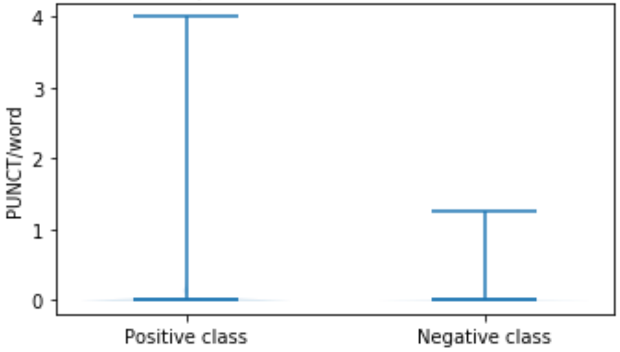
Violin plot of UPPERCASE/word distribution



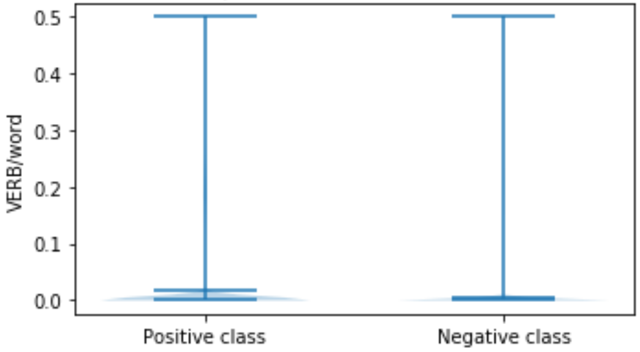
Violin plot of DIGITS/word distribution



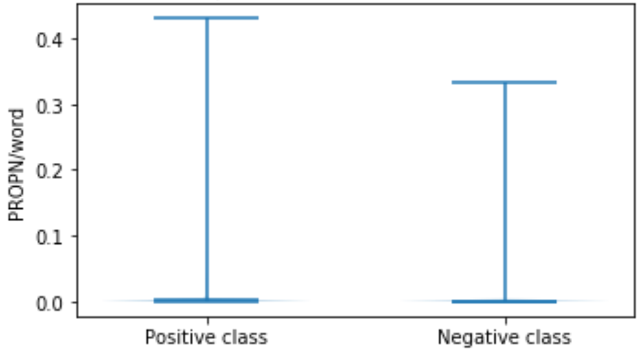
Violin plot of PUNCT/word distribution



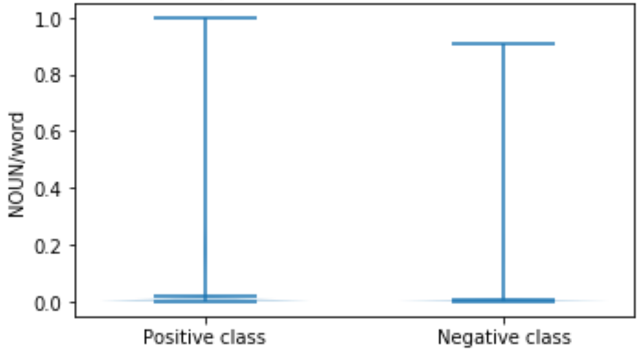
Violin plot of VERB/word distribution



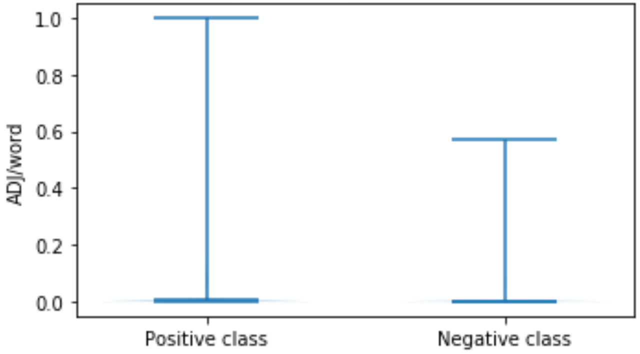
Violin plot of PROPN/word distribution



Violin plot of NOUN/word distribution



Violin plot of ADJ/word distribution

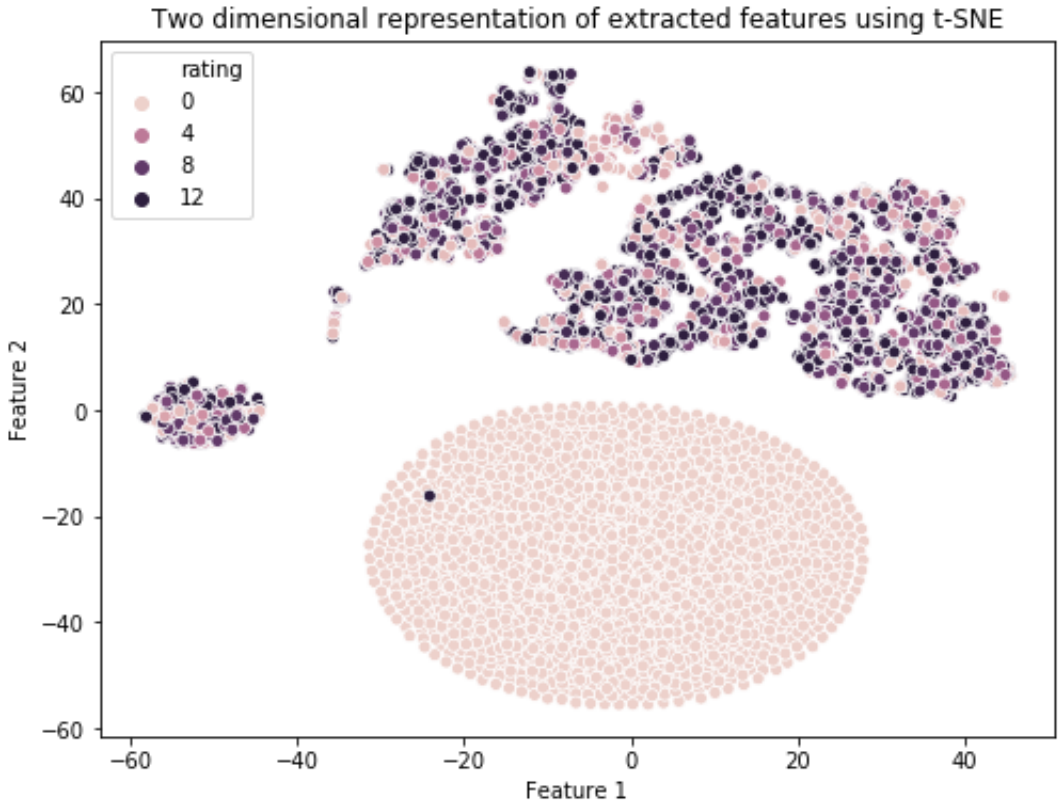


In [37]:

```
# Choose at random a sample of 10,000 examples to visualize
data_to_vis = dataset_feat.iloc[:, -11:].sample(n=10000)
feat_to_vis = data_to_vis.iloc[:, -10:]
label_to_vis = data_to_vis.iloc[:, 0]

# Perform MinMax feature scaling
feat_to_vis = MinMaxScaler().fit_transform(feat_to_vis)

# Perform dimensionality reduction using t-SNE
emb_vectors = TSNE(n_components=2, n_iter=1000).fit_transform(feat_to_vis)
```

In [38]:

```
#Visualize data in lower-dimensional space
plt.figure(figsize=(8,6))
# Create seaborn scatterplot
sns.scatterplot(x=emb_vectors[:, 0], y=emb_vectors[:, 1], hue=label_to_vis)
plt.title('Two dimensional representation of extracted features using t-SNE')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```

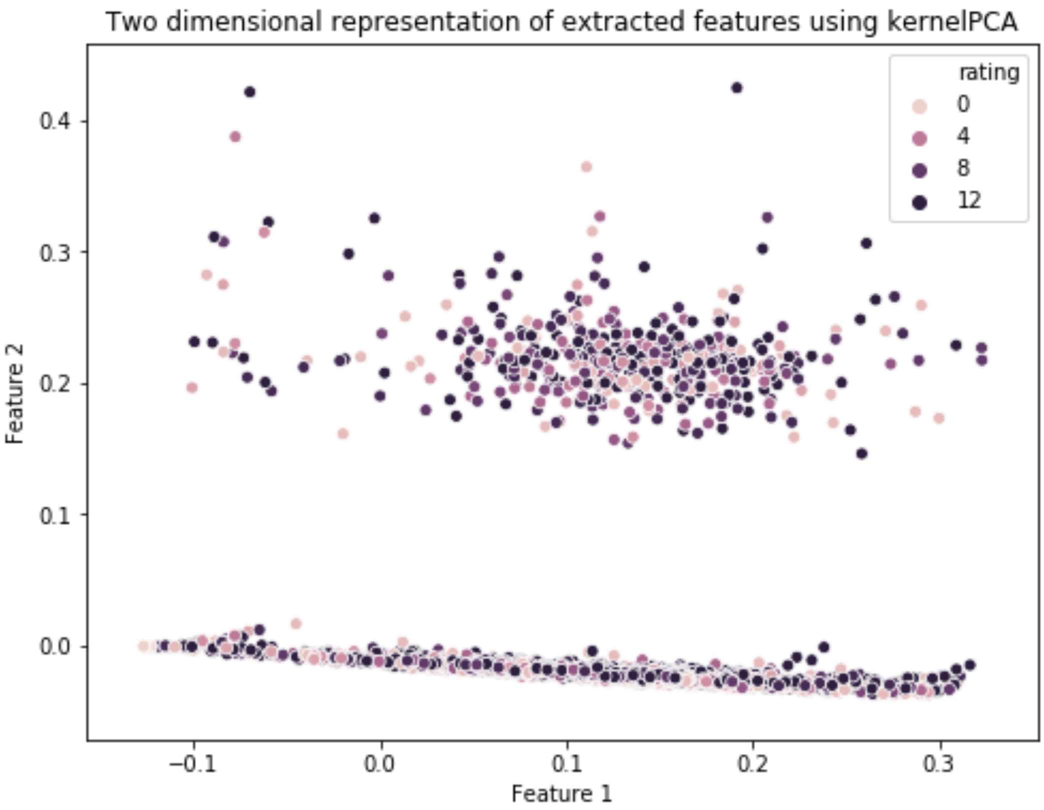


In [39]:

```
# Perform dimensionality reduction using kernel PCA
emb_vectorsPCA = KernelPCA(n_components=2, kernel='rbf').fit_transform(feat_t
```

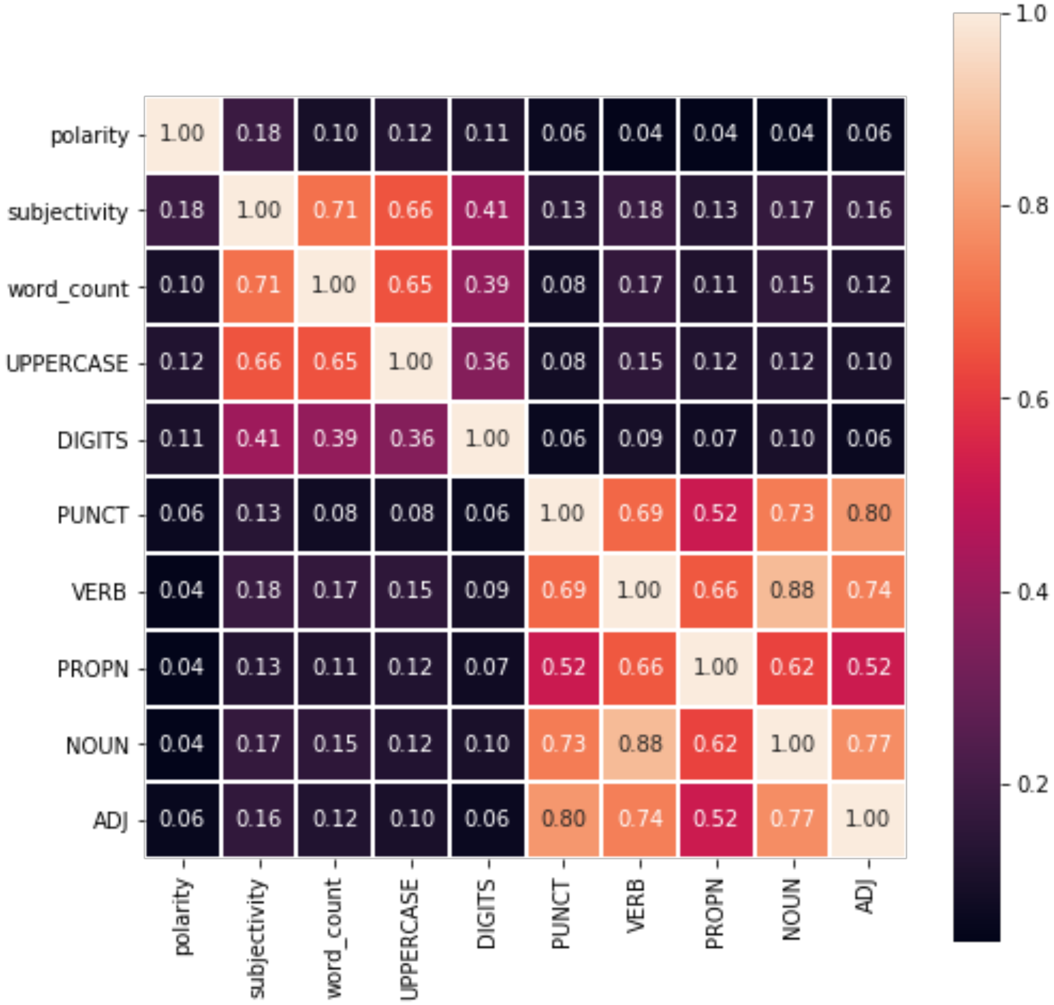
In [40]:

```
# Visualize data in lower-dimensional space
plt.figure(figsize=(8,6))
sns.scatterplot(x=emb_vectorsPCA[:, 0], y=emb_vectorsPCA[:, 1], hue=label_to_
plt.title('Two dimensional representation of extracted features using kernelP
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()
```



In [41]:

```
# Calculate correlations between the features
corr_matrix = dataset_feat.iloc[:, 2:].corr()

# Plot correlation matrix
plt.figure(figsize=(8,8))
ax = sns.heatmap(corr_matrix, annot=True, fmt='.2f', linewidths=1, square=Tru
# Set limit of y axis. To check current settings use: ax.get_ylim()
ax.set_ylim(10,0)
plt.show()
```



In [42]:

```
# Import the dataset
dataset_feat = pd.read_csv('dataset/datasets_feat/drugreview_feat.csv')
```

In [43]:

```
def token_filter(token):
    """Filter the token for text_preprocessing function.
    Check if the token is not: punctuation, whitespace, stopword or digit.

    Parameters
    ----------
    token : spacy.Token
        Token passed from text_preprocessing function.

    Returns
    -------
    Bool
       True if token meets the criteria, otherwise False.

    """
    return not (token.is_punct | token.is_space | token.is_stop | token.is_di

def text_preprocessing(df, batch_size, n_threads):
    """Perform text preprocessing using the following methods: removing HTML
    lemmatization and removing stopwords, whitespaces, punctuations, digits.

    Parameters
    ----------
    df : pandas.Series
        Pandas.Series containing strings to process.
    batch_size: int
        Size of text batch (recommended to be the power of 2).
    n_threads: int
        Number of threads in multiprocessing.

    Returns
    -------
    pandas.Series
       Pandas.Series containing processed strings.

    """
    # Remove HTML tags
    df = df.apply(remove_tags)
    # Make lowercase
    df = df.str.lower()
    processed_docs = []
    for doc in list(nlp.pipe(df, batch_size=batch_size)):
        # Remove stopwords, spaces, punctutations and digits
        text = [token for token in doc if token_filter(token)]
        # Lemmatization
        text = [token.lemma_ for token in text if token.lemma_ != '-PRON-']
        processed_docs.append(' '.join(text))
    return pd.Series(processed_docs, name='clean_review', index=df.index)
```

In [44]:

```
# Define the variables
import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter('ignore')
nlp = spacy.load('en_core_web_sm', disable = ['ner', 'parser', 'textcat', 'ta
batch_size = 512
n_threads = 2

# Test the processing time on a part of the trainig set, given batch_size and
print('Start processing 1000 examples using batch_size: {} and n_threads: {}'
start_time = timeit.default_timer()
text_preprocessing(dataset_feat.loc[:1000, 'review'], batch_size=batch_size, 
print('Processing time: {:.2f} sec'.format(timeit.default_timer() - start_tim
```

```
Start processing 1000 examples using batch_size: 512 and n_threads: 2
Processing time: 4.29 sec
```

In [45]:

```
def split_norm_save(df, name, path, part_size, batch_size, n_threads, nlp):
    """Split data frame into chunks of size equal: part_size and perform text
    Preprocess strings using the following methods: removing HTML tags, lower
    removing stopwords, whitespaces, punctuations, digits.

    Parameters
    ----------
    df : pandas.DataFrame
        Pandas.DataFrame containing 'review' column to preprocess.
    name : str
        Name of the CSV file to which export the data.
    path : str
        Absolute or relative path to directory where to save the data.
    part_size: int
        Size of the chunk to process (number of strings it contains).
    batch_size: int
        Size of text batch (recommended to be the power of 2).
    n_threads: int
        Number of threads in multiprocessing.
    nlp: spacy.lang.<language>
        Spacy language model (for example spacy.lang.en.English)

    Returns
    -------
    pandas.DataFrame
       Concatenation of the original data frame and pandas series of normaliz

    """
    from tqdm.notebook import tqdm
    if name not in os.listdir(path):
        dataset_parts = []
        N = int(len(df)/part_size)
        # Create list of dataframe chunks
        data_frames = [df.iloc[i*part_size:(i+1)*part_size, 0].copy() for i i
        print(len(data_frames))
        # Process dataset partialy
        i = 0
        for frame in tqdm(data_frames):
            # Normalize dataset chunk
            print(i)
            i += 1
            dataset_part = text_preprocessing(frame, batch_size=batch_size, n
            dataset_parts.append(dataset_part)
            # Reload nlp
            nlp = spacy.load('en_core_web_sm', disable = ['ner', 'parser', 't

        # Concatenate all parts into one series
        concat_clean = pd.concat(dataset_parts, axis=0, sort=False)
        # Concatenate dataset and cleaned review seires
        dataset_clean = pd.concat([df, concat_clean], axis=1)
        # Export data frame to CSV file
        dataset_clean.to_csv(path + name, index=False)
    else:
        print('File {} already exists in given directory.'.format(name))
```

In [46]:

```
# Define variables
nlp = spacy.load('en_core_web_sm', disable = ['ner', 'parser', 'textcat', 'ta
batch_size = 512
n_threads = 2
part_size = 5000
path = os.path.join(os.getcwd(), 'dataset/datasets_feat_clean/')
name = 'drugreview_feat_clean.csv'

# Perform text preprocessing and save the resulted frame to CSV file
split_norm_save(dataset_feat, name, path, part_size, batch_size, n_threads, n
```

```
File drugreview_feat_clean.csv already exists in given directory.
```

In [47]:

```
# Import preprocessed dataset from CSV file
dataset_feat_clean = pd.read_csv('dataset/datasets_feat_clean/drugreview_feat
```

In [48]:

```
# Display the random review before normalization
idx = np.random.randint(dataset_feat_clean.shape[0])
HTML(dataset_feat_clean.loc[idx, 'review'])
```

Out[48]: "I took Contrave as doctor ordered . I did not feel any improvement in appetite or weight loss. When I got to third week 2 am and 1 pm I was feeling bloated stomach hurt . I thought I will try to get to week 4 which was 2 am. and 2 pm. but I didn't get there. I was starting to feel out of breath and really bloated. I went back to doctor I said I can not take this anymore it is making me feel bad. I feel alright now . But my share of cost of Contrave was $229.00 which I am out of pocket and still have half the pills of 120 left."

In [49]:

```
# Display normalized review
HTML(dataset_feat_clean.loc[idx, 'clean_review'])
```

Out[49]: took contrave doctor ordered feel improvement appetite weight loss got week pm feeling bloated stomach hurt thought try week pm didn't starting feel breath bloated went doctor said anymore making feel bad feel alright share cost contrave $ pocket half pills left

In [50]:

```
# Load the language model
nlp = spacy.load('en_core_web_sm')

# Create the data frame that contains positive and neagtive reviews to visual
pos_to_visual = dataset_feat_clean.loc[dataset_feat_clean.rating >= 5, ['clea
neg_to_visual = dataset_feat_clean.loc[dataset_feat_clean.rating < 5, ['clean_
data_to_visual = pd.concat([pos_to_visual, neg_to_visual], axis=0)
# Replace numerical labels by strings (required by scattertext)
data_to_visual.rating = data_to_visual.rating.replace([10,9,8,7,6,5,4,3,2,1,0
```

In [51]:

```
# Create the Scattertext corpus
corpus = st.CorpusFromPandas(data_to_visual,
                             category_col='rating',
                             text_col='clean_review',
                             nlp=nlp).build()
```

In [52]:

```
# Create the scattertext plot
html = st.produce_scattertext_explorer(corpus,
          category='pos',
          category_name='Positive reviews',
          not_category_name='Negative reviews',
          width_in_pixels=600,
          height_in_pixels=500)
# Save the visualization as HTML file
open('assets/scattertext_visualization.html', 'wb').write(html.encode('utf-8'
```

Out[52]:

```
1631216
```

In [53]:

```
data_to_visual.loc[data_to_visual.rating =='neg', 'clean_review']
```

Out[53]:
```
6       mirena months stopped bleeding strange odor in...
9       horrible drug paxil garbage i&#039;ve majority...
12      feel like ground moving headaches diarrhea han...
14      cure bacterial vaginosis cause concurrent yeas...
15      wanted work buspar mg twice daily added wellbu...
                              ...
5266                                                 NaN
5267                                                 NaN
5268                                                 NaN
5269                                                 NaN
5270                                                 NaN
Name: clean_review, Length: 1500, dtype: object
```

In [54]:
```
data_to_visual.shape
```

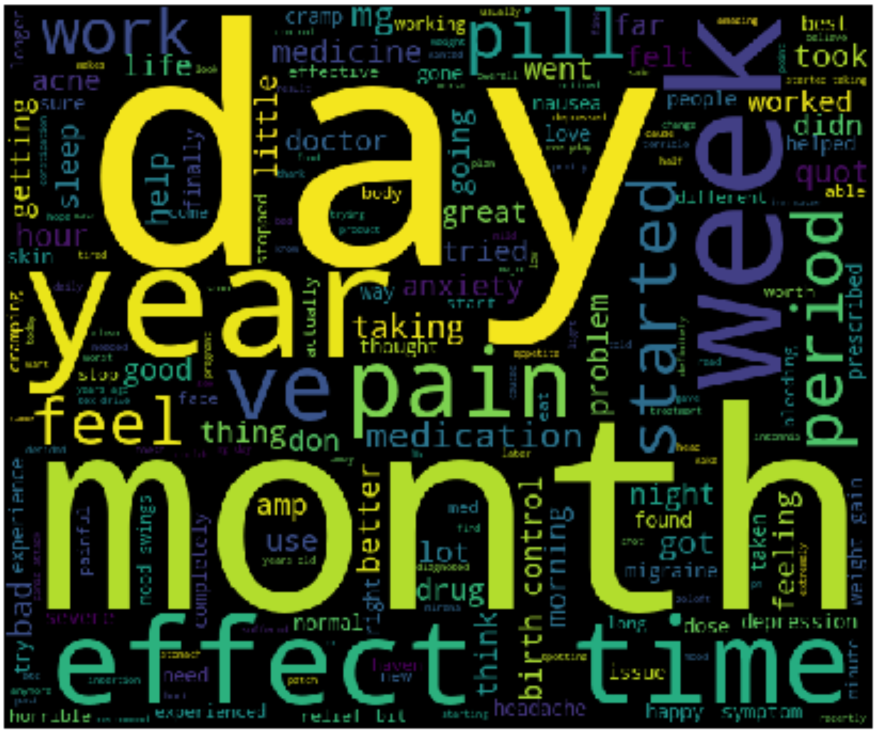
Out[54]:
```
(3000, 2)
```

In [55]:
```
data_to_visual = data_to_visual.dropna()
```

In [56]:
```
# Separate positive and negative reviews and then concatenate all reviews wit
pos_reviews = ' '.join(data_to_visual.loc[data_to_visual.rating =='pos', 'cle
neg_reviews = ' '.join(data_to_visual.loc[data_to_visual.rating =='neg', 'cle
```

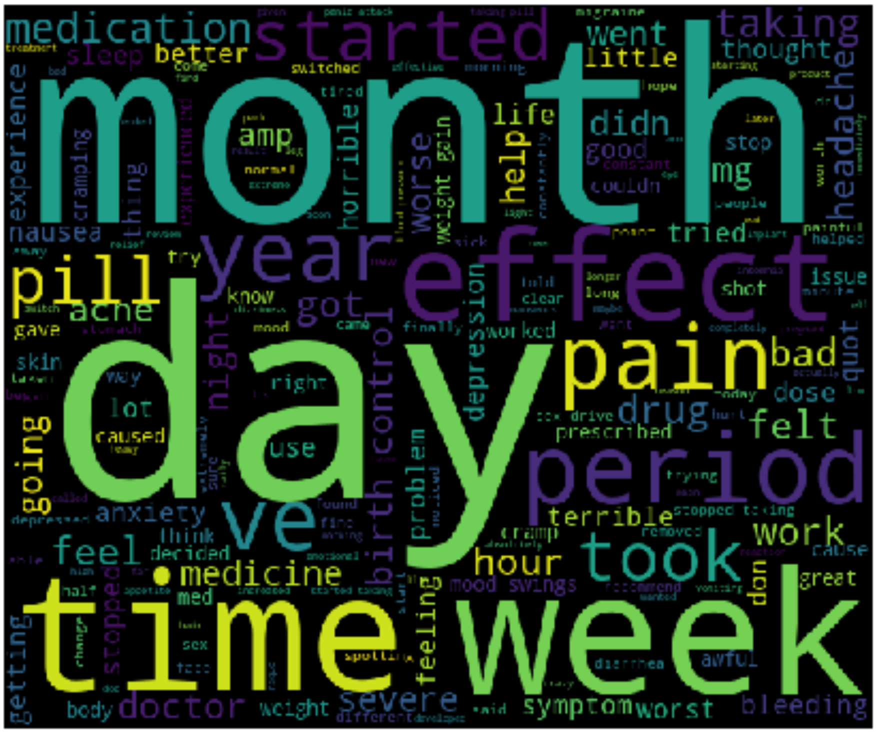
In [57]:
```
# Create wordcloud for positive reviews
wordcloud_pos = WordCloud(background_color='black',
                      width=600,
                      height=500).generate(pos_reviews)
```

In [58]:
```
# Depict wordcloud for positive reviews
plt.figure(figsize=(8,5))
plt.imshow(wordcloud_pos)
plt.axis('off')
plt.tight_layout()
```



In [59]:
```
# Create wordcloud for negative reviews
wordcloud_neg = WordCloud(background_color='black',
                      width=600,
                      height=500).generate(neg_reviews)
```

In [60]:
```
# Depict wordcloud for negative reviews
plt.figure(figsize=(8,5))
plt.imshow(wordcloud_neg)
plt.axis('off')
plt.tight_layout()
```



```
In [61]: # Split entire raw dataset into training, validation and test sets
         train_set, val_set, test_set = train_val_test_split(dataset_feat_clean, val_s
```

```
In [62]: # Check training, validation and test sets shapes
         print('Training set shape: {}'.format(train_set.shape))
         print('Validation set shape: {}'.format(val_set.shape))
         print('Test set shape: {}'.format(test_set.shape))
```

```
Training set shape: (25369, 13)
Validation set shape: (7248, 13)
Test set shape: (3625, 13)
```

```
In [63]: # Save training, validation and test sets to CSV files
         train_set.to_csv('dataset/drugreview_feat_clean/train_feat_clean.csv', index=
         val_set.to_csv('dataset/drugreview_feat_clean/val_feat_clean.csv', index=Fals
         test_set.to_csv('dataset/drugreview_feat_clean/test_feat_clean.csv', index=Fa
```

```
In [ ]:
```