

CSML1010 Project Working Copy

Sentiment Analysis with the Sentiment140 dataset

Pete Gray - YorkU #217653247

Introduction

In this project we explore and learn about natural language processing and the lifecycle of machine learning projects. Sentiment analysis will be performed using the Sentiment140 dataset[1]. We will explicitly execute data cleaning, data exploration, feature engineering, feature selection, modeling, model selection, and [OTHER STUFF?] The code is mostly assembled from bits and pieces of the coding exercises that are part of the course CSML1010, Fall 2019, at York University, Toronto.

[1] <http://help.sentiment140.com/for-students> (<http://help.sentiment140.com/for-students>)

TABLE OF CONTENTS

1. Load data and libraries
2. Data Cleaning
3. Data Exploration
4. Feature Engineering
5. Feature Selection

Import libraries

```
In [1]: import pandas as pd
import numpy as np
np.set_printoptions(precision=2, linewidth=80)
import warnings
warnings.filterwarnings("ignore")
import model_evaluation_utils as meu
```

Adjust pandas display

```
In [2]: pd.options.display.max_columns = 30
pd.options.display.max_rows = 100
pd.options.display.float_format = '{:.2f}'.format
pd.options.display.precision = 2
pd.options.display.max_colwidth = -1
```

Import matplotlib and seaborn and adjust defaults

```
In [3]: %matplotlib inline
%config InlineBackend.figure_format = 'svg'

from matplotlib import pyplot as plt
plt.rcParams['figure.dpi'] = 100

import seaborn as sns
sns.set_style("whitegrid")
```

Read data from local filesystem and csv source

```
In [4]: df = pd.read_csv("training.1600000.processed.noemoticon.csv", encoding="ISO-8859
```

Check data with quick visual inspection

In [5]: df

Out[5]:

			Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, that's a bummer. You shoulda got David Carr of Third Day to do it. ;D
0	0	1467810369				
0	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by texting it... and might cry as a result School today also. Blah!
1	0	1467810917	Mon Apr 06 22:19:53 PDT 2009	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Managed to save 50% The rest go out of bounds
2	0	1467811184	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	ElleCTF	my whole body feels itchy and like its on fire
3	0	1467811193	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	Karoli	@nationwideclass no, it's not behaving at all. i'm mad. why am i here? because I can't see you all over there.
4	0	1467811372	Mon Apr 06 22:20:00 PDT 2009	NO_QUERY	joy_wolf	@Kwesidei not the whole crew
...
1599994	4	2193601966	Tue Jun 16 08:40:49 PDT 2009	NO_QUERY	AmandaMarie1028	Just woke up. Having no school is the best feeling ever
1599995	4	2193601969	Tue Jun 16 08:40:49 PDT 2009	NO_QUERY	TheWDBboards	TheWDB.com - Very cool to hear old Walt interviews! â™« http://blip.fm/~8bmta
1599996	4	2193601991	Tue Jun 16 08:40:49 PDT 2009	NO_QUERY	bpbabe	Are you ready for your MoJo Makeover? Ask me for details
1599997	4	2193602064	Tue Jun 16 08:40:49 PDT 2009	NO_QUERY	tinydiamondz	Happy 38th Birthday to my boo of alll time!!! Tupac Amaru Shakur
1599998	4	2193602129	Tue Jun 16 08:40:50 PDT 2009	NO_QUERY	RyanTrevMorris	happy #charitytuesday @theNSPCC @SparksCharity @SpeakingUpH4H

1599999 rows × 6 columns

Give dataframe columns

```
In [6]: df.columns = ['sentiment', 'ID', 'Time', 'none', 'username', 'Text']
```

```
In [7]: df.count()
```

```
Out[7]: sentiment    1599999  
ID                1599999  
Time              1599999  
none              1599999  
username          1599999  
Text              1599999  
dtype: int64
```

Now it has columns, this seems better.

Check for nulls in the Text column

```
In [8]: df = df[df["Text"].notnull()]  
df.count()
```

```
Out[8]: sentiment    1599999  
ID                1599999  
Time              1599999  
none              1599999  
username          1599999  
Text              1599999  
dtype: int64
```

Set temporary dataset size, for quicker processing

```
In [9]: dev_data_size = 800
```

```
In [10]: start_row = int(800000-(dev_data_size/2))-1  
finish_row = int(800000+(dev_data_size/2))-1  
df_sm = df[start_row:finish_row]  
df_sm.count()
```

```
Out[10]: sentiment    800  
ID                800  
Time              800  
none              800  
username          800  
Text              800  
dtype: int64
```

```
In [11]: columns = [col for col in df.columns if not col.startswith('self')]
columns
```

```
Out[11]: ['sentiment', 'ID', 'Time', 'none', 'username', 'Text']
```

```
In [12]: raw_text = np.array(df_sm['Text'])
sentiments = np.array(df_sm['sentiment'])
raw_text[5:15]
```

```
Out[12]: array(['If ever there was a time when I wanted to rip my nose off of my face and throw it FAR, that time is NOW!! Running out of tissues ',
                "Went to the doctor, got some meds for the pain.. Hoping they work so I don't have to take x-rays.. Dreading going to work today.. ",
                'Everywhere I look...happy pregnant people. I miss my midwife ',
                '62 was the age my grandmother died from lung cancer. I never got to meet her. ',
                'WOW Farrah Fawcett died i think she had anal cancer. o.0',
                'Charlie has a new angel! Rest in peace farrah fawcett. ',
                "@Blue_Bunny I went & signed up, but I didn't see a coupon for Bomb pops. #houseofgems",
                '@michellebeckham OMG! I hope Glenn is okay. Sorry to hear the news. ',
                'R.I.P. Farrah Fawcett. I wanted to be her when I was little. ',
                "@ang_w Oh no! That's too bad. "], dtype=object)
```

=====

Data Cleaning

=====

Type *Markdown* and LaTeX: α^2

Cleaning function

```

In [13]: import re
def clean(s):
    s = s.replace(r'<lb>', "\n")
    s = s.replace(r'<tab>', "\t")

    # As a sanity check - s = s.replace(r'W', "Q")

    s = re.sub(r'<br */*>', "\n", s)
    s = s.replace("&lt;", "<").replace("&gt;", ">").replace("&";", "&")
    s = s.replace("&";", "&")
    # markdown urls
    s = re.sub(r'\(https*://[^\)]*\)', "", s)
    # normal urls
    s = re.sub(r'https*://[^\s]*', "", s)
    s = re.sub(r'_+', ' ', s)
    s = re.sub(r'"'+', '""', s)

    # NUMBERS IN THE TEXT DATA
    # The numbers in the data came to light during feature engineering.
    # I will try different things here.

    # A processor-efficient approach, as suggested at:
    # https://stackoverflow.com/questions/30315035/strip-numbers-from-string-in-p
    # s = s.translate(None, '0123456789')
    # Well, that totally didn't work.

    # From the same link, a more conventional, but less efficient approach:

    s = re.sub(r'\d+', '', s)

    # USERNAMES IN THE DATA
    # Let's see if life gets any cleaner with these removed, or if it just blows
    # Using code found at:
    # https://stackoverflow.com/questions/50830214/remove-usernames-from-twitter

    s = re.sub('@[^\s]+', '', s)

    # Was 4374 for 2000
    # 3593 and 1985

    return str(s)

```

Observations on the removal of Usernames from data

Running with 2000 rows, Bag of Words came up with 4374 dimensions. This took a dog's age to run through RFE. Applying the removal of usernames (strings beginning with '@') from the text data caused 15 of the 2000 to become null - they were stripped from the dataset. Most significantly, it resulted in a reduction of dimensions at the Bag of Words stage to 3593. While this doesn't appear to enable us to ramp up significantly, it does help.

It is worth noting that there appears to be a bit less gibberish in the selected features after applying this. So, a little quicker, a little cleaner, it's a keeper.

Create new column in dataframe

```
In [14]: df_sm["text_clean"] = ''
```

Iterate and clean

```
In [15]: for i, row in df_sm.iterrows():
          if i % 1000 == 0:
              print('processed:'.format(i), i)
          df_sm.at[i, "text_clean"] = clean(row.Text)
```

processed: 800000

Check results

In [16]: df_sm.head()

Out[16]:

	sentiment	ID	Time	none	username	
799599	0	2329056794	Thu Jun 25 10:17:56 PDT 2009	NO_QUERY	enge10	GUYS the ghost is back at room
799600	0	2329056832	Thu Jun 25 10:17:56 PDT 2009	NO_QUERY	alysamarsiella	RIP Farrah what a sham
799601	0	2329056954	Thu Jun 25 10:17:57 PDT 2009	NO_QUERY	velobabe	http://www.cnn.com/2009/SHC
799602	0	2329057090	Thu Jun 25 10:17:58 PDT 2009	NO_QUERY	XoAngelJenn36oX	it was so nice and sunny this
799603	0	2329057145	Thu Jun 25 10:17:58 PDT 2009	NO_QUERY	OliviaDAngelo	

Additional pre-processing: tokenization, removing extra whitespaces, lower casing and more advanced operations like spelling corrections, grammatical error corrections,

removing repeated characters.

```
In [17]: import nltk
wpt = nltk.WordPunctTokenizer()
nltk.download("stopwords")
stop_words = nltk.corpus.stopwords.words('english')

[nltk_data] Downloading package stopwords to
[nltk_data] C:\Users\Dell\AppData\Roaming\nltk_data...
[nltk_data] Package stopwords is already up-to-date!
```

Define normalization function

```
In [18]: def normalize_document(doc):
# lower case and remove special characters\whitespaces
doc = re.sub(r'^a-zA-Z0-9\s', '', doc, re.I)
doc = doc.lower()
doc = doc.strip()
# tokenize document
tokens = wpt.tokenize(doc)
# filter stopwords out of document
filtered_tokens = [token for token in tokens if token not in stop_words]
# re-create document from filtered tokens
doc = ' '.join(filtered_tokens)
return doc
```

```
In [19]: normalize_corpus = np.vectorize(normalize_document)
```

```
In [20]: df_sm["text_normalized"] = ''
```

```
In [21]: for i, row in df_sm.iterrows():
if i % 1000 == 0:
print('processed:'.format(i), i)
df_sm.at[i, "text_normalized"] = normalize_corpus(row.text_clean)
```

processed: 800000

check results

In [22]: df_sm

Out[22]:

	sentiment	ID	Time	none	username	
799599	0	2329056794	Thu Jun 25 10:17:56 PDT 2009	NO_QUERY	enge10	GUYS the ghost is back at room
799600	0	2329056832	Thu Jun 25 10:17:56 PDT 2009	NO_QUERY	alysamarsiella	RIP Farrah what a sham
799601	0	2329056954	Thu Jun 25 10:17:57 PDT 2009	NO_QUERY	velobabe	http://www.cnn.com/2009/SHC
799602	0	2329057090	Thu Jun 25 10:17:58 PDT 2009	NO_QUERY	XoAngelJenn36oX	it was so nice and sunny this
799603	0	2329057145	Thu Jun 25 10:17:58 PDT 2009	NO_QUERY	OliviaDAngelo	
...	
800394	4	1467898311	Mon Apr 06 22:42:53 PDT 2009	NO_QUERY	alliele	will be part of my friend's w kase

	sentiment	ID	Time	none	username	
800395	4	1467898335	Mon Apr 06 22:42:53 PDT 2009	NO_QUERY	cate3221	@Bern_morley it's the weath
800396	4	1467898386	Mon Apr 06 22:42:57 PDT 2009	NO_QUERY	Dezuray	@
800397	4	1467898396	Mon Apr 06 22:42:54 PDT 2009	NO_QUERY	kielymedia	Whoops, sorry to anyone try
800398	4	1467898410	Mon Apr 06 22:42:57 PDT 2009	NO_QUERY	Missmoore18	@chocolate_dip k @yoeyfreshier wasnt gettin

800 rows × 8 columns



```
In [23]: import spacy
nlp = spacy.load('en_core_web_sm')
```

```

In [24]: for i, row in df_sm.iterrows():
        if i % 1000 == 0:
            print(i)
        if (row["text_normalized"] and len(str(row["text_normalized"])) < 1000000):
            doc = nlp(str(row["text_normalized"]))
            adjectives = []
            nouns = []
            verbs = []
            lemmas = []

            for token in doc:
                lemmas.append(token.lemma_)
                if token.pos_ == "ADJ":
                    adjectives.append(token.lemma_)
                if token.pos_ == "NOUN" or token.pos_ == "PROPN":
                    nouns.append(token.lemma_)
                if token.pos_ == "VERB":
                    verbs.append(token.lemma_)

            df_sm.at[i, "text_lemma"] = " ".join(lemmas)
            df_sm.at[i, "text_nouns"] = " ".join(nouns)
            df_sm.at[i, "text_adjectives"] = " ".join(adjectives)
            df_sm.at[i, "text_verbs"] = " ".join(verbs)
            df_sm.at[i, "text_nav"] = " ".join(nouns+adjectives+verbs)
            df_sm.at[i, "no_tokens"] = len(lemmas)

```

800000

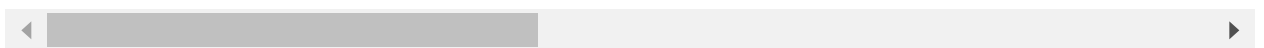
In [25]: df_sm

Out[25]:

	sentiment	ID	Time	none	username	
799599	0	2329056794	Thu Jun 25 10:17:56 PDT 2009	NO_QUERY	enge10	GUYS the ghost is back at room
799600	0	2329056832	Thu Jun 25 10:17:56 PDT 2009	NO_QUERY	alysamarsiella	RIP Farrah what a sham
799601	0	2329056954	Thu Jun 25 10:17:57 PDT 2009	NO_QUERY	velobabe	http://www.cnn.com/2009/SHC
799602	0	2329057090	Thu Jun 25 10:17:58 PDT 2009	NO_QUERY	XoAngelJenn36oX	it was so nice and sunny this
799603	0	2329057145	Thu Jun 25 10:17:58 PDT 2009	NO_QUERY	OliviaDAngelo	
...	
800394	4	1467898311	Mon Apr 06 22:42:53 PDT 2009	NO_QUERY	alliele	will be part of my friend's w kase

	sentiment		ID	Time	none	username	
800395	4	1467898335	Mon Apr 06 22:42:53 PDT 2009	NO_QUERY	cate3221	@Bern_morley it's the weath	
800396	4	1467898386	Mon Apr 06 22:42:57 PDT 2009	NO_QUERY	Dezuray		@
800397	4	1467898396	Mon Apr 06 22:42:54 PDT 2009	NO_QUERY	kielymedia	Whoops, sorry to anyone try	
800398	4	1467898410	Mon Apr 06 22:42:57 PDT 2009	NO_QUERY	Missmoore18	@chocolate_dip k @yoeyfreshier wasnt gettin	

800 rows × 14 columns



Sometimes, our cleaning reduces our text to nothing! Which makes a lot of stuff unable to run.

This hack has been helpful in that regard:

```
In [26]: # Save in case we need it later:

df_sm = df_sm[df_sm["text_nav"].notnull()]
df_sm.count()

# shuffle the dataset for later.
# df = df.sample(frac=1)
```

```
Out[26]: sentiment      792
ID                    792
Time                 792
none                 792
username             792
Text                 792
text_clean           792
text_normalized      792
text_lemma           792
text_nouns           792
text_adjectives      792
text_verbs           792
text_nav             792
no_tokens            792
dtype: int64
```

Saving Cleaned Data to the Filesystem

So I can run without re-cleaning, or move a chunk to the Cloud for experiments.

```
In [27]: # save cleaned data

df_sm.to_csv('cleaned01.csv', encoding='utf-8', index=False)
```

Loading Cleaned Data from the Filesystem

Saved in various sizes, load as appropriate

```
In [28]: # df_sm = pd.read_csv("s140_cln_100k.csv", encoding="utf-8")

## s140_cln_100k.csv
```

=====

Data Exploration

=====

Is dataset balanced?

We know that the original dataset is split right down the middle, with 800,000 positive documents and 799,999 negative. Let's check that:

```
In [29]: df['sentiment'].value_counts()
```

```
Out[29]: 4      800000
         0      799999
         Name: sentiment, dtype: int64
```

Excellent. Now, let's check that our hackishly sampled little subset is also balanced:

```
In [30]: df_sm['sentiment'].value_counts()
```

```
Out[30]: 0      397
         4      395
         Name: sentiment, dtype: int64
```

Good. Seems balanced enough for now.

Show data types in each column

```
In [31]: df_sm.dtypes
```

```
Out[31]: sentiment      int64
         ID             int64
         Time           object
         none           object
         username       object
         Text           object
         text_clean     object
         text_normalized object
         text_lemma     object
         text_nouns     object
         text_adjectives object
         text_verbs     object
         text_nav       object
         no_tokens      float64
         dtype: object
```

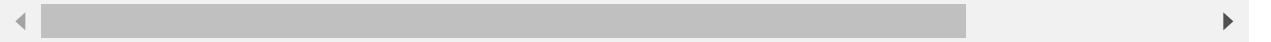
Summary of numerical features

Not the most useful thing, but helpful as a quick sanity check.

In [32]: `df_sm.describe().transpose()`

Out[32]:

	count	mean	std	min	25%	50%
sentiment	792.00	1.99	2.00	0.00	0.00	0.00
ID	792.00	1899579959.58	430909084.71	1467822272.00	1467861309.25	2329056813.00
no_tokens	792.00	8.86	5.51	1.00	4.00	8.00



Exploring text at different levels of cleaning

```
In [33]: df_sm[['text_clean', 'text_normalized', 'text_lemma', 'text_nav']].sample(10)
```

```
Out[33]:
```

	text_clean	text_normalized	text_lemma	text_nav
800384	If you are at #Paragon please vote my work! LOL	paragon please vote work lol	paragon please vote work lol	paragon work lol vote
799969	Aww shoot. Wonder if she tied the knot with Ryan O'Neal before she died. Sad	aww shoot wonder tied knot ryan oneal died . sad	aww shoot wonder tie knot ryan oneal die . sad	aww shoot wonder knot ryan oneal sad tie die
800327	if i know how to do something, and have time... Its so easy to answer	know something time .. easy answer	know something time .. easy answer	time answer easy know
799822	Awww!..Farrah Fawcett passed away. R..I.P. Kim remember when I would try to do my Farrah Fawcett flips in my hair?? Sad	awww!farrah fawcett passed away . r . . . p . kim remember would try farrah fawcett flips hair ?? sad	awww!farrah fawcett pass away . r . . . p . kim remember would try farrah fawcett flip hair ? ? sad	awww!farrah fawcett kim farrah fawcett flip hair sad pass remember would try
800027	reaching amritsar in an hour and (if i find a bus) should be at wagah border by pm -	reaching amritsar hour find bus wagah border pm -	reach amritsar hour find bus wagah border pm -	amritsar hour bus border pm wagah reach find
799660	darn it! all that work to get my OS back in English and it doesn't even have the options I want. Think i need to install some software	darn work get os back english doesnt even options want . think need install software	darn work get os back english do not even option want . think need install software	work os english option software darn get want think need install
800379	Week of the fitness challenge and I'm starting off strong. Just submitted my stats for the night and it ain't bad	week fitness challenge im starting strong submitted stats night ' bad	week fitness challenge -PRON- be start strong submitted stat night ' bad	week fitness challenge stat night strong submitted bad be start
800256	up and running, London calling	running london calling	run london call	london run call
799821	Damn... Farrah. Just saw Food Inc. last night, I wonder how many less people would get sick in the world if we could eat real foods.	damn . farrah . saw food inc . last night , wonder many less people would get sick world could eat real foods .	damn . farrah . see food inc . last night , wonder many less people would get sick world could eat real food .	farrah food inc . night people world food last many less sick real see wonder would could eat
800240	dear dbq, you were a pain in the asssss. i'm glad i'm DONE with you!	dear dbq pain asssss ' glad ' done !	dear dbq pain asssss ' glad ' do !	dbq pain asssss dear glad do

```
In [34]: # Import matplotlib and seaborn and adjust some defaults
%matplotlib inline
%config InlineBackend.figure_format = 'svg'

from matplotlib import pyplot as plt
plt.rcParams['figure.dpi'] = 100

import seaborn as sns
sns.set_style("whitegrid")
```

Creating a List of Tokens from a List of Documents

```
In [35]: def my_tokenizer(text):  
         return text.split() if text != None else []
```

```
In [36]: tokens = df_sm.text_nav.map(my_tokenizer).sum()
```

```
In [37]: print(tokens[:200])
```

```
['guy', 'house', 'havoc', 'room', 'hour', 'night', 'cleaning', 'odd', 'ghost',  
'wreak', 'live', 'farrah', 'shame', 'hair', 'tonight', 'honor', 'rip', 'may',  
'feather', 'fawcett', 'loss', 'cancer', 'farrah', 'rip', 'nice', 'morning', 'sk  
y', 'pitch', 'black', 'swimming', 'today', 'sunny', 'want', 'go', 'farrah', 'fa  
wcett', 'sad', 'time', 'nose', 'face', 'time', 'tissue', 'far', 'want', 'rip',  
'throw', 'run', 'doctor', 'med', 'pain', 'work', 'x', '-', 'ray', 'work', 'toda  
y', 'go', 'get', 'hope', 'take', 'dread', 'go', 'people', 'miss', 'midwife', 'h  
appy', 'pregnant', 'look', 'age', 'grandmother', 'lung', 'cancer', 'meet', 'di  
e', 'get', 'farrah', 'fawcett', 'cancer', 'anal', 'die', 'think', 'charlie', 'n  
ew', 'angel', 'peace', 'farrah', 'fawcett', 'rest', 'bunny', 'coupon', 'bomb',  
'pop', '#', 'houseofgem', 'go', 'sign', 'see', 'omg', 'hope', 'glenn', 'news',  
'hear', 'farrah', 'fawcett', 'rip', 'want', 'w', 'bad', 's', 'farrah', 'nooooo  
o', 'beach', 'hour', 'idk', 'internet', 'ill', 'leave', 'farrah', 'hair', 'smil  
e', 'spirit', 'live', 'angel', 'sad', 'fav', 'restaurant', 'hope', 'owner', 'ch  
ange', 'menu', 'staff', 'local', 'new', 'hear', 'sell', 'awww', 'farrah', 'fawc  
ett', 'charlie', 'angel', 'know', 'jealous', 'good', 'feel', 'shuck', 'triple',  
'quadruple', 'weekend', 'wen', 'dot', 'h', 'telling', 'ppl', 'double', 'book',  
'get', 'cancel', 'touchy', 'farrah', 'fawcett', 'die', 'farrah', 'rip', 'ugh',  
'wish', 'avenue', 'valencia', 'concert', 'tonight', 'grand', 'rapids', 'go', 'h  
ate', 'uggghhhhhh', 'wrist', 'soooo', 'today', 'hurt', 'badddd', 'go', 'garmo',  
'sad', 's', 'scull', 'week', 'wildwood', 'go', 'come', 'have', 'word', 'sound',  
'family', 'thought', 'last', 'good', 'good', 'see', 'would', 'hear', 'wish']
```

Counting Frequencies with a Counter

```
In [38]: from collections import Counter
```

```
counter = Counter(tokens)
counter.most_common(20)
```

```
Out[38]: [('farrah', 98),
 ('go', 84),
 ('fawcett', 68),
 ('good', 54),
 ('get', 53),
 ('sad', 52),
 ('be', 52),
 ('day', 44),
 ('love', 42),
 ('work', 41),
 ('time', 36),
 ('miss', 35),
 ('know', 35),
 ('quot', 34),
 ('rip', 33),
 ('thank', 33),
 ('want', 32),
 ('today', 31),
 ('die', 31),
 ('think', 30)]
```

```
In [39]: print([t[0] for t in counter.most_common(200)])
```

```
['farrah', 'go', 'fawcett', 'good', 'get', 'sad', 'be', 'day', 'love', 'work',
 'time', 'miss', 'know', 'quot', 'rip', 'thank', 'want', 'today', 'die', 'thin
 k', 'see', 's', 'night', 'make', 'feel', 'say', 'pass', 'tonight', 'could', 'to
 morrow', 'can', 'morning', 'new', 'peace', 'come', 'need', 'lol', 'would', 'tr
 y', 'find', 'rest', 'bad', 'great', 'angel', 'last', 'u', 'tell', 'sleep', 'hop
 e', 'hear', 'week', 'thing', 'bed', 'twitter', 'happy', 'wish', 'much', 'eat',
 'watch', 'friend', 'take', 'show', 'read', 'hair', 'look', 'hate', 'sorry', 'wa
 it', 'let', 'guy', 'charlie', 'hurt', 'life', 'hot', 'way', 'tweet', 'hour', 'm
 ay', 'news', 'family', 'r', 'year', 'school', 'glad', 'cancer', 'people', 'gam
 e', 'use', 'talk', 'many', 'man', 'ur', 'god', 'lose', 'call', 'mean', 'do', 'h
 aha', 'help', 'home', 'old', 'seem', 'house', 'ill', 'leave', 'have', 'right',
 'song', 'crazy', 'food', 'well', 'big', 'first', 'pm', 'twitt', 'suck', 'post',
 'w', 'ugh', 'lunch', 'pay', 'woman', 'little', 'mine', 'rain', 'fun', 'ready',
 'kid', 'agree', 'website', 'start', 'sure', 'real', 'ass', 'follow', 'test', 'g
 oodnight', 'yay', 'jon', 'update', 'amazing', 'movie', 'live', 'loss', 'face',
 'meet', '#', 'omg', 'sound', 'keep', 'person', 'wrong', 'shit', 'summer', 'answ
 er', 'reply', 'pool', 'catch', 'star', 'icon', 'enough', 'ask', 'p', 'moment',
 'fuck', 'enjoy', 'battle', 'forget', 'place', 'prayer', 'head', 'put', 'beautif
 ul', 'lay', 'sick', 'n', 'bored', 'girl', 'lot', 'buy', 'play', 'tired', 'aweso
 me', 'sweet', 'easy', 'luck', 'room', 'nice', 'pain', 'internet', 'awww', 'weat
 her', 'lovely', 'spring', 'damn', 'dance', 'poor', 'cut', 'death', 'storm']
```

Remove stopwords from list of tokens

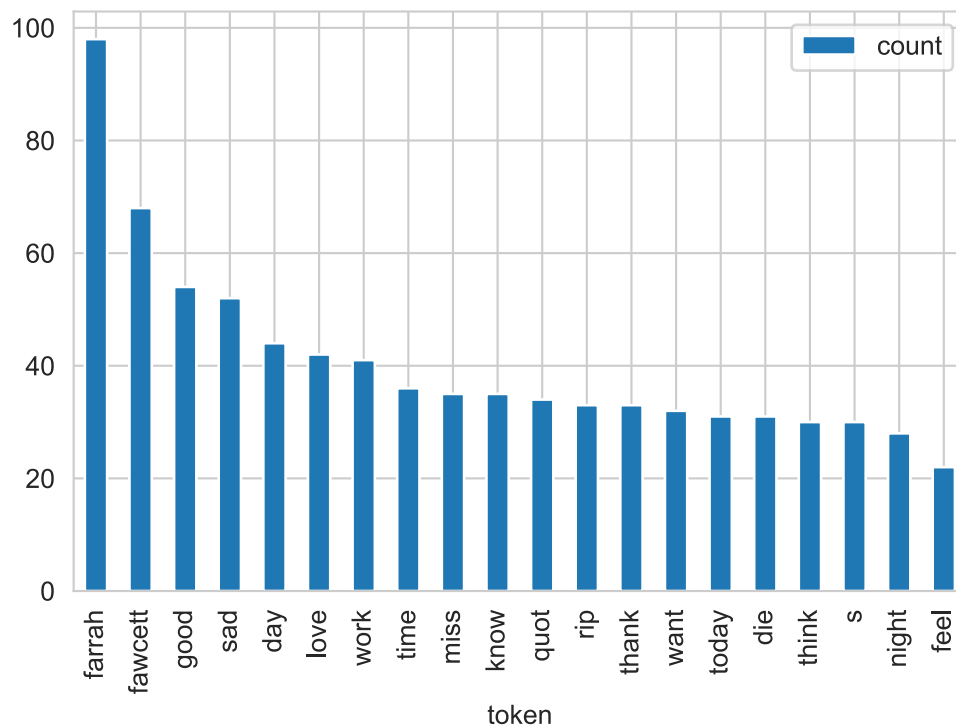
```
In [40]: from spacy.lang.en.stop_words import STOP_WORDS

def remove_stopwords(tokens):
    """Remove stopwords from a list of tokens."""
    return [t for t in tokens if t not in STOP_WORDS]

# rebuild counter
counter = Counter(remove_stopwords(tokens))
```

```
In [41]: # convert list of tuples into data frame
freq_df = pd.DataFrame.from_records(counter.most_common(20),
                                     columns=['token', 'count'])

# create bar plot
freq_df.plot(kind='bar', x='token');
```



Word clouds

```
In [42]: %matplotlib inline
import matplotlib.pyplot as plt
```

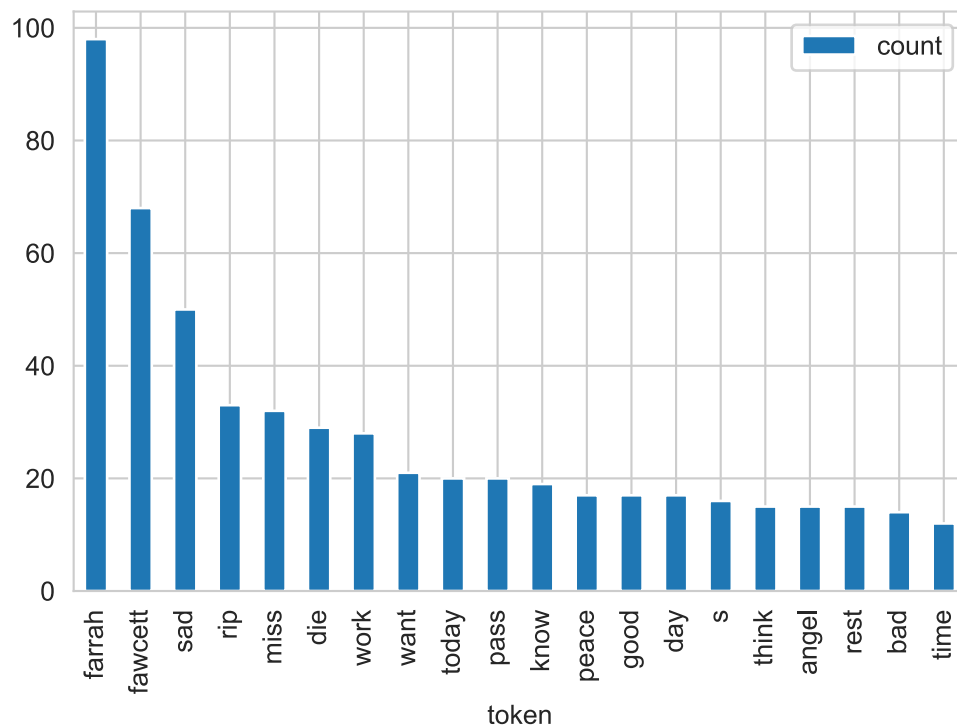


```

In [45]: neg_df = df_sm[df_sm['sentiment']==0]
neg_tokens = neg_df.text_nav.map(my_tokenizer).sum()
neg_counter = Counter(neg_tokens)
#neg_counter.most_common(20)
neg_counter = Counter(remove_stopwords(neg_tokens))
neg_freq_df = pd.DataFrame.from_records(neg_counter.most_common(20),
                                       columns=['token', 'count'])

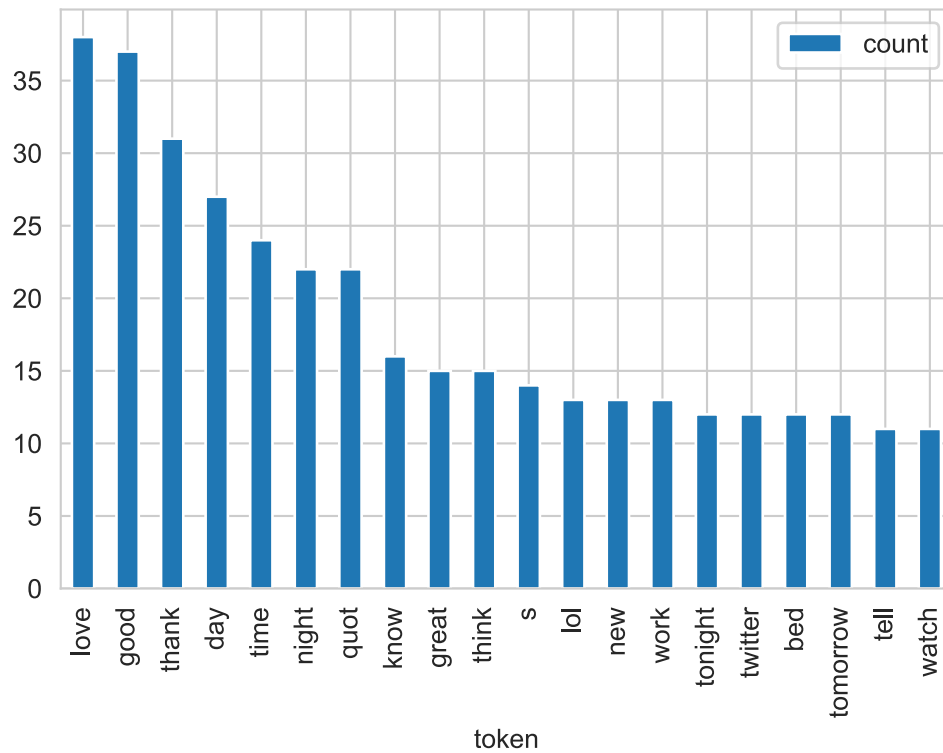
# create bar plot
neg_freq_df.plot(kind='bar', x='token');

```




```
In [47]: pos_df = df_sm[df_sm['sentiment']==4]
pos_tokens = pos_df.text_nav.map(my_tokenizer).sum()
pos_counter = Counter(pos_tokens)
#pos_counter.most_common(20)
pos_counter = Counter(remove_stopwords(pos_tokens))
pos_freq_df = pd.DataFrame.from_records(pos_counter.most_common(20),
                                       columns=['token', 'count'])

# create bar plot
pos_freq_df.plot(kind='bar', x='token');
```



```
wordcloud(pos_counter)
```



Well, WOW, those are some mighty positive words, in the positive documents word cloud!

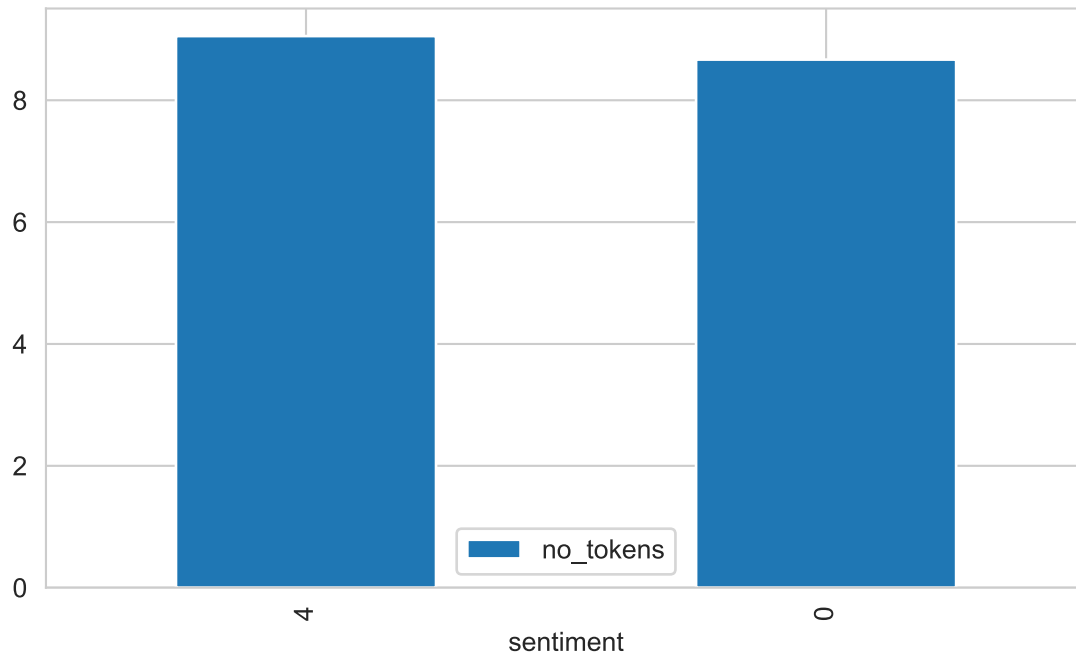
The most frequently occurring words are clearly different in the text labeled as positive and the text labeled as negative.

In the early phases of development, we were using a dataset, centered around the change at 800,000 rows, with 4000 samples. In this sample, the words "Farrah" and "Fawcett" were the most common words. This suggested perhaps that we need to be more careful about subsampling our data (perhaps a random selection approach would yield less weird results?) it made for a surprise when we looked at positive and negative sentiment. Who would have thought that "Farrah" and "Fawcett" would be exclusively from statements with negative sentiment? Surprised me, for sure, but there it is. Data Exploration!

Exploring text complexity

```
df_sm['no_tokens'] = df_sm.text_lemma\
    .map(lambda l: 0 if l==None else len(l.split()))
```

```
In [50]: # mean number of tokens by sentiment
df_sm.groupby(['sentiment']) \
    .agg({'no_tokens':'mean'}) \
    .sort_values(by='no_tokens', ascending=False) \
    .plot(kind='bar', figsize=(7,4));
```



```
In [51]: # render plots as retina or png, because svg is very slow
%config InlineBackend.figure_format = 'retina'

import seaborn as sns

def multi_boxplot(data, x, y, ylim = None):
    '''Wrapper for sns boxplot with cut-off functionality'''
    # plt.figure(figsize=(30, 5))
    fig, ax = plt.subplots()
    plt.xticks(rotation=90)

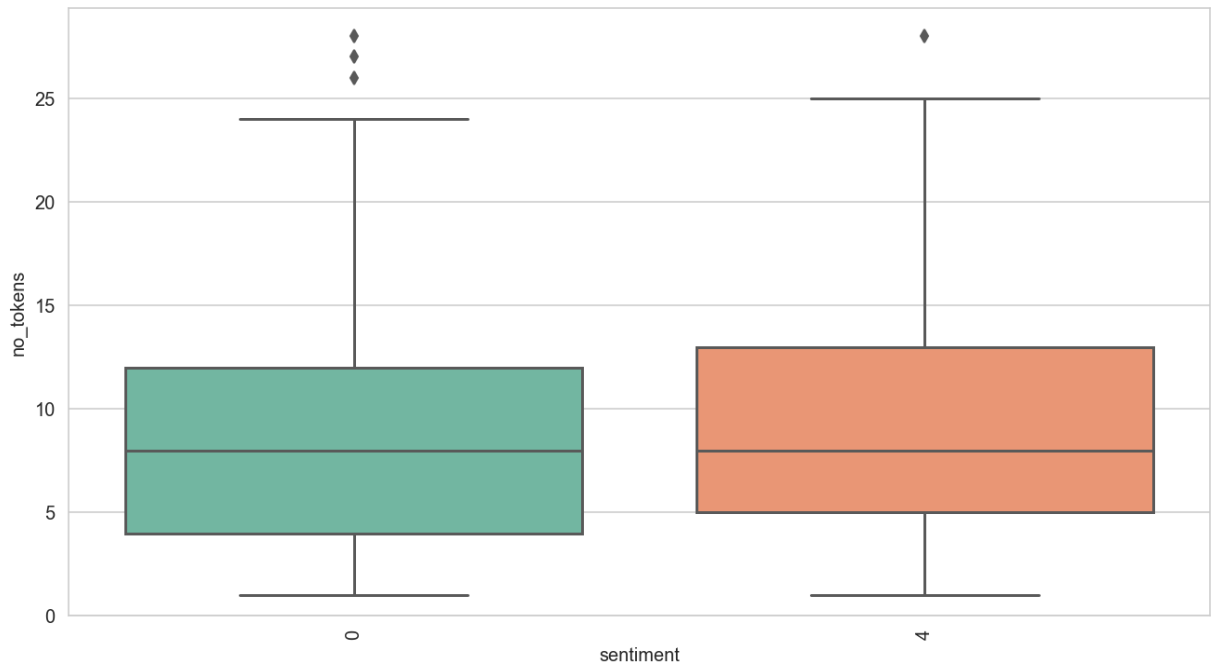
    # order boxplots by median
    ordered_values = data.groupby(x)[[y]] \
        .median() \
        .sort_values(y, ascending=False) \
        .index

    sns.boxplot(x=x, y=y, data=data, palette='Set2',
        order=ordered_values)

    fig.set_size_inches(11, 6)

    # cut-off y-axis at value ylim
    ax.set_ylim(0, ylim)
```

```
In [52]: multi_boxplot(df_sm, 'sentiment', 'no_tokens');
```



```
In [53]: # print text of outliers
df_sm['text_lemma'][df_sm.no_tokens > 1500]
```

```
Out[53]: Series([], Name: text_lemma, dtype: object)
```

```
In [54]: # cut-off diagram at y=40

# CAREFUL!!! this isn't that meaningful, and it takes for freaking ever to plot,
# multi_boxplot(df_sm, 'username', 'no_tokens', ylim=40)
```

=====

Feature Engineering

=====

Feature Engineering and Feature Selection are part of the Data Preparation stage of the CRISP-DM methodology. After data has been cleaned and explored, it must be transformed from raw.

... methodology, it has been cleaned and explored, it has been transformed from raw, unstructured text into a structured numeric format that can be used as inputs for our models. Simpler Data Engineering techniques focus on vectorizing individual words, with little emphasis on the contexts of the words. We use Bag of Words and Bag of N-Grams to explore these simpler approaches. While easy to use, and not terribly demanding in terms of computer power required, these techniques are fundamentally less powerful than more modern, processor intensive techniques that concern themselves more with the context of the words. We use pre-trained word embeddings for our advanced feature engineering efforts, in order to avoid computational bottlenecks. If time allows, we may attempt to train our own embedding at some point.

A vector space model is simply a mathematical model to represent unstructured text (or any other data) as numeric vectors, such that each dimension of the vector is a specific feature\attribute.

Bag of Words Model

The bag of words model represents each text document as a numeric vector where each dimension is a specific word from the corpus and the value could be its frequency in the document, occurrence (denoted by 1 or 0) or even weighted values. The model's name is such because each document is represented literally as a 'bag' of its own words, disregarding word orders, sequences and grammar.

```
In [55]: from sklearn.feature_extraction.text import CountVectorizer
```

```
cv = CountVectorizer(min_df=0., max_df=1.)
cv_matrix = cv.fit_transform(df_sm['text_nav'])
cv_matrix = cv_matrix.toarray()
cv_matrix
```

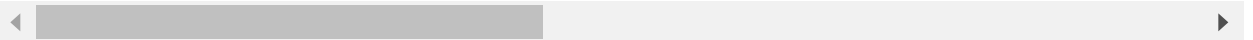
```
Out[55]: array([[0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                ...,
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

```
In [56]: # get all unique words in the corpus
vocab_bagowords = cv.get_feature_names()
# show document feature vectors
pd.DataFrame(cv_matrix, columns=vocab_bagowords)
```

Out[56]:

	aaawwww	aaru	abay	abc	ability	able	academy	account	accountant	ace	ache	ack	ac
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0
...
787	0	0	1	0	0	0	0	0	0	0	0	0	0
788	0	0	0	0	0	0	0	0	0	0	0	0	0
789	0	0	0	0	0	0	0	0	0	0	0	0	0
790	0	0	0	0	0	0	0	0	0	0	0	0	0
791	0	0	0	0	0	0	0	0	0	0	0	0	0

792 rows × 1943 columns



Micro dataframe visualization experiment

To satisfy my curiosity, I'm going to make up a teeny weeny little dataframe, to see if I can see some values in one of these arrays as it gets previewed here in the pandas dataframe thing. We just see the corners. All zeroes. Let's see if it looks more satisfying with just a handful of rows...

```
In [57]: df_micro = df_sm[1:10]
cv_micro = CountVectorizer(min_df=0., max_df=1.)
cv_micro_matrix = cv_micro.fit_transform(df_micro['text_nav'])
cv_micro_matrix = cv_micro_matrix.toarray()
cv_micro_matrix
```

```
Out[57]: array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0],
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0,
0, 1, 0, 1, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2,
1, 0, 0, 1, 0],
[0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 2, 0, 0, 0, 0, 1, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0, 1, 0, 0, 2],
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0],
[1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0],
[0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 0]], dtype=int64)
```

```
In [58]: # get all unique words in the corpus
vocab_micro = cv_micro.get_feature_names()
# show document feature vectors
pd.DataFrame(cv_micro_matrix, columns=vocab_micro)
```

```
Out[58]:
```

	age	anal	black	cancer	die	doctor	dread	face	far	farrah	fawcett	feather	get	go	g
0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	
1	0	0	0	1	0	0	0	0	0	1	1	0	0	0	
2	0	0	1	0	0	0	0	0	0	0	0	0	0	1	
3	0	0	0	0	0	0	0	0	0	1	1	0	0	0	
4	0	0	0	0	0	0	0	1	1	0	0	0	0	0	
5	0	0	0	0	0	1	1	0	0	0	0	0	1	2	
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
7	1	0	0	1	1	0	0	0	0	0	0	0	1	0	
8	0	1	0	1	1	0	0	0	0	1	1	0	0	0	

9 rows × 51 columns

Oh, that's much more gratifying!

While it's clearly useless from a practical perspective, it's nice to be able to see some numbers that aren't zeroes and let me know that the code is doing the thing I expect the code to be doing. I've got to try this on the next ones, too!

Bag of N-Grams model

A word is just a single token, often known as a unigram or 1-gram. We already know that the Bag of Words model doesn't consider order of words. But what if we also wanted to take into account phrases or collection of words which occur in a sequence? N-grams help us achieve that. An N-gram is basically a collection of word tokens from a text document such that these tokens are contiguous and occur in a sequence. Bi-grams indicate n-grams of order 2 (two words), Tri-grams indicate n-grams of order 3 (three words), and so on. The Bag of N-Grams model is hence just an extension of the Bag of Words model so we can also leverage N-gram based features. The following example depicts bi-gram based features in each document feature vector.

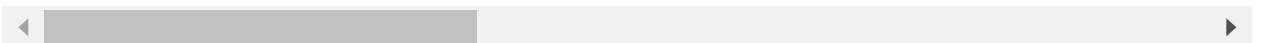

```
In [59]: # you can set the n-gram range to 1,2 to get unigrams as well as bigrams
bv = CountVectorizer(ngram_range=(2,5))
bv_matrix = bv.fit_transform(df_sm['text_nav'])

bv_matrix = bv_matrix.toarray()
vocab_ngrams = bv.get_feature_names()
pd.DataFrame(bv_matrix, columns=vocab_ngrams)
```

Out[59]:

	aaawwww farrah	aaawwww farrah ri	aaawwww farrah ri die	aaawwww farrah ri die miss	aaru hithavaru	aaru hithavaru ninage	aaru hithavaru ninage moovarolage	aaru hithavaru ninage moovarolage vote
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0
...
787	0	0	0	0	0	0	0	0
788	0	0	0	0	0	0	0	0
789	0	0	0	0	0	0	0	0
790	0	0	0	0	0	0	0	0
791	0	0	0	0	0	0	0	0

792 rows × 11518 columns



This gives us feature vectors for our documents, where each feature consists of a bi-gram representing a sequence of two words and values represent how many times the bi-gram was present for our documents.

And now with the Micro Dataframe:

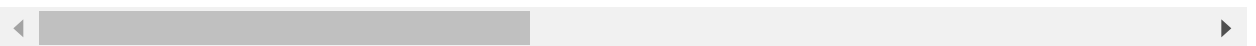
```
In [60]: bv_micro = CountVectorizer(ngram_range=(2,2))
bv_micro_matrix = bv_micro.fit_transform(df_micro['text_nav'])

bv_micro_matrix = bv_micro_matrix.toarray()
vocab_micro = bv_micro.get_feature_names()
pd.DataFrame(bv_micro_matrix, columns=vocab_micro)
```

Out[60]:

	age grandmother	anal die	black swimming	cancer anal	cancer farrah	cancer meet	die get	die think	doctor med	dread go	face time	far want	f
0	0	0	0	0	0	0	0	0	0	0	0	0	
1	0	0	0	0	1	0	0	0	0	0	0	0	
2	0	0	1	0	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	1	1	
5	0	0	0	0	0	0	0	0	1	1	0	0	
6	0	0	0	0	0	0	0	0	0	0	0	0	
7	1	0	0	0	0	1	1	0	0	0	0	0	
8	0	1	0	1	0	0	0	1	0	0	0	0	

9 rows × 58 columns



TF-IDF Model

There are some potential problems which might arise with the Bag of Words model when it is used on large corpora. Since the feature vectors are based on absolute term frequencies, there might be some terms which occur frequently across all documents and these may tend to overshadow other terms in the feature set. The TF-IDF model tries to combat this issue by using a scaling or normalizing factor in its computation. TF-IDF stands for Term Frequency-Inverse Document Frequency. There are multiple variants of this model but they all end up giving quite similar results.

```
In [61]: from sklearn.feature_extraction.text import TfidfVectorizer
```

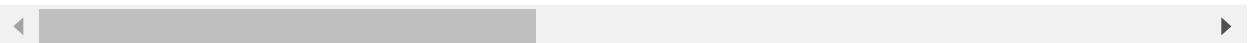
```
tv = TfidfVectorizer(min_df=0., max_df=1., use_idf=True)
tv_matrix = tv.fit_transform(df_sm['text_nav'])
tv_matrix = tv_matrix.toarray()

vocab_tfidf = tv.get_feature_names()
pd.DataFrame(np.round(tv_matrix, 2), columns=vocab_tfidf)
```

Out[61]:

	aaawwww	aaru	abay	abc	ability	able	academy	account	accountant	ace	ache	ack
0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
...
787	0.00	0.00	0.31	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
788	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
789	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
790	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
791	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

792 rows × 1943 columns



Gotta try that with the Micro Dataframe...

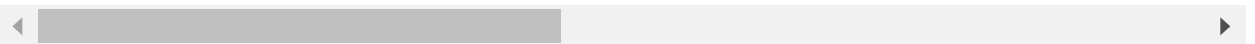
```
In [62]: tv_micro = TfidfVectorizer(min_df=0., max_df=1., use_idf=True)
tv_micro_matrix = tv_micro.fit_transform(df_micro['text_nav'])
tv_micro_matrix = tv_micro_matrix.toarray()

vocab_micro = tv_micro.get_feature_names()
pd.DataFrame(np.round(tv_micro_matrix, 2), columns=vocab_micro)
```

Out[62]:

	age	anal	black	cancer	die	doctor	dread	face	far	farrah	fawcett	feather	get	go
0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.25	0.00	0.38	0.00	0.00
1	0.00	0.00	0.00	0.42	0.00	0.00	0.00	0.00	0.00	0.37	0.42	0.00	0.00	0.00
2	0.00	0.00	0.33	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.28
3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.46	0.52	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.30	0.30	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.00	0.26	0.26	0.00	0.00	0.00	0.00	0.00	0.22	0.43
6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
7	0.41	0.00	0.00	0.30	0.35	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.35	0.00
8	0.00	0.49	0.00	0.36	0.41	0.00	0.00	0.00	0.00	0.32	0.36	0.00	0.00	0.00

9 rows × 51 columns



=====

Feature Selection

=====

Feature Selection is an essential step, which allows us to identify which features are most important to the predictive abilities of our models. Filter methods of Feature Selection involve looking at individual features in isolation, giving them a score by which they can be ranked in terms of their usefulness. We use Univariate Chi-squared statistical tests. Wrapper methods of Feature Selection consider sets of features in combination, which can give deeper insights into which features to select given their interactions and correlations with one another. We use Recursive Feature Elimination and Bagged Decision Trees for this type of feature selection. There are also Embedded Feature Selection techniques, however, these are done in concert with the modeling phase of the project, and if they will be attempted, they will be attempted during the modeling phase of the project.

Filter Method

The scikit-learn library provides the `SelectKBest` class that uses the chi squared (χ^2) statistical test to select the best features

Our Bag of Words vectorization has provided us with over a thousand features. `SelectKBest` can identify which of these features are most strongly correlated with our sentiment label. We produce a new Bag of Words containing only the features `SelectKBest` determines are most important.

```
In [63]: from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
# select the 500 features that have the strongest correlation to a class from the
# original thousands of features
selector = SelectKBest(chi2, k=500)
selected_features = \
bow_selected = selector.fit(cv_matrix, df_sm['sentiment']).get_support(indices=Tr
selected_features
```

```
Out[63]: array([ 17,  18,  23,  24,  25,  26,  34,  40,  47,  48,  55,  59,
        68,  70,  78,  83,  88,  89,  92,  98, 101, 105, 121, 122,
       127, 132, 134, 138, 142, 145, 146, 156, 158, 161, 163, 175,
       177, 188, 189, 192, 193, 194, 195, 213, 215, 225, 233, 235,
       241, 242, 243, 254, 267, 270, 275, 286, 299, 303, 307, 308,
       317, 335, 341, 361, 364, 366, 374, 388, 393, 405, 413, 425,
       431, 435, 436, 437, 447, 461, 462, 467, 469, 472, 473, 474,
       483, 492, 493, 494, 496, 513, 517, 527, 536, 537, 538, 550,
       560, 564, 565, 572, 573, 574, 578, 607, 614, 628, 642, 643,
       659, 665, 670, 671, 684, 685, 686, 698, 702, 722, 730, 731,
       736, 739, 740, 750, 751, 757, 767, 772, 773, 776, 784, 787,
       794, 796, 802, 804, 806, 815, 816, 819, 820, 822, 823, 824,
       826, 839, 851, 856, 875, 880, 882, 885, 895, 902, 907, 915,
       923, 931, 932, 936, 938, 941, 944, 946, 949, 957, 959, 960,
       965, 972, 984, 989, 991, 994, 996, 999, 1004, 1006, 1009, 1012,
      1019, 1029, 1039, 1050, 1054, 1056, 1057, 1066, 1068, 1072, 1079, 1080,
      1088, 1095, 1096, 1099, 1107, 1125, 1145, 1147, 1149, 1150, 1152, 1154,
      1173, 1185, 1199, 1201, 1202, 1208, 1229, 1244, 1250, 1264, 1279, 1286,
      1296, 1310, 1312, 1316, 1325, 1332, 1336, 1340, 1349, 1358, 1370, 1373,
      1380, 1383, 1384, 1385, 1396, 1399, 1405, 1406, 1417, 1423, 1444, 1445,
      1452, 1475, 1478, 1489, 1492, 1494, 1495, 1496, 1498, 1501, 1503, 1504,
      1506, 1507, 1509, 1510, 1511, 1514, 1516, 1518, 1520, 1524, 1525, 1526,
      1529, 1530, 1532, 1535, 1539, 1542, 1545, 1546, 1549, 1550, 1552, 1553,
      1557, 1559, 1561, 1562, 1563, 1564, 1566, 1568, 1569, 1570, 1571, 1572,
      1573, 1575, 1577, 1578, 1579, 1580, 1583, 1585, 1587, 1589, 1590, 1592,
      1596, 1598, 1600, 1603, 1604, 1608, 1610, 1611, 1613, 1614, 1615, 1617,
      1619, 1620, 1621, 1622, 1623, 1624, 1625, 1627, 1629, 1630, 1633, 1636,
      1637, 1641, 1643, 1645, 1646, 1647, 1648, 1650, 1652, 1653, 1654, 1655,
      1656, 1659, 1660, 1661, 1662, 1664, 1665, 1667, 1668, 1669, 1671, 1672,
      1675, 1676, 1678, 1679, 1680, 1681, 1683, 1684, 1685, 1686, 1687, 1689,
      1692, 1695, 1696, 1697, 1698, 1700, 1701, 1704, 1705, 1707, 1716, 1717,
      1718, 1721, 1728, 1729, 1731, 1733, 1734, 1735, 1738, 1740, 1745, 1746,
      1748, 1750, 1759, 1761, 1762, 1763, 1764, 1766, 1768, 1771, 1772, 1773,
      1775, 1776, 1777, 1778, 1779, 1780, 1781, 1782, 1783, 1784, 1788, 1789,
      1790, 1793, 1794, 1795, 1797, 1799, 1800, 1801, 1803, 1804, 1805, 1807,
      1808, 1809, 1812, 1814, 1816, 1817, 1818, 1821, 1822, 1823, 1826, 1828,
      1829, 1834, 1836, 1837, 1838, 1839, 1840, 1844, 1845, 1846, 1848, 1850,
      1851, 1853, 1854, 1855, 1856, 1858, 1860, 1861, 1862, 1863, 1865, 1866,
      1870, 1871, 1872, 1873, 1874, 1875, 1876, 1877, 1878, 1880, 1885, 1887,
      1888, 1889, 1896, 1898, 1899, 1901, 1903, 1904, 1905, 1906, 1907, 1908,
      1909, 1913, 1914, 1916, 1918, 1919, 1920, 1921, 1923, 1924, 1928, 1930,
      1931, 1933, 1934, 1935, 1938, 1939, 1940, 1941], dtype=int64)
```

Interpreting the Selected Features

That array of numbers tells us which indices of the Bag of Words vectorization were deemed most important by SelectKBest. Kind of boring on its own. It would be much more gratifying to see the actual words it has decided are most important.

View list of words selected by SelectKBest

```
In [64]: for x in selected_features:
         print(vocab_bagowords[x], end=' ')
```

add addict age agree ahh ahhh alexa amazing angel animal answer appreciate ask
asleep attack austin awesome awful awww baby back bad battle bb beat bed beer b
elieve beyonce big bike bless blessing blog blue border boring boy boyfriend br
ave brazil break breakfast buddy buffy bus buy bye can cancel cancer carry chal
lenge charlie check chocolate class client clothe club college congrat cookie c
rack crappy crazy cruise cvs dad day death design di die diego different discov
er double doubt drag dread drink drinking drive dun easter easy eat ed enjoy en
tire evening excellent excite excited face family farah farrah favorite fawcett
fawcetts feather flip food friday funeral funny gettin gig give glad good goodb
ye goodnight great grow hahaha happen happy hate hav have healthy hear hehe hig
h hill hills hit holla home honor hook hospital hot hour hungry hurt hybrid ice
icon idea idk ihad india interview ipod jill join jon jordan karma keith kid kn
ight ko lady lame last late laugh lauren lay lazy leave leaving less life link
lol long look loop lose loss love lovely luck lunch mad man marry mcmahon mediu
m meet meeting message miami miley miss missed mo monday money moon movie must
nervous new news next nice night noooooo nuts old one oneal option panama pass
peace person photoshop picture planet pool pop post prayer problem program prou
d put quot rain random reach ready real reason regret remember response rest ri
p rock sad sadden san seem sell series set sew sexy sezdawg share sheldawg shen
agian shhhhhhhhhh shiner shirt sho shoe shoedazzle short show shud sicilian sie
tar sigh sighting simple sin singing sister skim skype sleeeeeeeeeeeep sleep slus
hie small smell smile snow society soiree someone somewhere song sooo sooooooo
sore sorrry sorry soul sound spacecowboyi speak special speechless spelling spi
ke spiritual sport spring square squirrel stalk star stat stellar step stinkin
stone stop store storm story straiight strawberry stray stress stroke strong s
truggle stuck stuff style submitted suck suggest sulk sunburn sunisa super supp
ort supportstructure suppose surprise survive sushi suspect sutter suv sweat sw
eet sweetheart swim swopper sydney tafe tag tagalog talent talk tar tattoo taxi
s tbs te tea tear tech tedium teka tell temporary test tha thank thanks thans t
heatre theresa thinking tho thousand time timeless tina tipsy today toilet tomo
rrow tonight tool tooo top topping track traffic trainer transformer trusty try
na tryout tub tuck turn tvc tweetdeck tweetie tweetland twilight twin twit twit
t twitter twitterer twitterpeeps type uber udah ugh uhhh ulit unc uncle underne
ith ung unison universe unpack upcome update upload upset upto ur usb usmeall u
tility va vacuum van vegas vegetable venue version vibrant vivian vocal vote wa
gah wait waiting walle wallpaper walmart want wash waste watch watchin water wa
y weak website wedding wednesday week weird welcome wer wereare west westney wh
atcha whine white whole whoops wife windows winks winter wish woman wonderful w
oo woot wordahead work workout world worry would wow write wrong wtf www xd xox
o xxxx yay yayyy yesterday ym young youtube you're yrold yvonne zealand zhen zo
ne

Wrapper Method

Recursive Feature Elimination

The Recursive Feature Elimination (or RFE) method works by recursively removing attributes and building a model on those attributes that remain. It uses the model accuracy to identify which attributes (and combination of attributes) contribute the most to predicting the target attribute.

```
In [65]: from sklearn.feature_selection import RFE
        from sklearn.linear_model import LogisticRegression
```

```
In [66]: model = LogisticRegression(solver='lbfgs', max_iter=6)
        rfe = RFE(model, 500)
        fit = rfe.fit(cv_matrix, df_sm['sentiment'])
        print("Num Features: %d" % fit.n_features_)
        print("Selected Features: %s" % fit.support_)
        print("Feature Ranking: %s" % fit.ranking_)
```

Num Features: 500

Selected Features: [False False False ... False False False]

Feature Ranking: [1316 743 1141 ... 455 899 83]


```
In [67]: i=0
for x in fit.support_:
    if x:
        print(vocab_bagowords[i], end=' ')
    i=i+1
```

adam add addict afternoon age agree ahh ahhhh amazing amo angel animal animatio
n answer appreciate arne aroundd asia ask asleep ass assignment attack attracti
ve awake awesome awful awww babiesss baby background backup bad bb be beat bed
big bike blackberrys blessing blue bog bonjour boo border bored boring boy boyf
riend brate brazil break breakfast brussels buddy buffy bus butt buy bye cable
can cancel cancer car care carless carry ce challenge change charlie check chee
r cheers chilling chilly chocolate cider class clean cleaning client cliff clos
e clothe cold college complaint confuse congrat cookie cool could crack crappy
crazy curious cvs cw dad dam damn darn day death deck di die diego different di
scourse discover do dollar double doubt draag drag dread dream drink drinking d
rive dump dun dundrum dunno easter easy eat empty endless enjoy enough enter en
tire evening example excite excited fabulous face fail family fan farah farrah
fat favorite fawcett feel fill finish follow follower food forgot friday friend
fritz frustrating fuck funeral funny gal gettin ghost gig give glad gmail godpl
ease good goodbye goodnight goody great grebel grow guess guy gym haha hahaha h
ahahaha happen happy hard hate hates hav have havoc head healthy hear hehe hell
high hill hills hit holidayholiday holla holy home honey hoo hook hospital hot
hour house hungry hurt hybrid ice icon ihad inbox india inhibit interview iphon
e ipod jay jill join jon jordan karma keith kick kid knight know knowwww ko lab
lady lame land laptop last laugh lauren lay lazy le leav leave leaving let life
link listen listening living lol long look loop lose loss lost love loved lovel
y luck lunch mad mag magner man marry maw mean medium meee meet meeting mess me
ssage miami midnight miley miss missed mmorrow mms mo monday money month moon m
orning movie much music muslim must need nervous new news night nobodys noooooo
nooooooooo note number nuts obsess offer old omg one oneal oppress option paper p
ass peace peddie people peoplethe person phoenix photoshop pics picture planet
play please poodle pool pop post press problem program proud put quizez quot q
uothello rain random reach reading ready real reason regret remember repair rep
ly response rest return rip ripfarrahfawcett rock ruin sad san say sedang seem
sell sense seperate serial series set settle shirt shit shoot sigh simple singe
sit site small smell smile snow sobriety society song sooo sorry soul sound spa
cecowboyi speak spending spring squirrel stalk star start step stevens stop sto
rm story storys strange stress strong stuck stuff suck summer sunburn sure suv
sweet swim take talent talk tap te tear tell test texan thank theresa thing tho
tiffaney til time tingle tipsy today toilet tomorrow tonight touchy trainer tre
kkie trouble tvc tweetdeck twilight twitt twitter uber ugh umbrella update uplo
ad ur use vacuum vegas version vote wait wallpaper want warm wat watch watchin
water way website week welcome whole whoops wid wife will win wish wks woman wo
nder wonderful word work workout would wreak write wrong wtf xoxo yay yesterday
young

More Wrapper Methods

Bagged Decision Trees - ExtraTreesClassifier classifier class

```
In [68]: from sklearn.ensemble import ExtraTreesClassifier
```

```
In [69]: # feature extraction
model_bagged = ExtraTreesClassifier(n_estimators=10)
model_bagged.fit(cv_matrix, df_sm['sentiment'])
# np.set_printoptions(threshold=np.inf)
print(model_bagged.feature_importances_)
# np.set_printoptions(threshold=200)

[0.00e+00 0.00e+00 0.00e+00 ... 5.74e-06 0.00e+00 5.75e-04]
```

```

In [70]: i=0
n=0
min_imp = 0.00041
min_starred = 0.004
for x in model_bagged.feature_importances_:
    if x>min_imp:
        if x>min_starred:
            print('***', end='')
            print(vocab_bagowords[i], end=' ')
            n=n+1
        i=i+1
print(' ')
print(' ')
print('Number of Features with minimum importance ', min_imp, ': ', n)

```

ability able ad add afraid agree ahhhh amazing angel animation annoying answer
 r arne ask ass assignment attack awesome awful awww babbies back background
 ***bad bb be beautiful bed beef big bike blackberrys bless blessing blew bog
 boo bored boring boyfriend break brush brussels buddy bulky bun butt buy bye
 california call can cancel cancer car care caring carless caro carry cel chan
 ge charlie cheer cheers chill chilly chocolate citizen clean cliff close clot
 he color come complaint computer congrat contest cooler could coupon craappy
 crack crap crappy curse cut cw cybercommand dad dam damn dance dang darn ***d
 ay daylol deck ***die diego different dinner direct discourse dnt do dollar d
 ownload drag dread drink dude dun dundrum dungeon dunno ***easy eat effin ema
 il employe ento erin essay esurance evening everyone excited eye face fail fa
 ll fan farah ***farrah favorite ***fawcett fawke fear ***feel feeling fellow
 find finish follower food forget forgot friday friend fritz frustrating fuck
 fun funeral game germany ***get gig gina give ***glad gmail go godplease ***g
 ood goodbye ***goodnight grandma ***great grebel guess guy gymnastic gyms hah
 a hahahaha hangover happen happy hard ***hate hates hatin have head headache
 hear hehe hell heyyyyyy home hoo hook hope horrible hot ***hour hugs ***hungr
 y hurt ice icon idea ill impossible ina inhibit injury iphone ipod jealous ji
 ll jon joolz karma kid kind knob know knowww lame land laptop ***last laugh l
 ay least leav leave let levon library life lil line link listen little loan l
 ol lonely loner long look loop lose loss lost ***love loved lovely luck ***lu
 nch lybrary mad make man many married maw may mean medium meee meet meeting m
 ess message miley mine ***miss missed misses mms mom monday month moon mornin
 g movie moviess much munroe must nasty need new ***news next night nobodys no
 n noooooo noooooooo oaf obsess offer old omg oneal onli ontario overheated par
 agon ***pass ***peace people peoplethe persiankiwi person pick pics picture p
 lace plain play please pm poodle pool pop post poster power ppl prague pray
 prayer problem quot rachel rain reach read ***ready real reduce regret rememb
 er repair ***reply ***rest return ***rip ripfarrahfawcett rock room ***sad sa
 y school scratch screw second sedang see seem self sell sense seperate septem
 ber series set settle shell shoot shop shoulder show sick sigh sinus sister s
 ite sjo slack sleeeeeeeeeep sleep slg small smell smile snow software song **
 *sorry sound spanish speak spend spending spoilt spring stand star start stat
 stay stealth steph stevens stop ***storm storys strange stuck stuff submitted
 suck sunburn suppose surprised swayze sweet swim take talk tap te tell test t
 exan ***thank theresa think tho tiffaney til time tip tipsy tire tired tnite
 today toilet tomorrow tonight touchy tourist track trainer transfer transform
 er trap trouble try turn tvc tweet tweetpic ***twitter uggghh ***ugh update u
 r usmeall usual veil verify version video vip vote wait walk wanna ***want **
 *watch way weather website ***week well whole wife windows ***wish woman wond
 er wonderful woohoo word ***work world would wreak wrist wrong xoxo yay yeano

t year yep yesterday yike ym young zta

Number of Features with minimum importance 0.00041 : 501

Note that the stars (*) denote features that were given much higher importances.**

Feature Selection on TF-IDF encoded features

Filter Method - SelectKBest

```
In [71]: selector_tfidf = SelectKBest(chi2, k=500)
selected_features_tfidf = \
selector_tfidf.fit(tv_matrix, df_sm['sentiment']).get_support(indices=True)
selected_features_tfidf
```

```
Out[71]: array([  0,   3,   7,  14,  17,  18,  23,  24,  25,  26,  27,  34,
  39,  40,  41,  44,  47,  48,  51,  54,  55,  58,  59,  64,
  68,  70,  73,  78,  83,  88,  89,  92,  94,  98, 105, 111,
 114, 117, 119, 122, 127, 130, 132, 134, 138, 142, 146, 153,
 154, 156, 158, 159, 161, 163, 168, 175, 176, 177, 183, 188,
 189, 190, 192, 193, 194, 195, 200, 207, 210, 211, 212, 213,
 229, 233, 235, 242, 243, 249, 254, 262, 267, 269, 270, 276,
 286, 291, 299, 303, 307, 308, 317, 326, 334, 335, 341, 345,
 361, 363, 364, 374, 380, 388, 389, 393, 394, 408, 413, 414,
 421, 422, 425, 430, 431, 435, 436, 437, 440, 447, 450, 456,
 462, 467, 469, 472, 473, 484, 488, 492, 493, 494, 496, 497,
 499, 500, 513, 517, 518, 527, 536, 537, 538, 550, 560, 564,
 565, 568, 572, 573, 574, 578, 607, 611, 613, 614, 619, 624,
 628, 630, 631, 633, 638, 642, 643, 652, 653, 659, 660, 665,
 671, 676, 680, 681, 684, 685, 686, 688, 698, 702, 706, 710,
 712, 713, 721, 722, 728, 730, 731, 735, 736, 742, 748, 749,
 750, 751, 757, 760, 766, 767, 772, 773, 776, 782, 784, 785,
 787, 794, 796, 802, 804, 806, 814, 815, 816, 820, 822, 824,
 826, 833, 836, 844, 845, 848, 851, 853, 856, 862, 864, 871,
 875, 880, 882, 885, 895, 898, 901, 902, 907, 915, 922, 928,
 931, 932, 936, 938, 941, 943, 944, 946, 949, 951, 954, 957,
 959, 960, 961, 962, 967, 972, 979, 984, 986, 988, 990, 991,
 994, 996, 999, 1004, 1006, 1009, 1012, 1014, 1015, 1018, 1019, 1031,
1039, 1040, 1043, 1046, 1050, 1054, 1055, 1059, 1066, 1068, 1072, 1078,
1079, 1080, 1081, 1082, 1087, 1088, 1089, 1091, 1094, 1095, 1096, 1097,
1099, 1107, 1118, 1125, 1127, 1137, 1145, 1147, 1149, 1154, 1157, 1162,
1172, 1173, 1174, 1180, 1181, 1185, 1186, 1187, 1199, 1201, 1202, 1204,
1208, 1215, 1222, 1224, 1226, 1229, 1236, 1244, 1250, 1263, 1264, 1273,
1279, 1284, 1285, 1286, 1294, 1296, 1297, 1300, 1310, 1312, 1316, 1325,
1327, 1332, 1336, 1340, 1349, 1370, 1373, 1380, 1383, 1384, 1385, 1396,
1397, 1399, 1405, 1406, 1411, 1417, 1418, 1423, 1426, 1444, 1445, 1448,
1452, 1465, 1472, 1475, 1478, 1482, 1485, 1489, 1492, 1503, 1504, 1507,
1515, 1518, 1525, 1529, 1550, 1552, 1553, 1556, 1557, 1559, 1564, 1566,
1569, 1571, 1572, 1573, 1577, 1578, 1592, 1596, 1598, 1604, 1606, 1608,
1611, 1614, 1615, 1616, 1621, 1625, 1627, 1629, 1632, 1633, 1635, 1641,
1657, 1660, 1662, 1672, 1680, 1681, 1683, 1687, 1692, 1693, 1695, 1696,
1701, 1705, 1714, 1715, 1716, 1721, 1722, 1729, 1730, 1738, 1742, 1748,
1761, 1762, 1768, 1771, 1772, 1775, 1776, 1778, 1779, 1783, 1788, 1793,
1794, 1800, 1804, 1805, 1807, 1808, 1814, 1817, 1818, 1822, 1828, 1837,
1839, 1845, 1848, 1853, 1854, 1855, 1860, 1863, 1865, 1866, 1873, 1876,
1877, 1880, 1885, 1889, 1896, 1899, 1904, 1905, 1906, 1907, 1913, 1914,
1916, 1919, 1920, 1923, 1925, 1928, 1931, 1934], dtype=int64)
```

```
In [72]: for x in selected_features_tfidf:
        print(vocab_tfidf[x], end=' ')
```

aaawwww abc account actress add addict age agree ahh ahhh ahhhh alexa amanda am
azing american anal angel animal anne anotha answer apply appreciate aroundd as
k asleep asscancer attack austin awesome awful awww awww baby bad bah balcony
barrie bash bb beat becca bed beer believe beyonce bike blackberrys blame bless
blessing blew blog blue bog border bored boring bowling boy boyfriend brand bra
ve brazil break breakfast brissa brother brussels brutha bryon buddy butler buy
bye cancel cancer caring carry cel challenge change charlie cheer chocolate chu
bbx class client clothe club college common confuse congrat cookie coool crack
crap crappy cruise cupcakes cvs cw dad dam daylol death debby depend deposit de
sign devunity di die diego different direct discover dnt dollar doubt drag drea
d drink drinking dundrum dyeing easter easy eat ed edinburgh effect effin enjoy
entire ento evening excellent excite excited face family farah farrah faucett f
avorite fawcett fawcetts feather flip follow following food forgot freakin frid
ay friended friendster fritz fucking funeral funny garmo geeke gettin gf gig gl
ad gmail godplease going good goodbye goodnight goooooood great grow guess gunn
na gyal gym hahah hahaha hang happen happy hash hate hawaii headline heah healt
hy hear hehe hellz heyyyyyy high hill hills hit holidayholiday holla hollywood
home honor hook hospital hot hour hunch hungry hurt ice icon idk ihad ina incen
diary inspiration inspire interesting interview invite ipod jakarta jane jennif
er jill join jon jordan karma kay keepin keith kid knight knowww la lady lame l
ast late laugh laundry lauren lay lazy lbda learn leave leaving less let levon
lil link lj lol london loner longtime look loop lose loss love lovely luck lunc
h lurker luv mac mad manga marry martini mate maw mcmahon medium meee mello mes
sage miami miley mirror miss missed misses misssssssss mms mo moccasin model m
ommy monday money monroe moon movie munroe must mutha necc nervous new news nig
ht nightttime ninguem noodlebox noooooo noooooooo notice noticed nuts nyc nyet ol
d one oneal ontario option overheated packing paige paint panama paragon pass p
eace persiankiwi person phew photoshop picnic pics picture plain planet plannin
g pleaase pool pop post prayer prepare problem program proud put rain random re
ach ready real reason regret rehearsal remember response rest ri rip ripfarrahf
awcett rock role sad sadden sadness san scratch sedang seem sell sense septembe
r series set shenagian shhhhhhhhhh shirt shoulder shud sigh simple small smell
smile snap snow society song sooo sore sorry soul sound speak special squirrel
stalk star step stevens stinkin stop storm story storys stress stuck stuff styl
e succumbed suck suffering sunburn sux sweet swim talk te tea tear tell test te
xan tha thank theresa tho tiffaney til time tipsy tire toilet tom top touchy tr
ainer tryna tryout tvc tweetdeck tweetie twilight twin twitt twitter uber ugh u
nc uncle universe update upload upset upto usmeall va vacuum vegas version vote
wait wallpaper want watch watchin water website week weird welcome westney whit
e whole wife windows wish woman woo work workout world worry write wrong wtf xd
xoxo yay yeanot yesterday young you%re

RFE on TF-IDF vectors

```
In [73]: model_tfidf = LogisticRegression(solver='lbfgs', max_iter=6)
rfe = RFE(model_tfidf, 500)
fit_tfidf = rfe.fit(tv_matrix, df_sm['sentiment'])
print("Num Features: %d" % fit.n_features_)
print("Selected Features: %s" % fit.support_)
print("Feature Ranking: %s" % fit.ranking_)
```

Num Features: 500

Selected Features: [False False False ... False False False]

Feature Ranking: [1316 743 1141 ... 455 899 83]

```
In [74]: i=0
for x in fit_tfidf.support_:
    if x:
        print(vocab_bagowords[i], end=' ')
    i=i+1
```

able account add addict afraid age agree ahh ahhh ahhhh amanda amazing angel answer apply appreciate aroundd ask asleep ass attack awesome awful awww baby bad bah barrie bash bb be beat bed beer big bike blackberrys blame bless blessing blew blog blue bog border bored boring boy boyfriend brand brazil break breakfast brussels buddy buy bye can cancel cancer car care caring carry cat challenge change charlie check cheer chocolate class close clothe club college common confuse congrat cookie cooler could crack crap crappy cruise cupcakes cut cvs cw dad dam damn day daylol death debby di die diego different dnt dollar double doubt drag dread drink drinking drive dun dundrum dyeing easter easy eat effect effin english enjoy entire ento essay evening excellent excite excited fabulous face family fan farah farrah favorite fawcett feel flip follow food forgot friday friend friended fritz fuck fun funeral funny garmo geeke gettin gig give glad gmail go godplease good goodbye goodnight goooooood great guess gunnna gyal gym habanero haha hahah hahaha hand hang happen happy hash hate hav have head healthy hear hehe help heyyyyyy high hill hills hit holidayholiday holla holy home hook hospital hot hour hunch hungry hurt hybrid ice icon idk ihad ina inspire interesting interview invite ipod jakarta jane jealous jennifer jill join jon jordan karma keepin keith kid knight know knowww lady lame land laptop last late laugh laundry lauren lay lazy lbda leave leaving let levon life lil line link listen lj lol london loner long longtime look loop lose loss lost love lovely luck lunch lurker mac mad man manga married marry martini mate maw medium meet message miami midnight miley mirror miss missed misses mmorrow mms mo moccasin mommy monday money month moon morning movie munroe must nasty necc need nervous new news night ninguem nobodys noodlebox noooooo nooooooo notice nuts nyc old one oneal ontario option overheated packing paint paragon pass peace people persiankiwi person phew photoshop picnic pics picture plain planet planning please pool pop post problem program proud put quizez quot rachel rain random reach reading ready real reason regret remember reply response rest rip ripfarr ahfawcett rock sad sadden san saving say scratch screw second sedang see seem sell sense september set shenagian shirt shoulder show sigh sinus site sleep small smell smile snow society song sooo sorry sound spanish speak special spend spending spring squirrel stalk stand star start step stevens stinkin stop storm story storys strange stress stuck stuff style suck sunburn sure suv sweet swim take talk te tea tell test texan thank theresa tho tiffaney til time tipsy tire today toilet tom tomorrow tonight touchy trainer trouble tryna tvc tweetdeck twilight twitt twitter uber ugh unc uncle update upload upset upto va vacuum vegas version vote wait wallpaper wanna want watch watchin water way website week welcome well westney white whole wife win windows wish woman wonder wonderful wo o work workout worry would write wrong wtf xd xoxo yay yeanot yesterday young you%re

Selecting TF-IDF Features with Bagged Decision Trees

This represents our most sophisticated feature selection method, being used with our most sophisticated feature engineering method.

Earlier, in the Feature Engineering section, we vectorized our text data using the Term Frequency - Inverse Document Frequency method. This resulted in our documents being represented as vectors with thousands of dimensions. Here we will select the most statistically relevant dimensions using a Wrapper Method known as an Extra Trees Classifier, which is an example of Bagged Decision Trees.

```
In [75]: model_bagged_tfidf = ExtraTreesClassifier(n_estimators=20)
model_bagged_tfidf.fit(tv_matrix, df_sm['sentiment'])
# np.set_printoptions(threshold=np.inf)
print(model_bagged_tfidf.feature_importances_)
# np.set_printoptions(threshold=200)

[0.00e+00  2.46e-06  0.00e+00  ...  2.33e-06  0.00e+00  2.17e-04]
```

```
In [76]: i=0
n=0
min_imp = 0.00040
min_starred = 0.003
for x in model_bagged_tfidf.feature_importances_:
    if x>min_imp:
        if x>min_starred:
            print('***', end='')
            print(vocab_tfidf[i], end=' ')
            n=n+1
        i=i+1
print(' ')
print(' ')
print('Number of Features with minimum importance ', min_imp, ': ', n)
```

ability able ack add addict afraid afternoon age agree ahhhh ***amazing ***an
 gel answer aroundd asia ask ass assignment attack attractive awesome awful aw
 ww background backup ***bad banana barrie bb ***be beautiful ***bed blackberr
 ys blew bomb bonjour border bored boring boyfriend brazil break breakfast bri
 ng brussels buddy butt buy bye cable can cancel ***cancer car care caring car
 less caro cat ce cel ***change ***charlie cheer chill chocolate citizen cliff
 close coffee cold college come confuse congrat could crap crappy crazy curse
 ***cut cw cyrus dad dam ***damn dang ***day daylol ddi death deck ***die diff
 erent disappointed dnt do double dread dream drink drive dun dundrum dunno **
 *easy eat ed email emailing endless english enjoy ento essay everyone excite
 excited eye face fall famous fan farah ***farrah ***favorite ***fawcett feath
 er ***feel fellow fight find finish first flip flippin follow food forgot fri
 day friend fritz fuck full fun funeral funny game get gettin gig girl give gl
 ad glastonbury ***gmail ***go god godplease ***good goodbye ***goodnight gran
 dfather grandma ***great guess guy habanero haha hahahaha hair happen happy h
 ard ***hate have head heah hear help hill hills hit hold holidayholiday home
 honey honor hook hope hospital ***hot ***hour ***hungry ***hurt hybrid ice ic
 on idea idk idkk ill inhibit internet iphone ipod jakarta jane jay ***jealous
 jeez jerk jill jon jordan ka karma keep kid kind ***know knowww lady lame lan
 d laptop ***last laugh lauren lay le leak least leav leave leavin leaving let
 ***life line link listen listening lol lonely loner long look loop lose loss
 ***love loved lovely low ***luck ***lunch lybrary mad magner mail make man ma
 rry maw mean medium ***meee meet meeting mess message miami midnight miley **
 *miss missed misses mms mom monday month moon morning movie much mum munroe m
 utant nasty need new ***news next ***night ninguem nobodys non noooooo nooooo
 oo nose note notice nuts offer old omg oneal oppress option overheated owner
 paper ***pass past ***peace people peoplethe performance persiankiwi phone ph
 otoshop pick picture pitch place plain play pleaase pm poker ***pool poor pop
 post power prayer pressure ***problem put quizez quot quothello rain read rea
 ding ***ready real reason red regret remember ***reply request ***rest return
 right ***rip ripfarrahfawcett rock room roomie round ***sad sam saving saw sa
 y school screw sedang see seem self sell sense seperate september shea shell
 shit shop shoulder show shut sick sidekick singe sinus skin slack sleep slg s
 mall smell smile snow sobriety society song soo sooo ***sorry sound spanish s
 peak spring stand star start stealth stevens stop ***storm storys strange str
 ong study ***stuff submitted ***suck summer sun sunburn sure sweet swimming t
 ake talk te ***tell test texan ***thank theresa thing think tho thought tiffa
 ney til ***time tingle tip tipsy tnite ***today tom tomorrow tonight toooom *
 *touchy tourist trainer trap trekkie trip trouble try tv tvc tweet tweetpic
 twitt twitter uggghh ***ugh uncle ***update upload ur urbane use verify vote
 wait wanna ***want warm watch water way wayne weather ***week welcome ***well

whole wife win ***wish witch woman wonder wonderful woohoo word ***work would
wrist write ***wrong wtf ***yay yeannot year yep yesterday yike young yrs

Number of Features with minimum importance 0.0004 : 511

A record and summary of 6 types of Feature Selection

Because I WILL be dropping RFE for being too intensive.

Running with 3000 samples gave us vectors with around 5000 dimensions. After much running time, we were able to visually inspect the 500 most important features, as selected by each method. It is perhaps not informative to look at, and compare, these lists of features - but it is fun. And it may give some "subconscious" intuition into that age-old question, "What are these things thinking?" So, before I disable some of these, so I can run bigger datasets through all this code, here's a record of this activity, in the comments/markdown.

I will, quite simple, strip the top two and bottom two rows of each output, and paste them here with clear labels. Pretty sure I won't be running RFE going forward - spins forever, with a pretty low number of samples.

SelectKBest with Bag of Words

account actress add age agree ahhh airport amazing american angel animal anne anniversary
announce answer appreciate arm arrive attend awesome awful aww awww back bad battle beach
beautiful bed bedtime beer believe beloved beyonce birthday blog blonde blue boo book boy
boyfriend brand brave britney bro ... tomorrow tongue toooo tragic trust tt tuesday tweet twilight
twin twit twitter ugh ughhh unc unfair update updates upgrade upload upset ur use vacuum vegas
version voice vote wait walters want warped wat watch way wayne website welcome window wish
woke woman wonderful woop work wrist wrong wtf xd xoxo yay young youth yup

RFE with Bag of Words

aargh account add addict address afraid age agree ahhh air airport album amazing angel
anniversary answer app arm arrive artistic attack attend attention awesome awful awesome aww
baby back bad bag battle bc beach bed bet big birthday bit blib blog blue bonjour boo bore bored
boy boyfriend brain brand break ... tragic true trust tuesday tv tweeting twin twit twitter ugh unc
understand unfair updates upgrade upload upset vacation vegas version vibe video vote wait
walmart wanna want wat watch way wayne welcome whole whyyyy wife win wish woke wonder
wonderful woop word work working world write wrong wtf xavier xd yay young yup

Extra Trees Forest with Bag of Words

able account add addict afraid age agree ahhhh air album alex amazing angel appetite arm art
audrey awesome awful aww baby back bad bag battle bc be beach beautiful bed begin beloved
big bit blaaaaarg blache blog boo bore boy boyfriend break bubba bug burn bus bye call can

cancel cancer car care cat cause ... twitter ugh ughh uncle unfair unusual upgrade upgradeable
upload upset upsetting ur use vacation vibe vote wait wake walk wanna want warm warning watch
way weather wee week weekend welcome well wholee will wish wishing woke woman wonderful
work working world would wrong xoxo yay yeanot year yesterday young yup zoo

SelectKBest with TF-IDF

account actress add adore age agreeahaha ahhh airport amazing american anal angel anne
anniversary announce annoying answer apologize appreciate arm arrive attend awesome awful
awsome aww back background bad bag battle beach beautiful beauty bed bedtime beer beie
believe beloved bestie beyonce birthday ... touchy tragic trust tuesday tweetie twilight twin twit twitt
twitter ugh ughhh unc uncle understand unfair updates upload upset upsetting use va vegas
version view voice vote wait wallpaper want watch way website welcome westney window wish
woke woman wonderful woo work wrong xd xoxo yay yayyy young youth yup zoo

RFE with TF-IDF

able account add address afraid age agree ahhh airport album amazing american angel anne
anniversary answer appreciate arm arrive art attention awesome awful awsome aww back
background bad bag battle bc be beach bed believe big birthday bit blog blue bonjour boo book
bore bout boy boyfriend brand brb break bro ... upsetting va vacation vegas version vibe vote wait
walk wanna want watch way wee welcome whole window wish wishing woke wonder wonderful
word work working world wrong wtf xoxo yay young yud yup zoo

Extra Trees Forest with TF-IDF

able actress add addict afraid age agree ahhhh airport alex amazing andrewcilley angel appetite
arm art awesome awful aww baby back bad battle bc be beach beautiful bed believe beloved big
bit blaaaaarg blache blackberrys blog boo book bore bowls boy boyfriend break brussels bubba
bug bunny burn butterfly ... upgrade upgradeable upload upset upsetting ur use vacation vegas
vibe video wait wake walk walmart wanna want warm warning watch way weather wee week
weekend welcome well whole wholee will window wish wishing woke woman wonderful word work
working world worried would wrong xoxo yay yeanot year yesterday young yud yup yuri zoo

Last but (maybe) not least - Selecting with our Bag-of-N-Grams

Comparing features for Bag-o-words and TF-IDF was very much like comparing apples with apples.

This is going to look a little different, I'll bet, so I'll keep it out from between those two.

```
In [77]: # select the 50 features that have the strongest correlation to a class from the
# original thousands of features
selector_ngrams = SelectKBest(chi2, k=400)
selected_features_ngrams = \
ngrams_selected = selector_ngrams.fit(bv_matrix, df_sm['sentiment']).get_support
selected_features_ngrams
```

```
Out[77]: array([ 65, 66, 67, 68, 212, 245, 527, 562, 592, 656,
1162, 1175, 1178, 1309, 1917, 1974, 2006, 2143, 2158, 2427,
2694, 2695, 2745, 2762, 2778, 2779, 2780, 2791, 2798, 2799,
2812, 2813, 2828, 2846, 2859, 2863, 2871, 2890, 2898, 2946,
2950, 2956, 2994, 2995, 3011, 3021, 3022, 3044, 3045, 3062,
3096, 3113, 3117, 3126, 3141, 3153, 3250, 3253, 3435, 3637,
3644, 3752, 3771, 3778, 3817, 3866, 3891, 3907, 3946, 4047,
4126, 4399, 4601, 4883, 4884, 4929, 5179, 5272, 5475, 5476,
5780, 5794, 5802, 5828, 5947, 6489, 6814, 7214, 7222, 7249,
7253, 7254, 7255, 7256, 7267, 7652, 7656, 7657, 8095, 8096,
8097, 8167, 8269, 8290, 8302, 8312, 8313, 8314, 8320, 8336,
8342, 8348, 8377, 8409, 8444, 8501, 8578, 8941, 8974, 9086,
9635, 9636, 9637, 9701, 9851, 9958, 9959, 9960, 9961, 9969,
10004, 10733, 10798, 10835, 10898, 10911, 10912, 10917, 10918, 10919,
10920, 10921, 10922, 10927, 10928, 10929, 10930, 10931, 10932, 10933,
10934, 10935, 10936, 10937, 10938, 10939, 10940, 10941, 10942, 10943,
10944, 10945, 10946, 10947, 10948, 10949, 10950, 10951, 10952, 10953,
10955, 10956, 10957, 10958, 10959, 10960, 10980, 10981, 10982, 10983,
10993, 10994, 10995, 10996, 11005, 11006, 11007, 11008, 11009, 11010,
11012, 11013, 11014, 11016, 11017, 11018, 11025, 11026, 11027, 11028,
11029, 11030, 11031, 11032, 11033, 11034, 11035, 11036, 11037, 11038,
11039, 11040, 11048, 11049, 11065, 11066, 11067, 11068, 11070, 11079,
11084, 11085, 11090, 11091, 11092, 11093, 11094, 11095, 11107, 11108,
11109, 11110, 11116, 11117, 11118, 11119, 11120, 11133, 11134, 11178,
11182, 11183, 11184, 11185, 11186, 11187, 11188, 11189, 11190, 11191,
11196, 11197, 11198, 11199, 11200, 11201, 11210, 11220, 11221, 11232,
11233, 11234, 11235, 11245, 11246, 11247, 11248, 11249, 11250, 11251,
11252, 11253, 11254, 11255, 11256, 11257, 11258, 11272, 11286, 11295,
11301, 11302, 11303, 11304, 11305, 11314, 11317, 11318, 11319, 11320,
11324, 11325, 11326, 11327, 11332, 11338, 11339, 11340, 11341, 11342,
11343, 11344, 11345, 11353, 11354, 11355, 11356, 11363, 11364, 11376,
11377, 11378, 11379, 11380, 11381, 11382, 11383, 11384, 11390, 11391,
11392, 11393, 11398, 11399, 11400, 11401, 11402, 11403, 11404, 11405,
11406, 11407, 11408, 11409, 11410, 11411, 11412, 11413, 11414, 11415,
11416, 11417, 11418, 11421, 11422, 11423, 11425, 11432, 11433, 11434,
11435, 11443, 11444, 11445, 11446, 11450, 11451, 11452, 11453, 11454,
11455, 11456, 11457, 11458, 11459, 11460, 11461, 11462, 11463, 11464,
11469, 11470, 11471, 11472, 11480, 11481, 11482, 11483, 11484, 11485,
11486, 11487, 11488, 11489, 11498, 11499, 11500, 11501, 11502, 11503,
11504, 11505, 11506, 11507, 11508, 11509, 11510, 11511, 11512, 11513],
dtype=int64)
```

```
In [78]: for x in selected_features_ngrams:
          print(vocab_ngrams[x], end=' - ')
```

afternoon thank - afternoon thank leaving - afternoon thank l
eaving sad - afternoon thank leaving sad theresa - angel sad
- answer find - battle cancer - be be - be ne
ed - bed night - can believe - can wait - can
wait see - charlie angel - day day - day sad -
- day sun - die miss - die think - ed mcMahon -
- family friend - family friend sad - farrah fawcett -
farrah charlie - farrah fawcett - farrah fawcett battle -
- farrah fawcett battle cancer - farrah fawcett cancer - farr
ah fawcett charlie - farrah fawcett charlie angel - farrah fa
wcett find - farrah fawcett find die - farrah fawcett hear -
- farrah fawcett morning - farrah fawcett pass - farrah fawce
tt rest - farrah fawcett sad - farrah fawcett woman -
farrah fawcetts - farrah rest - farrah rip - farrah s
ad - fawcett battle - fawcett battle cancer - fawcett
cancer - fawcett charlie - fawcett charlie angel - fa
wcett find - fawcett find die - fawcett hear - fawcet
t morning - fawcett pass - fawcett rest - fawcett sad
- fawcett today - fawcett woman - find die -

RFE with Bag of N-Grams?

Sorry, no.

RFE takes a fabulously long time, even with a few thousand vectors with a few thousand dimensions.

My bag-o-n-grams has 2-grams, 3-grams, and 4-grams. It's kinda fun. With 3000 samples, it has TENS of THOUSANDS of dimensions. I think RFE, in general, won't play a big role in my plans. I suppose I could run this on a few hundred samples to see if it works. But so what if it does? Can't move ahead with any constructive with dinky little dataframes like that.

Let's do the forest with the ngrams and call it a day.

```
In [79]: model_bagged_ngrams = ExtraTreesClassifier(n_estimators=20)
          model_bagged_ngrams.fit(bv_matrix, df_sm['sentiment'])
          # np.set_printoptions(threshold=np.inf)
          print(model_bagged_ngrams.feature_importances_)
          # np.set_printoptions(threshold=200)
```

```
[0. 0. 0. ... 0. 0. 0.]
```

```
In [80]: i=0
n=0
min_imp = 0.00041
min_hilite = 0.0028
for x in model_bagged_ngrams.feature_importances_:
    if x>min_imp:
        if x>min_hilite:
            print('|||--', end="")
            print(vocab_ngrams[i], end='--||| -      - ')
        else:
            print(vocab_ngrams[i], end=' -      - ')
        n=n+1
    i=i+1
print(' ')
print(' ')
print('Number of Features with minimum importance ', min_imp, ': ', n)
```

```
actress favorite -      - ad sound flag -      - afraid nasty be -      -
|||--ahhhh fall--||| -      - ai yrs leav -      - angel good -      - ang
el sad -      - aroundd good lay -      - awful little -      - |||--aww s
ad--||| -      - aww thank -      - babysitting fair -      - bad lay rain
suck -      - bad oooow stop hurt -      - bad sure -      - bah pass -
- balcony hawaii -      - balcony hawaii miss -      - barrie bad -      -
barrie bad sure come -      - battle cancer -      - bbz farrah -      - b
e be -      - |||--be loner--||| -      - be need -      - bed night -
- |||--blackberrys fritz--||| -      - blew tire -      - bog twitter -
- boo hoo carless -      - |||--boyfriend miss--||| -      - bring catch -
- bunny coupon -      - call come -      - can believe -      - can leave
-      - |||--can wait--||| -      - can wait see -      - candid steph se
e verify get -      - |||--car overheated--||| -      - caro play play -
- |||--cat miss--||| -      - |||--charlie angel--||| -      - cheer day -
- cheer day today -      - cheer day today tomorrow -      - chill arne -
- citizen would -      - class apply -      - close sound -      - clothe
paint -      - coffee deck cat -      - coffee deck cat show -      - colo
r choice -      - color choice green -      - come say -      - could good
-      - could win -      - |||--crap feel--||| -      - crunch comment ug
```

Milestone 1 - Conclusion

Data has been loaded, cleaned, saved, and explored.

Feature Engineering has been carried out using Bag of Words, Bag of N-Grams, and TF-IDF (Term Frequency - Inverse Document Frequency)

Three Feature Selection methods, Select K Best, Recursive Feature Elimination, and Bagged Decision Trees, have been carried out on the engineered features.

Beginning of Milestone 2

Working models.

```
In [88]: from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB

X_train, X_test, y_train, y_test = train_test_split(df_sm['text_nav'], df_sm['sentiment'],
                                                    random_state=42)
count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(X_train)
tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)

clf = MultinomialNB().fit(X_train_tfidf, y_train)
```

```
In [89]: print(clf.predict(count_vect.transform(["AArgh, this is the silliest thing ever"])))

[0]
```

```
In [90]: print(clf.predict(count_vect.transform(["I absolutely love this code."])))

[4]
```

```
In [91]: print(clf.predict(count_vect.transform(["Not sure what I think of this one."])))

[4]
```

```
In [132]: print(clf.predict(count_vect.transform(["Why would you do that?"])))

[0]
```



```

In [109]: from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC

from sklearn.model_selection import cross_val_score

models = [
    RandomForestClassifier(n_estimators=200, max_depth=3, random_state=0),
    LinearSVC(),
    MultinomialNB(),
    LogisticRegression(random_state=0),
]
CV = 5
cv_df = pd.DataFrame(index=range(CV * len(models)))
entries = []
for model in models:
    model_name = model.__class__.__name__
    accuracies = cross_val_score(model, tv_matrix, df_sm['sentiment'], scoring='acc')
    for fold_idx, accuracy in enumerate(accuracies):
        entries.append((model_name, fold_idx, accuracy))
cv_df = pd.DataFrame(entries, columns=['model_name', 'fold_idx', 'accuracy'])

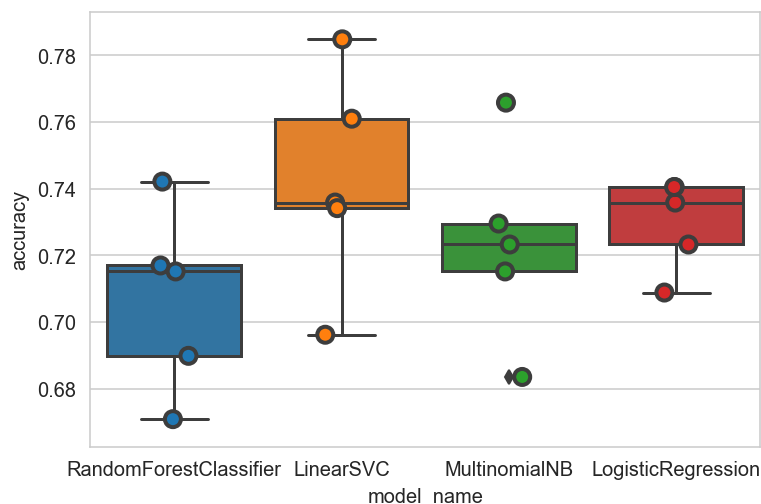
```

```

In [110]: import seaborn as sns

sns.boxplot(x='model_name', y='accuracy', data=cv_df)
sns.stripplot(x='model_name', y='accuracy', data=cv_df,
              size=8, jitter=True, edgecolor="gray", linewidth=2)
plt.show()

```



```
In [111]: cv_df.groupby('model_name').accuracy.mean()
```

```
Out[111]: model_name
LinearSVC          0.74
LogisticRegression 0.73
MultinomialNB      0.72
RandomForestClassifier 0.71
Name: accuracy, dtype: float64
```

```
In [112]: from sklearn.model_selection import train_test_split

model = LinearSVC()

X_train, X_test, y_train, y_test, indices_train, indices_test = train_test_split
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

Confusion Matrix

Calculate, and display, true positives, true negatives, false positives, and false negatives.

We'll show both the raw matrix, and a "Heatmap", which looks cool, but doesn't give too much insight with only two labels.

```
In [126]: from sklearn.metrics import confusion_matrix

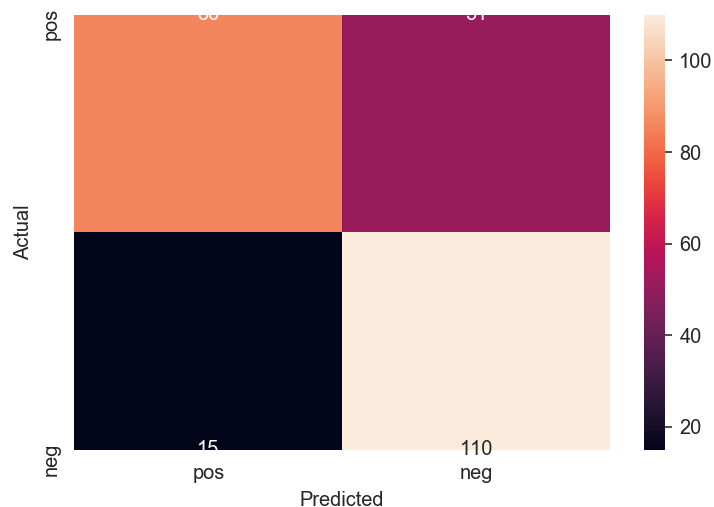
conf_mat = confusion_matrix(y_test, y_pred)
print('RAW CONFUSION MATRIX')
print(conf_mat)
print('Because the heatmap may LOOK cool, but doesn\'t tell us much, with only two labels')
print('-----')
print('Heatmap of Confusion Matrix:')
fig, ax = plt.subplots(figsize=(6,4))
sns.heatmap(conf_mat, annot=True, fmt='d',
            xticklabels=['pos', 'neg'], yticklabels=['pos', 'neg'])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

RAW CONFUSION MATRIX

```
[[ 86  51]
 [ 15 110]]
```

Because the heatmap may LOOK cool, but doesn't tell us much, with only two labels

Heatmap of Confusion Matrix:



```
In [127]: model.fit(tv_matrix, df_sm['sentiment'])
```

```
Out[127]: LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
                    intercept_scaling=1, loss='squared_hinge', max_iter=1000,
                    multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
                    verbose=0)
```

Precision, Recall and f1-Score

```
In [130]: from sklearn import metrics
print(metrics.classification_report(y_test, y_pred,
                                   target_names=['0', '4']))
```

	precision	recall	f1-score	support
0	0.85	0.63	0.72	137
4	0.68	0.88	0.77	125
accuracy			0.75	262
macro avg	0.77	0.75	0.75	262
weighted avg	0.77	0.75	0.74	262

Interesting points of comparison!!

Early in this project, I was doing sentiment predictions with Afinn, to be used as a baseline later. At our first arrival at this point, things are looking better already!

Weighted Avg F1-Scores

0.63 - Afinn original values.

0.74 - 800 rows, Tf-idf encoding, NO feature selection, Linear SVC

=====

Ensembling

=====

Ensemble methods are meta-algorithms that combine several machine learning techniques into one predictive model in order to decrease variance (bagging), bias (boosting), or improve predictions (stacking). Ensemble methods can be divided into two groups: sequential ensemble methods where the base learners are generated sequentially (e.g. AdaBoost) and parallel ensemble methods where the base learners are generated in parallel (e.g. Random Forest). The basic motivation of sequential methods is to exploit the dependence between the base learners since the overall performance can be boosted by weighing previously mislabeled examples with higher weight. The basic motivation of parallel methods is to exploit independence between the base learners since the error can be reduced dramatically by averaging.

Most ensemble methods use a single base learning algorithm to produce homogeneous base learners, i.e. learners of the same type leading to homogeneous ensembles. There are also some methods that use heterogeneous learners, i.e. learners of different types, leading to

heterogeneous ensembles. In order for ensemble methods to be more accurate than any of its individual members the base learners have to be as accurate as possible and as diverse as possible.

Bagging

Bagging stands for **bootstrap aggregation**. One way to reduce the variance of an estimate is to average together multiple estimates. For example, we can train M different trees f_m on different subsets of the data (chosen randomly with replacement) and compute the ensemble:

$$f(x) = \frac{1}{M} \sum_{m=1}^M f_m(x)$$

Translation: The final result is the average of the M sub-results.

Bagging uses bootstrap sampling to obtain the data subsets for training the base learners. For aggregating the outputs of base learners, bagging uses voting for classification and averaging for regression.

```
In [133]: %matplotlib inline

import itertools
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

from sklearn import datasets

from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import BaggingClassifier
from sklearn.model_selection import cross_val_score, train_test_split

from mlxtend.plotting import plot_learning_curves
from mlxtend.plotting import plot_decision_regions

np.random.seed(0)
```

```
In [134]: # X, y = iris.data[:, 0:2], iris.target

X = tv_matrix
y = np.array(df_sm['sentiment'])

clf1 = DecisionTreeClassifier(criterion='entropy', max_depth=1)
clf2 = KNeighborsClassifier(n_neighbors=1)

bagging1 = BaggingClassifier(base_estimator=clf1, n_estimators=10, max_samples=0)
bagging2 = BaggingClassifier(base_estimator=clf2, n_estimators=10, max_samples=0)
```

```

In [145]: from sklearn.decomposition import PCA

label = ['Decision Tree', 'K-NN', 'Bagging Tree', 'Bagging K-NN']
clf_list = [clf1, clf2, bagging1, bagging2]

fig = plt.figure(figsize=(10, 8))
gs = gridspec.GridSpec(2, 2)
grid = itertools.product([0,1],repeat=2)

for clf, label, grd in zip(clf_list, label, grid):
    scores = cross_val_score(clf, X, y, cv=3, scoring='accuracy')

    print("Accuracy: %.2f (+/- %.2f) [%s]" %(scores.mean(), scores.std(), label))

    #clf.fit(X, y)

    pca = PCA(n_components = 2)
    X_flattened = pca.fit_transform(X)
    #clf.fit(X_flattened, y)
    #clf.fit(X, y)
    clf.fit(X_flattened, y)
    ax = plt.subplot(gs[grd[0], grd[1]])
    fig = plot_decision_regions(X=X_flattened, y=y, clf=clf, legend=2)
    plt.title(label)

    #plot_decision_regions(X_train2, y_train, clf=clf, legend=2)

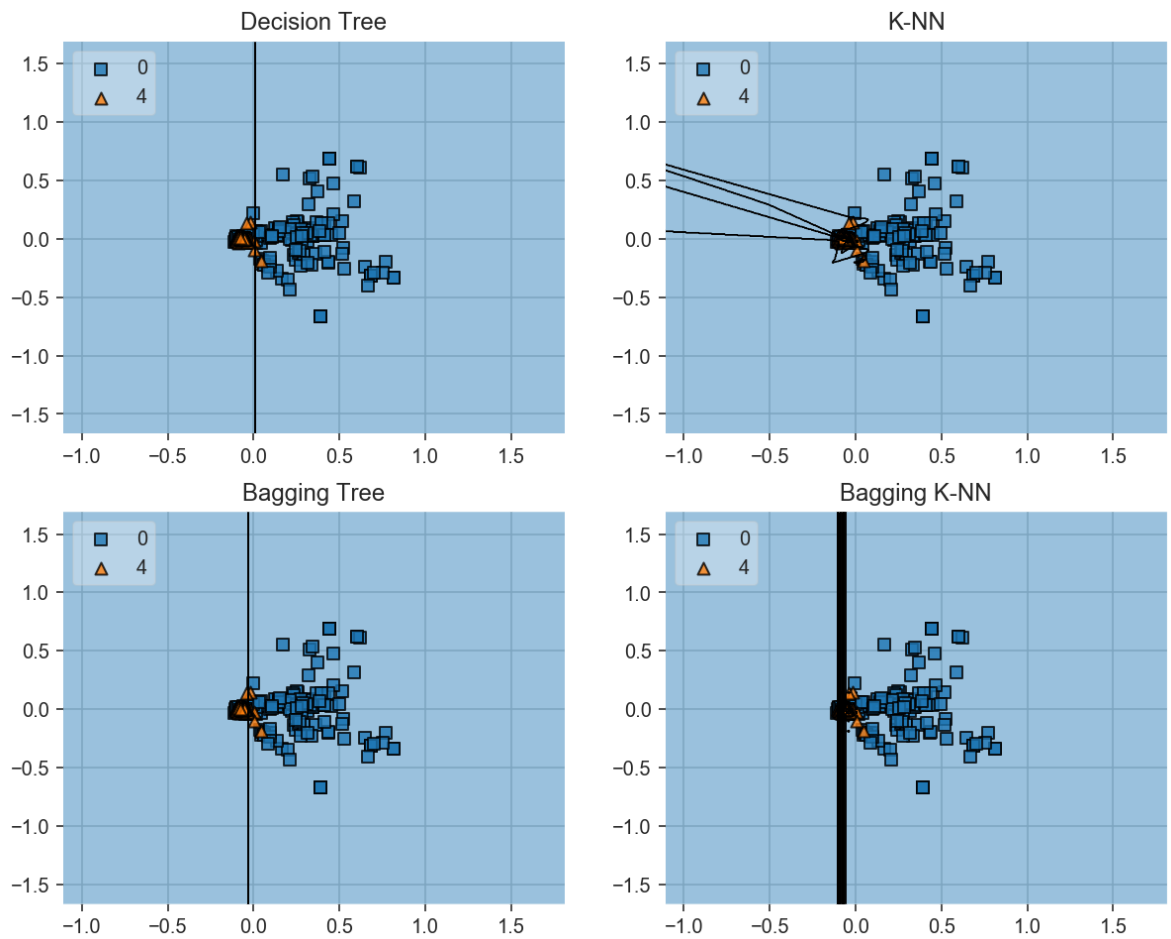
plt.show()

```

```

Accuracy: 0.62 (+/- 0.01) [Decision Tree]
Accuracy: 0.65 (+/- 0.03) [K-NN]
Accuracy: 0.62 (+/- 0.01) [Bagging Tree]
Accuracy: 0.60 (+/- 0.04) [Bagging K-NN]

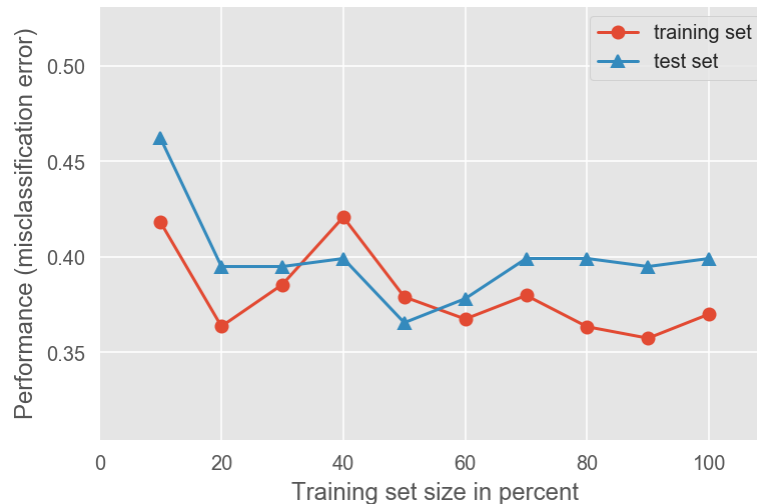
```



The figure above shows the decision boundary of a decision tree and k-NN classifiers along with their bagging ensembles applied to the Iris dataset. The decision tree shows axes parallel boundaries while the $k = 1$ nearest neighbors fits closely to the data points. The bagging ensembles were trained using 10 base estimators with 0.8 subsampling of training data and 0.8 subsampling of features. The decision tree bagging ensemble achieved higher accuracy in comparison to k-NN bagging ensemble because k-NN are less sensitive to perturbation on training samples and therefore they are called *stable learners*. Combining stable learners is less advantageous since the ensemble will not help improve generalization performance.


```
In [146]: #plot learning curves
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_

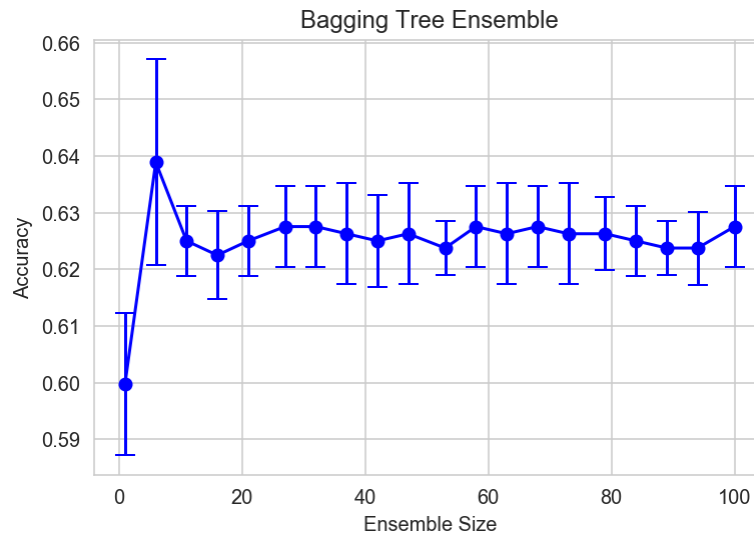
plt.figure()
plot_learning_curves(X_train, y_train, X_test, y_test, bagging1, print_model=False)
plt.show()
```



The figure above shows learning curves for the bagging tree ensemble. We can see an average error of 0.35 on the training data and a U-shaped error curve for the testing data. The smallest gap between training and test errors occurs at around 80% of the training set size.

```
In [147]: #Ensemble Size
# num_est = map(int, np.linspace(1,100,20))
num_est = np.linspace(1,100,20).astype(int)
bg_clf_cv_mean = []
bg_clf_cv_std = []
for n_est in num_est:
    bg_clf = BaggingClassifier(base_estimator=clf1, n_estimators=n_est, max_samp
    scores = cross_val_score(bg_clf, X, y, cv=3, scoring='accuracy')
    bg_clf_cv_mean.append(scores.mean())
    bg_clf_cv_std.append(scores.std())
```

```
In [148]: plt.figure()
(, caps, _) = plt.errorbar(num_est, bg_clf_cv_mean, yerr=bg_clf_cv_std, c='blue',
# caps = plt.errorbar(num_est, bg_clf_cv_mean, yerr=bg_clf_cv_std, c='blue', fmt=
for cap in caps:
    cap.set_markeredgewidth(1)
plt.ylabel('Accuracy'); plt.xlabel('Ensemble Size'); plt.title('Bagging Tree Ensemble')
plt.show()
```



The figure above shows how the test accuracy improves with the size of the ensemble. Based on cross-validation results, we can see the accuracy increases until approximately 10 base estimators and then plateaus afterwards. Thus, adding base estimators beyond 10 only increases computational complexity without accuracy gains for the Sentiment140 dataset.

Boosting

Boosting refers to a family of algorithms that are able to convert weak learners to strong learners. The main principle of boosting is to fit a sequence of weak learners (models that are only slightly better than random guessing, such as small decision trees) to weighted versions of the data,

where more weight is given to examples that were mis-classified by earlier rounds. The predictions are then combined through a weighted majority vote (classification) or a weighted sum (regression) to produce the final prediction. The principal difference between boosting and the committee methods such as bagging is that base learners are trained in sequence on a weighted version of the data.

```
In [149]: import itertools
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

from sklearn import datasets

from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import cross_val_score, train_test_split

from mlxtend.plotting import plot_learning_curves
from mlxtend.plotting import plot_decision_regions
```

```
In [168]: X = tv_matrix
y = np.array(df_sm['sentiment'])

clf = DecisionTreeClassifier(criterion='entropy', max_depth=1)

num_est = [1, 2, 3, 10]
label = ['AdaBoost (n_est=1)', 'AdaBoost (n_est=2)', 'AdaBoost (n_est=3)', 'AdaBoost (n_est=10)']
```

```
In [169]: len(X)
```

```
Out[169]: 792
```

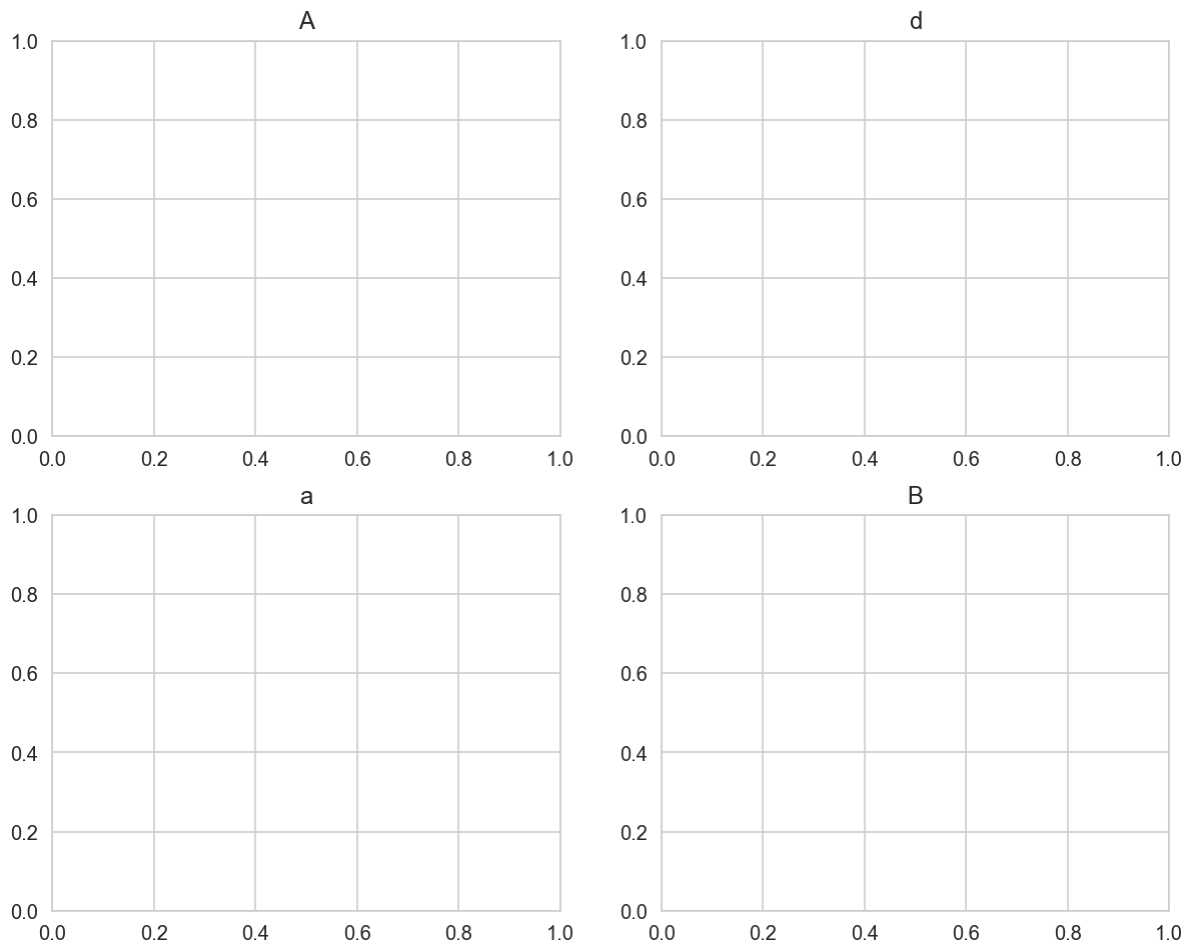
```

In [171]: fig = plt.figure(figsize=(10, 8))
gs = gridspec.GridSpec(2, 2)
grid = itertools.product([0,1],repeat=2)

for n_est, label, grd in zip(num_est, label, grid):
    boosting = AdaBoostClassifier(base_estimator=clf, n_estimators=n_est)
    #pca = PCA(n_components = 2)
    #X_flattened = pca.fit_transform(X)
    boosting.fit(X_flattened, y)
    ax = plt.subplot(gs[grd[0], grd[1]])
    #fig = plot_decision_regions(X=X, y=y, clf=boosting, Legend=2)
    plt.title(label)

plt.show()

```

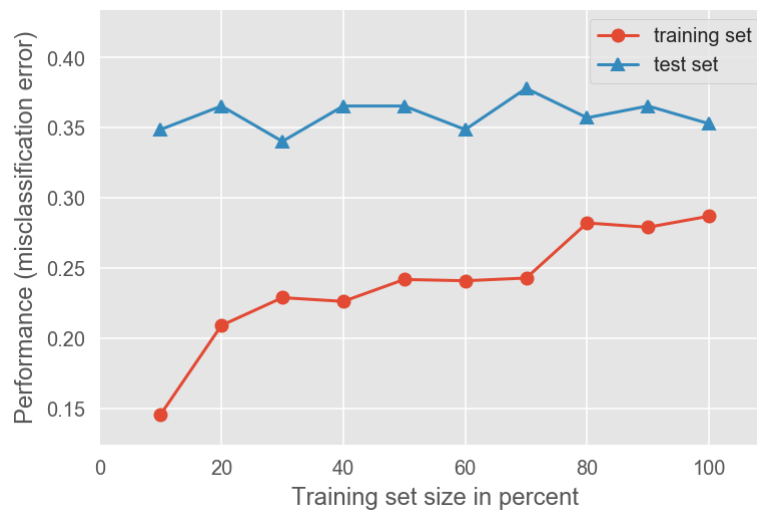


The AdaBoost algorithm is illustrated in the figure above. Each base learner consists of a decision tree with depth 1, thus classifying the data based on a feature threshold that partitions the space into two regions separated by a linear decision surface that is parallel to one of the axes.

```
In [172]: #plot Learning curves
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

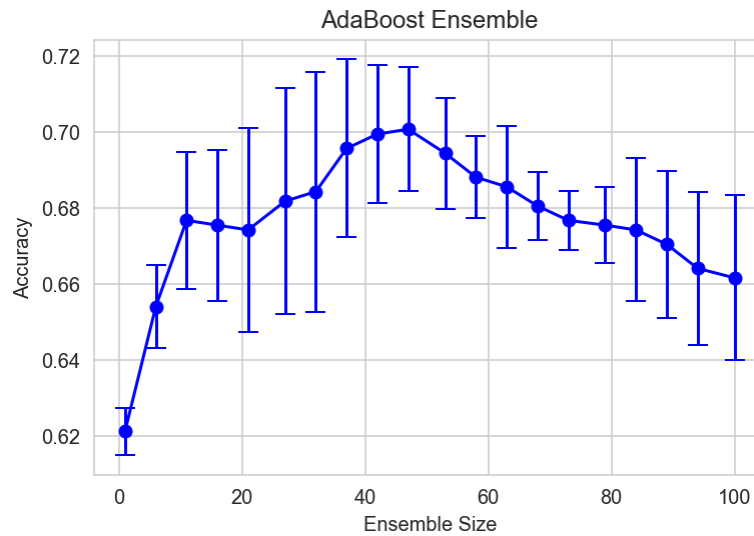
boosting = AdaBoostClassifier(base_estimator=clf, n_estimators=100)

plt.figure()
plot_learning_curves(X_train, y_train, X_test, y_test, boosting, print_model=False)
plt.show()
```



```
In [173]: #Ensemble Size
#num_est = map(int, np.linspace(1,100,20))
num_est = np.linspace(1,100,20).astype(int)
bg_clf_cv_mean = []
bg_clf_cv_std = []
for n_est in num_est:
    ada_clf = AdaBoostClassifier(base_estimator=clf, n_estimators=n_est)
    scores = cross_val_score(ada_clf, X, y, cv=3, scoring='accuracy')
    bg_clf_cv_mean.append(scores.mean())
    bg_clf_cv_std.append(scores.std())
```

```
In [174]: plt.figure()
(, caps, _) = plt.errorbar(num_est, bg_clf_cv_mean, yerr=bg_clf_cv_std, c='blue')
for cap in caps:
    cap.set_mmarkeredgewidth(1)
plt.ylabel('Accuracy'); plt.xlabel('Ensemble Size'); plt.title('AdaBoost Ensemble')
plt.show()
```



Stacking

Stacking is an ensemble learning technique that combines multiple classification or regression models via a meta-classifier or a meta-regressor. The base level models are trained based on complete training set then the meta-model is trained on the outputs of base level model as features. The base level often consists of different learning algorithms and therefore stacking ensembles are often heterogeneous.

```
In [175]: import itertools
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec

from sklearn import datasets

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from mlxtend.classifier import StackingClassifier

from sklearn.model_selection import cross_val_score, train_test_split

from mlxtend.plotting import plot_learning_curves
from mlxtend.plotting import plot_decision_regions
```

```
In [176]: X = tv_matrix
y = np.array(df_sm['sentiment'])

clf1 = KNeighborsClassifier(n_neighbors=1)
clf2 = RandomForestClassifier(random_state=1)
clf3 = GaussianNB()
lr = LogisticRegression()
sclf = StackingClassifier(classifiers=[clf1, clf2, clf3],
                          meta_classifier=lr)
```

```

In [178]: label = ['KNN', 'Random Forest', 'Naive Bayes', 'Stacking Classifier']
          clf_list = [clf1, clf2, clf3, sclf]

          fig = plt.figure(figsize=(10,8))
          gs = gridspec.GridSpec(2, 2)
          grid = itertools.product([0,1],repeat=2)

          clf_cv_mean = []
          clf_cv_std = []
          for clf, label, grd in zip(clf_list, label, grid):

              pca = PCA(n_components = 2)
              X_flattened = pca.fit_transform(X)

              scores = cross_val_score(clf, X_flattened, y, cv=3, scoring='accuracy')
              print("Accuracy: %.2f (+/- %.2f) [%s]" %(scores.mean(), scores.std(), label))
              clf_cv_mean.append(scores.mean())
              clf_cv_std.append(scores.std())

              clf.fit(X_flattened, y)
              ax = plt.subplot(gs[grd[0], grd[1]])
              fig = plot_decision_regions(X=X_flattened, y=y, clf=clf)
              plt.title(label)

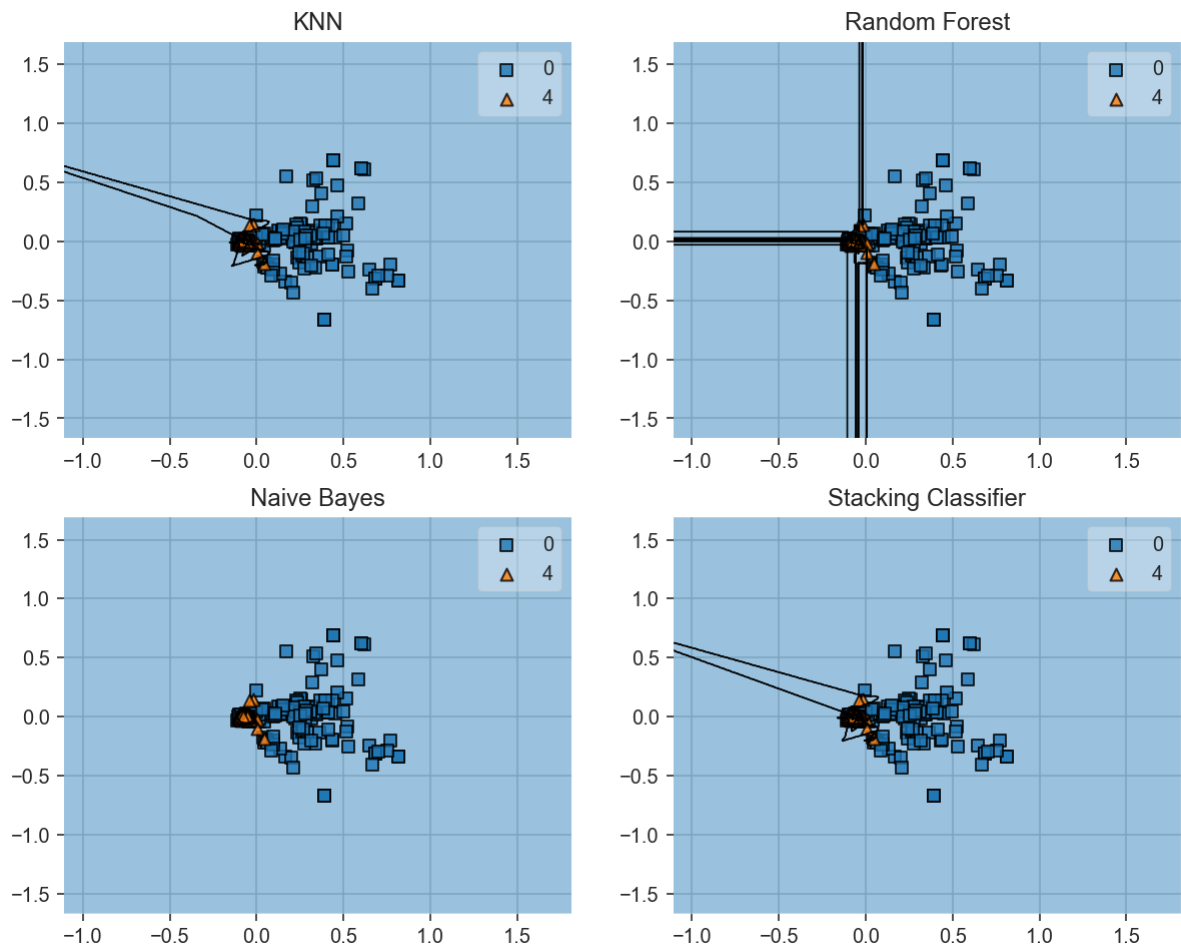
          plt.show()

```

```

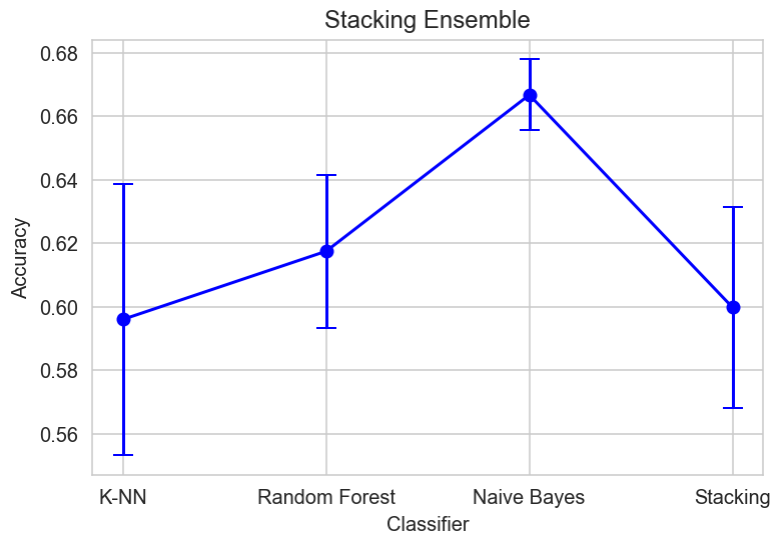
Accuracy: 0.60 (+/- 0.04) [KNN]
Accuracy: 0.62 (+/- 0.02) [Random Forest]
Accuracy: 0.67 (+/- 0.01) [Naive Bayes]
Accuracy: 0.60 (+/- 0.03) [Stacking Classifier]

```

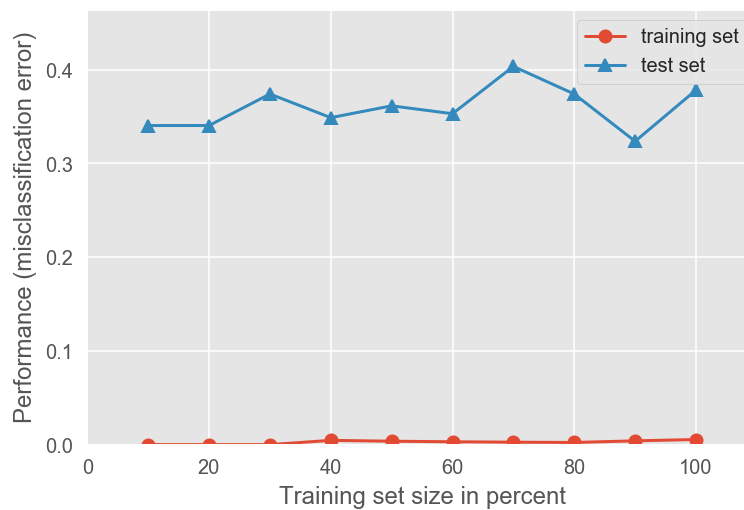
The stacking ensemble is illustrated in the figure above. It consists of k-NN, Random Forest and Naive Bayes base classifiers whose predictions are combined by Logistic Regression as a meta-classifier. We can see the blending of decision boundaries achieved by the stacking classifier.

```
In [181]: #plot classifier accuracy
plt.figure()
(_, caps, _) = plt.errorbar(range(4), clf_cv_mean, yerr=clf_cv_std, c='blue', fm
for cap in caps:
    cap.set_markeredgewidth(1)
plt.xticks(range(4), ['K-NN', 'Random Forest', 'Naive Bayes', 'Stacking'])
plt.ylabel('Accuracy'); plt.xlabel('Classifier'); plt.title('Stacking Ensemble')
plt.show()
```



```
In [182]: #plot learning curves
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_

plt.figure()
plot_learning_curves(X_train, y_train, X_test, y_test, scf, print_model=False, s
plt.show()
```



```
###
```

End of working working file.

```
###
```

Sentiment Analysis with AFINN

As a quick and dirty sanity check, I've set up AFINN in the early stages of data cleaning, and intend to keep a little record of AFINN's performance, as I increase the rigour of the data cleaning.

```
In [81]: from afinn import Afinn

afn = Afinn(emoticons=True)
```

```
In [82]: texts = np.array(df_sm['text_nav'])
sentiments = np.array(df_sm['sentiment'])

# extract data for model evaluation
#train_texts = texts[:10000]
#train_sentiments = sentiments[:10000]

#test_texts = texts[40000:60000]
#test_sentiments = sentiments[40000:60000]
sample_ids = [626, 533, 310, 123, 654, 400]
```

```
In [83]: #for text_clean, sentiment in zip(texts[sample_ids], sentiments[sample_ids]):
#         print('TEXT:', texts)
#         print('Actual Sentiment:', sentiment)
#         print('Predicted Sentiment polarity:', afn.score(texts))
#         print('-'*60)
```

```
In [84]: # Predict sentiment with Afinn

#sentiment_polarity = [afn.score(Text) for Text in normalized_texts]
#predicted_sentiments = ['positive' if score >= 1.0 else 'negative' for score in sentiment_polarity]
#predicted_sentiments = [4 if score >= 1.0 else 0 for score in sentiment_polarity]
```

```
In [85]: #meu.display_model_performance_metrics(true_labels=test_texts, predicted_labels=predicted_sentiments,
#                                               classes=['positive', 'negative'])
#meu.display_model_performance_metrics(true_labels=test_sentiments, predicted_labels=predicted_sentiments,
#                                       classes=[4, 0])
```

Checking cleaning with AFINN

I'm curious about how deeper cleaning affects predictive models. So I set up Afinn after the very first round of data cleaning, and am going to track results here in the markdown. For simplicity, I will monitor the effects of different levels of cleaning on "weighted avg f1-score"

Round 1, most basic cleaning, 20000 rows: 0.63

Round 2, include normalization, 20000 rows: 0.63

In [86]: `df_sm.head()`

Out[86]:

	sentiment	ID	Time	none	username	
799599	0	2329056794	Thu Jun 25 10:17:56 PDT 2009	NO_QUERY	enge10	GUYS the ghost is back at room
799600	0	2329056832	Thu Jun 25 10:17:56 PDT 2009	NO_QUERY	alysamarsiella	RIP Farrah what a sham
799601	0	2329056954	Thu Jun 25 10:17:57 PDT 2009	NO_QUERY	velobabe	http://www.cnn.com/2009/SHC
799602	0	2329057090	Thu Jun 25 10:17:58 PDT 2009	NO_QUERY	XoAngelJenn36oX	it was so nice and sunny this
799603	0	2329057145	Thu Jun 25 10:17:58 PDT 2009	NO_QUERY	OliviaDAngelo	

Save to database

In [87]: `#df.to_sql('df_sm', con)`

In []:

In []: