2019-1002 IST 736 Text Mining

Homework Assignment 3 (week 3)

Ryan Timbrook
NetID: RTIMBROO
Course: IST 736 Text Mining
Term: Fall, 2019
Topic: Text Vectorization on NFL Team News

Homework Assignment 3 (week 3)

Table of Contents

1 Introduction

The NFL was already a lucrative business before Fantasy football took off, now you can't turn on a news program without hearing about Fantascy stats and which provider has better predictive modeling and real-time feedback on player's performance. According to Gallup, Football still overwhelmingly dominates American enthusiasm, with 37% calling it their favorite sport, compared to 11% for basketball, it's the closest rival; Baseball is at 9%, after decades of declining stature. NFL revenue grew an estimated $900 million to $14 billion in 2017, in 2018 it generated about $15 billion. The league announced in January that it's aiming to boost it's annual revenue to $25 billion by 2027. Fantasy football and the spread of legalized sports betting across the U.S. promises to lock in fans and keep them focused on the game.

Data Science aims to be at the heart of how fantasy players and those gambling on teams make their choices. IBM announced this 2019 season, "Fantasy Insights with Watson". "ESPN Fantasy Insights draws upon the latest in machine learning techniques to turn unstructured data into valuable insights. Nearly 10 million players rely on the combined resources of Watson Discovery and Watson OpenScale running on the IBM Cloud to give them a competitive edge."
- *https://ibm.com/fantasy*

In this research the aim is to use text mining techniques on public news and media web sites to aggregate data relating to NFL teams and it's players. For this initial problem, the goal is to vectorize the text data and explore the vectors for interesting patterns that could provide insights into possible next steps such as sentiment analysis and classification.

1.1 Purpose

Identify public sentiment toward NFL teams and it's players that could help fans choose teams and players to play in their fantasy leagues.

1.2 Scope

Create an initial dataset from text mining an NFL media web site to be used as a baseline for sentiment analysis on teams and players.

- Data on NFL teams and their rosters web scrapped from a single source.
- Perform vectorization pre-processing steps on NFL teams text scrapped from web site
- Perform vectorization EDA to identify interesting patterns in the data.

2 Analysis and Models

2.1 About the Data

Teams and Players data was mined from *https://www.lineups.com* website. Focusing on Teams for this research, the Team data object created has the following characteristics:

- Team name
- Team player roster year
- A collection of active players that make up its roster
- Team roster player summary statistics
- Team text to be used as baseline vectorization text
    - Data scrapped from subsite specific to each team's active players roster. It represents this specific website commentary on each teams prior year performance and outlook for the new season. It poses team questions and opinions on what the teams should do if they want to have a winning season.
    - All text scrapped from this site was cleaned of any HTML tags before storing it for use in the following modeling steps.
    - The text scrapped was stored in this object under a hierarchy as:
        - source='lineups.com'
            - text_topic='team_roster_news'
                - datetime stamp -> this represents the point in time when the data was collected
                    - text data scrapped
- Player text to be used for follow on analysis focusing on players

Note that this dataset is just a small sampling, it is not a good representation of the overall public and media sentiment on the NFL Teams. More robust data mining will be conducted for follow-on analysis.

2.1.1 *Dataset Info*

A collection of 32 NFL teams media text data scrapped from the above-mentioned website. Each team's text data is its own document and the collection of these team documents makes up the overall vocabulary corpus to be analyzed for this study.

The overall data set collection memory size is: **248 KB**.

2.1.2 *Data Exploration & Cleaning*

The following cleaning and transformation techniques were performed programmatically in python using a jupyter notebook for code execution and visualization. The python version used was Anaconda 3.6.

Focusing on the goal of vectorizing team text as individual documents to be used as a corpus for analyzing media and public sentiment toward NFL teams, the following text preparation pipeline steps were taken:

- Load the text for each NFL team as an individual document
- Tokenize each document
    - The **nltk word_tokenize** class was used for this step.
- Perform Bag Of Words analysis
- Clean each documents tokens:
    - Remove punctuation

- Remove non-alphabetic
- Lower case text
- Remove stop words

This section will cover the cleaning steps leaving the vectorization steps to be discussed in the Modeling section below.

2.1.2.1 Cleaning Steps Taken:

2.1.2.1.1 Initial Tokenization Team Text

Using the nltk word_tokenize class, each line the individual team's text document was tokenized. These figures represent pre-cleaning numbers.

Table 2.1: Team text total token count sample and descriptive stats:

team	tokens	total_tokens
denver_broncos	[As, mentioned, before, ,, Joe, Flacco, is, a,...	1647
new_england_patriots	[If, LaCosse, manages, to, produce, and, play,...	1545
los_angeles_rams	[If, Gurley, canâ€™t, replicate, his, producti...	1539
dallas_cowboys	[While, Dak, and, Zeke, are, currently, gettin...	1518
carolina_panthers	[Although, much, of, the, attention, is, on, t...	1505
miami_dolphins	[While, thereâ€™s, going, to, be, some, talent...	1491
cleveland_browns	[The, Browns, offensive, line, has, undergone,...	1487
detroit_lions	[If, the, Lions, want, to, reach, the, playoff...	1482
jacksonville_jaguars	[Foles, will, immediately, bring, in, a, winni...	1481
los_angeles_chargers	[Los, Angeles, doesnâ€™t, have, any, commitmen...	1442

Table 2.2: Overall Team Text Corpus Feature Count - Top 20

feature	feature_count
the	1124
,	977
.	798
to	580
a	536
and	425
in	379
of	366
be	251
for	218
that	191
is	188
year	165
will	160
he	149
as	148
season	144
their	143
his	134
with	131

Figure 2.1: Sampling of Team Text Bag Of Words Word Clouds

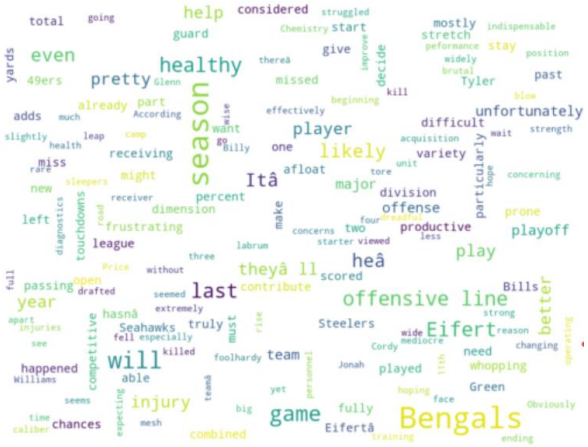
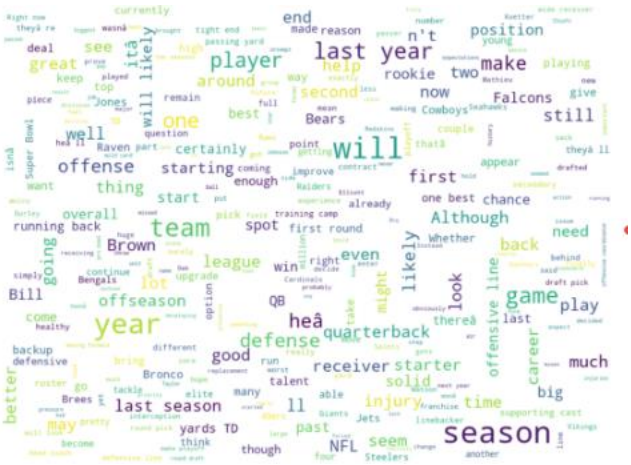


Figure 2.2: Overall Team Text Corpus Bag Of Words Word Cloud



- Total Tokens Prior to Vectorization Cleaning:
  21237

2.1.2.1.2 Vectorization Preprocessing Steps

For each text document, the following pre-processing vectorization steps were taken:

(note - each of the bellow steps is controlled via a Boolean True or False conditional statement that allowed the testing of each of these steps independently as well as in combination to evaluate optimal vectorization preparation)

- All hashtag tokens were removed
- All URL tokens were removed
- Punctuation was removed using the python string. punctuation values
  i. '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
- Non-Alphabetic tokens were removed using the python string method isalpha()
- Lowercase all of the token characters
- Stop words were removed using the NLTK English stopwords list
  i. additionally, this step allows the addition of custom stop words to be added to the list for fine-tuning.
- Stemming was performed on one trial to evaluate any vocabulary reduction and to capture the new dataset for future evaluation trials.
  i. for reference, see 'kept_lower_stem_feature_counts.csv'

- A kept feature word count document was generated and stored for later evaluation as:
    i. ./data/kept_features.csv
- Integer feature vector mappings were created for wrapped retrieval of features by an indexed id.

Post-pre-processing: Each cleaned text was saved to its own file to be used as a corpus of documents in the vectorization process. The file name is the team name concatenated to a string representing the corpus category (_nfl_team_text.txt). In this case it looks like '{team_name}_ nfl_team_text.txt ' and is loaded to a relative directory path as './corpus/teams/v2/cleaned/{team_name}_ nfl_team_text.txt.

Total Feature Count Prior to Vectorization Preprocessing: **21237**

Table 2.3: Vocabulary Size Reduction Comparisons (small sample)

After Preprocessing Feature Count	Remove Stop Word	Remove Punctuation	Remove Non-Alpha	Lowercase	Stemming
9879	True	True	True	True	False
9879	True	True	True	True	True
10460	True	True	True	False	False
18647	True	True	False	False	False
18338	False	False	True	False	True

As shown above in Table 2.3, each of the cleaning steps affects the total feature set vocabulary in varying ways. Ideally, given the time, it's recommended to save a new file after each transformation to evaluate the linguistical impacts it has on the overall quality of the new dataset in performing sentiment analysis on the tweet texts.

One example of needing to run many scenarios to pick the best preprocessing techniques for this task is the lowercasing of the dataset. Often people use capitalization to convey a tone and or strength to a statement they are making through text. By removing these human expressions of sentiment we're possibly missing out on valuable insights and could skew the data and it's meaning.

For our purposes of this trial the first configuration set shown above in Table 2.3 was used for the Vectorization modeling that will be described in section 3.

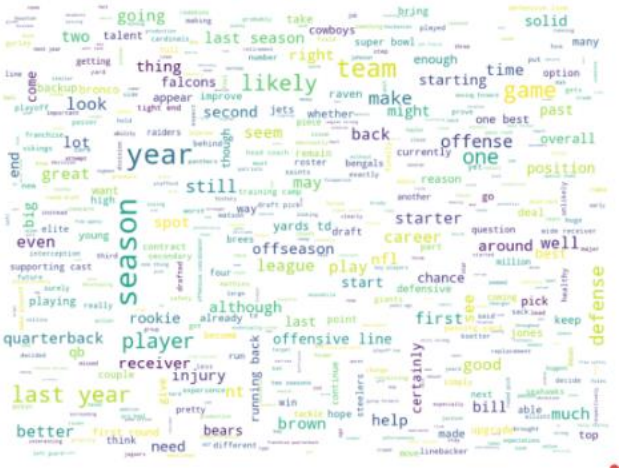
Figure 2.4: Top 10 Cleaned Feature Counts

feature	feature_count
year	167
season	147
last	116
one	84
team	79
likely	63
make	51
first	51
still	50
back	44
get	43
two	41

- Figure 2.3 is an output of the top 10 feature frequencies after vectorization preprocessing.

Image 2.3: Word Cloud of full corpus cleaned

- Image 2.3 is a Word Cloud representation of the same dataset



3 Models

3.1.1 *CountVectorizer - Vectorize Team Text Documents*

Utilizing the python package **sklearn.feature_extraction.text CountVectorizer** class, this model converts a collection of team text documents to a matrix of token counts. This implementation of CountVectorizer produces a sparse representation of the counts using scipy.sparse.coo_matrix.

In-text mining, it is important to create the document-term matrix (DTM) of the corpus we are interested in. A DTM is basically a matrix, with documents designated by rows and words by columns, that the elements are the counts or the weights (usually by tf-idf). The subsequent analysis is usually based creatively on DTM.

CountVectorizer supports counts of N-grams of words or consecutive characters. Once fitted, the vectorizer has built a dictionary of feature indices:
The index value of a word in the vocabulary is linked to its frequency in the whole training corpus.

The data for these vectorization steps is retrieved by reading .txt files from a local file directory created during the prior cleaning process steps. Where ever input='filename' for vectorization parameter this collection of files is used.

- path=f'{corpusDir}/teams/v2/cleaned/
- A collection of 32 documents are retrieved containing media text specific to each team's performance.

For this experiment unigram, bigram and trigram models were vectorized and saved to file for analysis and future additive modeling.

You can think of an N-gram as the sequence of N words, by that notion, a 2-gram (or bigram) is a two-word sequence of words like “please turn”, “turn your”, or ”your homework”, and a 3-gram (or trigram) is a three-word sequence of words like “please turn your”, or “turn your homework”. N-grams are used in building predictive language models based on models learned word sequencing.

3.1.1.1 CountVectorizer Unigram Model

Unigrams are individual words in sequence from a sentence or document collection. This is the most common vectorization approach

CountVectorizer has many parameters that influence the feature word output and ultimately the overall strength of the vocabulary being processed. Here are a few for reference future consideration: for a complete list, follow this link to the sklearn tutorial site.

3.1.1.1.1 CountVectorizer Unigram Details

CountVectorizer Unigram Parameters:

- input='filename'
- ngram_range=(1,1)
- stop_words='english'
- max_features=None
- max_df=1.0
- min_df=1
- analyzer=word

Homework Assignment 3 (week 3)

Both fit and transform methods were performed on this model.

3.1.1.1.2 CountVectorizer Unigram Results

Unigram Document Term Matrix Dimensions

Size	Rows	Columns(vocabulary)	DataType
4963	32	2113	scipy.sparse.csr.csr_matrix

: CountVector Unigram Spars Matrix Snippet

	aaron	abdullah	ability	able	abry	absence	absent	absolute	absolutely	accompanied	...	young	younger	youngster	youngsters	zay	zeitler
0	0	0	0	2	0	0	0	0	0	0	...	5	0	0	0	0	0
5	0	0	0	1	0	0	0	1	0	0	...	0	0	0	0	0	0
10	2	0	1	0	0	0	0	0	0	0	...	0	0	0	0	0	0
15	0	0	0	1	0	0	0	0	1	0	...	2	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0

Below is a sampling of the Unigram output file: Its format is one line per NFL Team. The first token in the line is the team name/text filename vectorization was performed on. It is then followed by the term and term frequency for each of the kept features. The full output vector file can be found in the supplemental zip file under ./output/count_unigram_feature_vector_tf.txt

: CountVector Unigram Feature Vector Output Example - single line for each of the NFL Teams -

new_york_jets_nfl_team_text.txt able 2 adds 1 afc 4 agency 1 air 1 allows 1 anderson 1 ball 2 bell 3 berth 2 best 2 better 2 big 1 biggest 1 bowl 2 breakout 2 buster 1 campaign 2 card 2 centerpieces 1 certainly 4 cj 1 comes 3 contract 1 core 1 crowder 1 darnold 3 dead 2 defense 1 direction 2 earn 2 east 4 ended 1 england 2 enunwa 1 especially 1 face 1 favorites 2 finish 2 finishing 4 fit 1 form 2 free 1 fruition 2 gives 1 going 3 good 1 ground 1 guaranteed 1 happened 2 huge 2 hurting 1 jets 16 league 1 leap 2 leveon 1 like 2 likely 4 linebacker 1 losing 1 lot 1 main 1 make 2 mean 2 middle 1 million 1 money 1 mosely 2 near 2 needs 1 new 3 nt 1 numbers 1 offense 1 offer 2 offering 1 offseason 1 options 1 patriots 2 picked 4 pieces 1 players 2 playoffs 2 point 1 poses 1 postseason 2 predicted 2 presence 1 pressure 1 putting 1 qualified 2 ravens 1 receiver 3 relationship 1 running 1 said 2 season 3 seasons 1 second 4 sides 3 sign 1 signed 1 signing 1 similar 1 simply 2 skrine 1 slated 2 slightly 2 spending 1 spot 2 starters 1 steelers 2 step 2 stranger 2 suiting 1 super 2 supposed 1 taken 1 takes 2 talent 1 team 2 teams 2 thing 1 things 2 threat 1 time 2 turning 1 uniform 1 usually 1 wide 2 wild 2 win 3 working 1 years 2

3.1.1.2 CountVectorizer Bigram Model

3.1.1.2.1 CountVectorizer Bigram Details

CountVectorizer Bigram Parameters:

- input='filename'
- ngram_range=(1,2)
- stop_words='english'
- max_features=None
- max_df=1.0

Homework Assignment 3 (week 3)

- min_df=1
- analyzer=word

Both fit and transform methods were performed on this model.

3.1.1.2.2 CountVectorizer Bigram Results

Bigram Document Term Matrix Dimensions

Size	Rows	Columns(vocabulary)	DataType
11507	32	8240	scipy.sparse.csr.csr_matrix

: CountVector Bigram Spars Matrix Snippet

	aaron	aaron jones	aaron rodgers	abdullah	abdullah clear	abdullah played	ability	ability fairly	ability make	ability navigate	...	zeke best	zeke currently	ziggy	ziggy ansah	zip	zip used	zone	zone best
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
10	2	0	2	0	0	0	1	0	1	0	...	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0

Below is a sampling of the Bigram output file: Its format is one line per NFL Team. The first token in the line is the team name/text filename vectorization was performed on. It is then followed by the term and term frequency for each of the kept features. The full output vector file can be found in the supplemental zip file under ./output/cnt_bigrams_feature_vector_tf.txt.
(To save space in this report, only a portion of the single line was added for display)

: CountVector Unigram Feature Vector Output Example - single line for each of the NFL Teams -

new_york_jets_nfl_team_text.txt able 2 able numbers 1 able steelers 1 adds 1 adds lot 1 afc 4 afc east 2 afc offer 2 agency 1 agency usually 1 air 1 air simply 1 allows 1 allows pressure 1 anderson 1 anderson enunwa 1 ball 2 ball make 2 bell 3 bell comes 1 bell jets 1 bell poses 1 berth 2 berth form 2 best 2 best pieces 1 best wide 1 better 2 better able 1 better jets 1 big 1 big signing 1 biggest 1 biggest needs 1 bowl 2 bowl slated 2 breakout 2 breakout second 2 buster 1 buster skrine 1 campaign 2 campaign predicted 2 card 2 card spot 2 centerpieces 1 centerpieces able 1 certainly 4 certainly adds 1 certainly better 1 certainly like 2 cj 1 cj mosely 1 comes 3 comes fruition 2 comes jets 1 contract 1 contract ended 1 core 1 core league 1 crowder 1 crowder fit 1 darnold 3 darnold breakout 2 darnold options 1 dead 2 dead campaign 2 defense 1 defense especially 1 direction 2 direction jets 2 earn 2 earn postseason 2 east 4 east huge 2 east said 2 ended 1 ended steelers 1 england 2 england patriots 2 enunwa 1 enunwa best 1 especially 1 especially face 1 face 1 face losing 1 favorites 2 favorites likely 2 finish 2 finish new 2 finishing 4 finishing dead 2 finishing second 2 fit 1 fit signed 1 form 2 form wild 2 free 1 free agency 1 fruition 2 fruition mean 2 gives 1 gives darnold 1 going 3 going offseason 1 going starters 1 good 1 good thing 1 ground 1 ground air 1 guaranteed 1 guaranteed money 1 happened 2 happened takes 2 huge 2 huge leap 2 hurting 1 hurting jets 1 jets 16 jets biggest 1 jets defense 1 jets earn 2 jets going 2 jets middle 1 jets near 2 jets offense 1 jets picked 2 jets putting 1 jets supposed 1 jets

3.1.1.3 CountVectorizer Trigram Model

3.1.1.3.1 CountVectorizer Trigram Details

Homework Assignment 3 (week 3)

CountVectorizer Bigram Parameters:

- input='filename'
- ngram_range=(1,3)
- stop_words='english'
- max_features=None
- max_df=1.0
- min_df=1
- analyzer=word

Both fit and transform methods were performed on this model.

3.1.1.3.2 CountVectorizer Trigram Results

Trigram Matrix Dimensions

Size	Rows	Columns(vocabulary)	DataType
18202	32	14901	scipy.sparse.csr.csr_matrix

: CountVector Trigram Spars Matrix Snippet

	aaron	aaron jones	aaron jones clear	aaron rodgers	aaron rodgers returned	aaron rodgers russell	aaron rodgers seeking	abdullah	abdullah clear	abdullah clear backfield	...	zip	zip used	zip used brown	zone	zone best	zone best slot	zone season	zone season qbr
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
10	2	0	0	2	0	0	2	0	0	0	...	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0

Below is a sampling of the Bigram output file: Its format is one line per NFL Team. The first token in the line is the team name/text filename vectorization was performed on. It is then followed by the term and term frequency for each of the kept features. The full output vector file can be found in the supplemental zip file under ./output/ cnt_trigrams_feature_vector_tf.txt.
(To save space in this report, only a portion of the single line was added for display)

: CountVector Unigram Feature Vector Output Example - single line for each of the NFL Teams -

new_york_jets_nfl_team_text.txt able 2 able numbers 1 able numbers similar 1 able steelers 1 able steelers win 1 adds 1 adds lot 1 adds lot jets 1 afc 4 afc east 2 afc east said 2 afc offer 2 afc offer slightly 2 agency 1 agency usually 1 agency usually good 1 air 1 air simply 1 air simply nt 1 allows 1 allows pressure 1 allows pressure taken 1 anderson 1 anderson enunwa 1 anderson enunwa best 1 ball 2 ball make 2 ball make playoffs 2 bell 3 bell comes 1 bell comes jets 1 bell jets 1 bell jets working 1 bell poses 1 bell poses threat 1 berth 2 berth form 2 berth form wild 2 best 2 best pieces 1 best pieces buster 1 best wide 1 best wide receiver 1 better 2 better able 1 better able steelers 1 better jets 1 better jets putting 1 big 1 big signing 1 big signing jets 1 biggest 1 biggest needs 1 biggest needs wide 1 bowl 2 bowl slated 2 bowl slated finish 2 breakout 2 breakout second 2 breakout second season 2 buster 1 buster skrine 1 buster skrine jets 1 campaign 2 campaign predicted 2 campaign predicted comes 2 card 2 card spot 2 card spot likely 2 centerpieces 1 centerpieces able 1 centerpieces able numbers 1 certainly 4 certainly adds 1 certainly adds lot 1 certainly better 1 certainly better jets 1 certainly like 2 certainly like direction 2 cj 1 cj mosely 1 cj mosely spending 1 comes 3 comes fruition 2 comes fruition mean 2 comes jets 1 comes jets supposed 1 contract 1 contract ended 1 contract ended steelers 1 core 1 core league 1 core league certainly 1 crowder 1 crowder fit 1 crowder fit signed 1 darnold 3 darnold breakout 2 darnold breakout second 2 darnold options 1 darnold options allows 1 dead 2 dead campaign 2 dead

campaign predicted 2 defense 1 defense especially 1 defense especially face 1 direction 2 direction jets 2 direction jets going 2 earn 2 earn postseason 2 earn postseason berth 2 east 4 east huge 2 east huge leap 2 east said 2 east said jets 2 ended 1 ended steelers 1 ended steelers offering 1 england 2 england patriots 2 england patriots afc 2 enunwa 1 enunwa best 1 enunwa best wide 1 especially 1 especially face 1 especially face losing 1 face 1 face losing 1 face losing best 1 favorites 2 favorites likely 2 favorites likely win 2 finish 2 finish new 2 finish new england 2 finishing 4 finishing dead 2 finishing dead campaign 2 finishing second 2 finishing second east 2 fit 1 fit signed 1 fit signed jets 1 form 2 form wild 2 form wild card 2 free 1 free agency 1 free agency usually 1 fruition 2 fruition mean 2 fruition mean jets 2 gives 1 gives darnold 1 gives darnold options 1 going 3 going offseason 1 going offseason jets 1 going starters 1 going starters time 1 good 1 good thing 1 good thing contract 1 ground 1 ground air 1 ground air simply 1 guaranteed 1 guaranteed money 1 guaranteed money leveon 1 happened 2 happened takes 2 happened takes darnold 2 huge 2 huge leap 2 huge leap finishing 2 hurting 1 hurting jets 1 hurting jets offense 1 jets 16 jets biggest 1 jets biggest needs 1 jets defense 1 jets defense especially 1 jets earn 2 jets earn postseason 2 jets going 2 jets going

3.1.2 *TfidfVectorizer - Vectorize Team Text Documents*

Utilizing the python package **sklearn.feature_extraction.text TfidfVectorizer** class, this model converts a collection of team text documents to a matrix transformed to a normalized tf or tf-idf representation. This implementation of TfidfVectorizer produces a sparse representation of the counts using scipy.sparse.coo_matrix. Tf means term-frequency while tf-idf means term-frequency times inverse document-frequency. This is a common term weighting scheme in information retrieval, that has also found good use in document classification.

The goal of using tf-idf is to scale down the impact of tokens that occur very frequently in a given corpus and that are hence empirically less informative than features that occur in a small fraction of the training corpus

formula used: tf-idf(d, t) = tf(t) * idf(d, t)

- tf(t)= the term frequency is the number of times the term appears in the document
- idf(d, t) = the document frequency is the number of documents 'd' that contain term 't'

TfidfVectorizer supports counts of N-grams of words or consecutive characters. Once fitted, the vectorizer has built a dictionary of feature indices:
The index value of a word in the vocabulary is linked to its frequency in the whole training corpus.

The data for the vectorization steps is readin from a local file directory created during the prior cleaning process steps.

- path=f'{corpusDir}/teams/v2/cleaned/
- A collection of 32 documents are retrived containing media text specific to each team's performance.

For this experiment unigram, bigram and trigram models were vectorized and saved to file for analysis and future additive modeling.

Further, to perform a comparative analysis on the individual team documents the tfidf vectorized trigram model was passed through Vader's SentimentIntensityAnalyzer in order to perform unsupervised classification of the trigram terms. This process will be described under section 3.1.3, Vectorization Results

3.1.2.1 TfidfVectorizer Unigram Model

3.1.2.1.1 TfidfVectorizer Unigram Details

TfidfVectorizer Unigram Parameters:

- input='filename'

- ngram_range=(1,1)
- stop_words='english'
- max_features=None
- max_df=1.0
- min_df=1
- analyzer=word

Both fit and transform methods were performed on this model.

3.1.2.1.2 TfidfVectorizer Unigram Results

: Unigram Matrix Dimensions

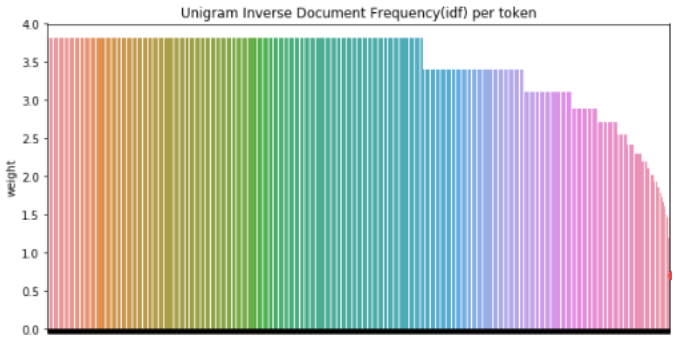
Size	Rows	Columns(vocabulary)	DataType
4963	32	2113	scipy.sparse.csr.csr_matrix

IDF Top 10 List:

feature	weight
handing	3.80336
historically	3.80336
horrible	3.80336
hobkins	3.80336
testing	3.80336
hopes	3.80336
hopeless	3.80336
texans	3.80336
theme	3.80336
honest	3.80336

IDF Lowest 10 List:

feature	weight
season	1.0625
year	1.0625
team	1.2006
make	1.3610
likely	1.3610
game	1.4054
going	1.4519
like	1.4519
offense	1.4519
games	1.5007



Below is a sampling of the Unigram output file: It's format is one line per NFL Team. The first token in the line is the team name/text filename vectorization was performed on. It is then followed by the term and tfidf weighted frequency for each of the kept features. The full output vector file can be found in the supplemental zip file under ./output/tfidf_unigram_feature_vector_tf.txt.
(To save space in this report, only a portion of the single line was added for display)

: TfidfVector Unigram Feature Vector Output Example - single line for each of the NFL Teams -

new_york_jets_nfl_team_text.txt able 0.044428487367548605 adds 0.03907812534221443 afc 0.17496500834091186 agency 0.031106457522029847 air 0.031106457522029847 allows 0.043741252085227965 anderson 0.043741252085227965 ball 0.058667238026818315 bell 0.11723437602664329 berth 0.08748250417045593 best 0.03569970568803523 better 0.034519889580494616 big 0.018471662958831726 biggest 0.031106457522029847 bowl 0.044428487367548605 breakout 0.07815625068442886 buster 0.043741252085227965 campaign 0.08748250417045593 card 0.06640656086123294 centerpieces 0.043741252085227965 certainly 0.0822739434696734 cj 0.03576958426504338 comes 0.09331937256608953 contract 0.026443330779016304 core 0.023134789701845265 crowder 0.043741252085227965 darnold 0.1312237562556839 dead 0.08748250417045593 defense 0.021361951193224572 direction 0.07815625068442886 earn 0.08748250417045593 east 0.17496500834091186 ended 0.03576958426504338 england 0.07815625068442886 enunwa 0.043741252085227965 especially 0.029333619013409157 face 0.031106457522029847 favorites 0.08748250417045593 finish 0.07815625068442886 finishing 0.17496500834091186 fit 0.03907812534221443 form 0.058667238026818315 free 0.025231612610431893 fruition

0.08748250417045593 gives 0.03576958426504338 going 0.050096473350633094 good 0.01912902390761698 ground 0.03576958426504338 guaranteed 0.03907812534221443 happened 0.07153916853008677 huge 0.050463225220863786 hurting 0.043741252085227965 jets 0.6998600333636474 league 0.017849852844017616 leap 0.07153916853008677 leveon 0.043741252085227965 like 0.033397648900422065 likely 0.06261034651080656 linebacker 0.025231612610431893 losing 0.031106457522029847 lot 0.023134789701845265 main 0.03320328043061647 make 0.03130517325540328 mean 0.07153916853008677 middle 0.03320328043061647 million 0.023134789701845265 money 0.029333619013409157 mosely 0.07815625068442886 near 0.08748250417045593 needs 0.03320328043061647 new 0.0694043691055358 nt 0.023134789701845265 numbers 0.029333619013409157 offense 0.016698824450211033 offer 0.07153916853008677 offering 0.043741252085227965

3.1.2.2 TfidfVectorizer Bigram Model

3.1.2.2.1 TfidfVectorizer Bigram Details

TfidfVectorizer Bigram Parameters:

- input='filename'
- ngram_range=(1,2)
- stop_words='english'
- max_features=None
- max_df=1.0
- min_df=1
- analyzer=word

Both fit and transform methods were performed on this model.

3.1.2.2.2 TfidfVectorization Bigram Results

: Bigram Matrix Dimensions

Size	Rows	Columns(vocabulary)	DataType
4963	32	2113	scipy.sparse.csr.csr_matrix

IDF Top 10 List:		IDF Lowest 10 List:	
feature	**weight**	**feature**	**weight**
lot improve	3.80336	season	1.0625
poor secondary	3.80336	year	1.0625
position greg	3.80336	team	1.2006
position deominant	3.80336	likely	1.3610
position count	3.80336	make	1.3610
position chemistry	3.80336	game	1.4054
position certain	3.80336	going	1.4519
position backup	3.80336	like	1.4519
position age	3.80336	offense	1.4519
position aaron	3.80336	games	1.5007

Below is a sampling of the Bigram output file: It's format is one line per NFL Team. The first token in the line is the team name/text filename vectorization was performed on. It is then followed by the term and tfidf weighted frequency for each of the kept features. The full output vector file can be found in the supplemental zip file

under ./output/tfidf_bigram_feature_vector_tf.txt
(To save space in this report, only a portion of the single line was added for display)

: TfidfVector Bigram Feature Vector Output Example - single line for each of the NFL Teams -

new_york_jets_nfl_team_text.txt able 0.03440558688825807 able numbers 0.03387338931369401 able steelers 0.03387338931369401 adds 0.030262246512444857 adds lot 0.03387338931369401 afc 0.13549355725477605 afc east 0.06774677862738802 afc offer 0.06774677862738802 agency 0.0240889571190269 agency usually 0.03387338931369401 air 0.0240889571190269 air simply 0.03387338931369401 allows 0.03387338931369401 allows pressure 0.03387338931369401 anderson 0.03387338931369401 anderson enunwa 0.03387338931369401 ball 0.045432128686428146 ball make 0.06774677862738802 bell 0.09078673953733457 bell comes 0.03387338931369401 bell jets 0.03387338931369401 bell poses 0.03387338931369401 berth 0.06774677862738802 berth form 0.06774677862738802 best 0.02764598568872475 best pieces 0.03387338931369401 best wide 0.03387338931369401 better 0.026732331679657485 better able 0.03387338931369401 better jets 0.03387338931369401 big 0.014304524924359789 big signing 0.03387338931369401 biggest 0.0240889571190269 biggest needs 0.03387338931369401 bowl 0.03440558688825807 bowl slated 0.06774677862738802 breakout 0.060524493024889714 breakout second 0.06774677862738802 buster 0.03387338931369401 buster skrine 0.03387338931369401 campaign 0.06774677862738802 campaign predicted 0.06774677862738802 card 0.05142548925332931 card spot 0.060524493024889714 centerpieces 0.03387338931369401 centerpieces able 0.03387338931369401 certainly 0.06371324972799017 certainly adds 0.03387338931369401 certainly better 0.03387338931369401 certainly like 0.06774677862738802 cj 0.027700099920276056 cj mosely 0.030262246512444857 comes 0.07226687135708071 comes fruition 0.06774677862738802 comes jets 0.03387338931369401 contract 0.020477814317777742 contract ended 0.03387338931369401 core 0.017915667725608944 core league 0.030262246512444857 crowder 0.03387338931369401 crowder fit 0.03387338931369401 darnold 0.10162016794108204 darnold breakout 0.06774677862738802 darnold options 0.03387338931369401 dead 0.06774677862738802 dead campaign 0.06774677862738802 defense 0.016542774949796113 defense especially 0.03387338931369401 direction 0.060524493024889714 direction jets 0.06774677862738802 earn 0.06774677862738802 earn postseason 0.06774677862738802 east 0.13549355725477605 east huge 0.06774677862738802 east said 0.06774677862738802 ended 0.027700099920276056 ended steelers 0.03387338931369401 england 0.060524493024889714 england patriots 0.06774677862738802 enunwa 0.03387338931369401 enunwa best 0.03387338931369401 especially 0.022716064343214073 especially face 0.03387338931369401 face 0.0240889571190269 face losing 0.03387338931369401 favorites 0.06774677862738802 favorites likely 0.06774677862738802 finish 0.060524493024889714 finish new 0.06774677862738802 finishing 0.13549355725477605 finishing dead 0.06774677862738802 finishing second 0.06774677862738802 fit

3.1.2.3 TfidfVectorizer Trigram Model

3.1.2.3.1 TfidfVectorizer Trigram Details

TfidfVectorizer Bigram Parameters:

- input='filename'
- ngram_range=(1,3)
- stop_words='english'
- max_features=None
- max_df=1.0
- min_df=1
- analyzer=word

Both fit and transform methods were performed on this model.

3.1.2.3.2 TfidfVectorizer Trigram Results

: Trigram Matrix Dimensions

Size	Rows	Columns(vocabulary)	DataType

Homework Assignment 3 (week 3)

18202	32	14901	scipy.sparse.csr.csr_matrix

IDF Top 10 List:		IDF Lowest 10 List:	
feature	**weight**	**feature**	**weight**
major leap foolhardy	3.80336	year	1.0625
pro bowl led	3.80336	season	1.0625
prior acquiring josh	3.80336	team	1.2006
prior year	3.80336	make	1.3610
prior year tyrann	3.80336	likely	1.3610
priority draft	3.80336	game	1.4054
priority draft seleted	3.80336	like	1.4519
priority likely	3.80336	offense	1.4519
priority likely result	3.80336	going	1.4519
priority passer	3.80336	games	1.5007

Below is a sampling of the Trigram output file: Its format is one line per NFL Team. The first token in the line is the team name/text filename vectorization was performed on. It is then followed by the term and tfidf weighted frequency for each of the kept features. The full output vector file can be found in the supplemental zip file under ./output/tfidf_trigram_feature_vector_tf.txt
(To save space in this report, only a portion of the single line was added for display)

: TfidfVector Trigram Feature Vector Output Example - single line for each of the NFL Teams -

new_york_jets_nfl_team_text.txt able 0.02900004318310668 able numbers 0.02855146043710011 able numbers similar 0.02855146043710011 able steelers 0.02855146043710011 able steelers win 0.02855146043710011 adds 0.025507672882575636 adds lot 0.02855146043710011 adds lot jets 0.02855146043710011 afc 0.11420584174840044 afc east 0.05710292087420022 afc east said 0.05710292087420022 afc offer 0.05710292087420022 afc offer slightly 0.05710292087420022 agency 0.020304283689641017 agency usually 0.02855146043710011 agency usually good 0.02855146043710011 air 0.020304283689641017 air simply 0.02855146043710011 air simply nt 0.02855146043710011 allows 0.02855146043710011 allows pressure 0.02855146043710011 allows pressure taken 0.02855146043710011 anderson 0.02855146043710011 anderson enunwa 0.02855146043710011 anderson enunwa best 0.02855146043710011 ball 0.03829417873573676 ball make 0.05710292087420022 ball make playoffs 0.05710292087420022 bell 0.07652301864772691 bell comes 0.02855146043710011 bell comes jets 0.02855146043710011 bell jets 0.02855146043710011 bell jets working 0.02855146043710011 bell poses 0.02855146043710011 bell poses threat 0.02855146043710011 berth 0.05710292087420022 berth form 0.05710292087420022 berth form wild 0.05710292087420022 best 0.02330245902842549 best pieces 0.02855146043710011 best pieces buster 0.02855146043710011 best wide 0.02855146043710011 best wide receiver 0.02855146043710011 better 0.022532351376914634 better able 0.02855146043710011 better able steelers 0.02855146043710011 better jets 0.02855146043710011 better jets putting 0.02855146043710011 big 0.01205710694218192 big signing 0.02855146043710011 big signing jets 0.02855146043710011 biggest 0.020304283689641017 biggest needs 0.02855146043710011 biggest needs wide 0.02855146043710011 bowl 0.02900004318310668 bowl slated 0.05710292087420022 bowl slated finish 0.05710292087420022 breakout 0.05101534576515127 breakout second 0.05710292087420022 breakout second season 0.05710292087420022 buster 0.02855146043710011 buster skrine 0.02855146043710011 buster skrine jets 0.02855146043710011 campaign 0.05710292087420022 campaign predicted 0.05710292087420022 campaign predicted comes 0.05710292087420022 card 0.04334590814865311 card spot 0.05101534576515127 card spot likely 0.05710292087420022 centerpieces 0.02855146043710011 centerpieces able 0.02855146043710011 centerpieces able numbers 0.02855146043710011 certainly 0.053703109307469854 certainly adds 0.02855146043710011 certainly adds lot 0.02855146043710011 certainly better 0.02855146043710011 certainly better jets 0.02855146043710011 certainly like 0.05710292087420022 certainly like direction 0.05710292087420022 cj 0.023348071244165488 cj mosely 0.025507672882575636 cj mosely spending 0.02855146043710011 comes 0.06091285106892305 comes fruition 0.05710292087420022 comes fruition mean 0.05710292087420022 comes jets 0.02855146043710011 comes jets supposed

3.1.3 *Vectorization Classification Results*

To assess the overall sentiment of the media text on this corpus and it's individual team documents, Vader Sentiment Classification was performed on the full corpus trigram model created in the above sections, as well as each individual teams trigram model in order to establish a sentiment ranking score to compare each of the NFL Teams to.

Behind Vader's scoring is its core sentiment analysis engine, vaderSentiment.py.

"The Python code for the rule-based sentiment analysis engine. Implements the grammatical and syntactical rules described in the paper, incorporating empirically derived quantifications for the impact of each rule on the perceived intensity of sentiment in sentence-level text. Importantly, these heuristics go beyond what would normally be captured in a typical bag-of-words model. They incorporate **word-order sensitive relationships** between terms. For example, degree modifiers (also called intensifiers, booster words, or degree adverbs) impact sentiment intensity by either increasing or decreasing the intensity. " - https://github.com/cjhutto/vaderSentiment

Vader Scoring:

"The compound score is computed by summing the valence scores of each word in the lexicon, adjusted according to the rules, and then normalized to be between -1 (most extreme negative) and +1 (most extreme positive). This is the most useful metric if you want a single unidimensional measure of sentiment for a given sentence. " - https://github.com/cjhutto/vaderSentiment

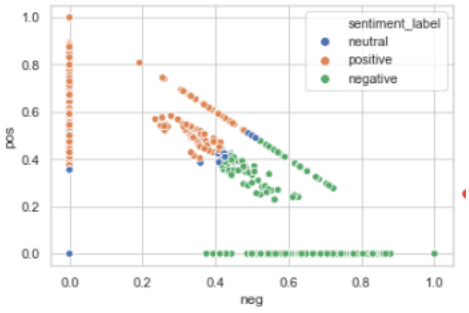
To standardize thresholds for classifying these sentences as either positive, neutral, or negative the recommended compound score values from the github vaderSentiment site were used.
The were as follows:

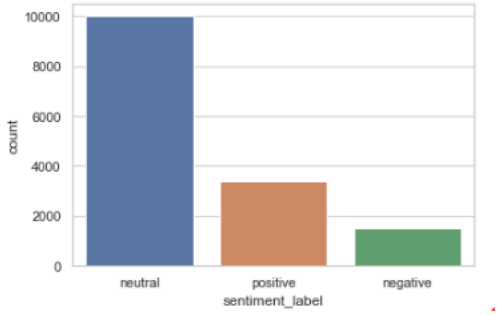
1. positive sentiment: compund score >= 0.05
2. neutral sentiment: (compund score > -0.05) and (compund score < 0.05)

negative sentiment: compund score <= -0.05

Vader polarity scoring on the trigram model:

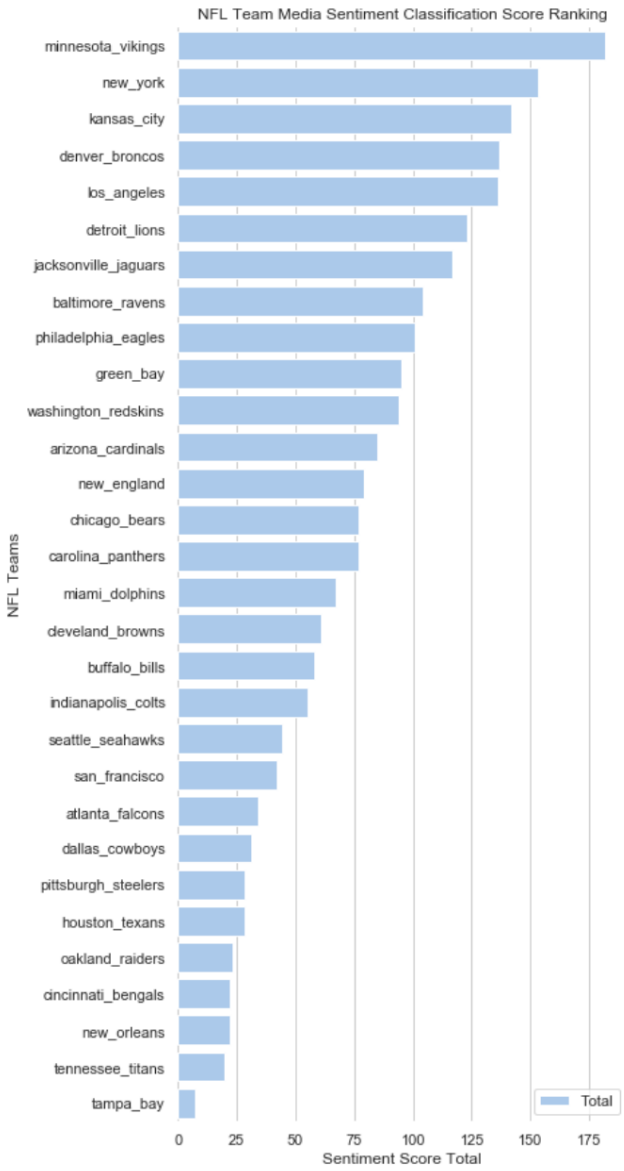
	compound	neg	neu	pos
count	14901.000000	14901.000000	14901.000000	14901.000000
mean	0.055914	0.070670	0.768491	0.160838
std	0.238882	0.207285	0.336176	0.292888
min	-0.877900	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.426000	0.000000
50%	0.000000	0.000000	1.000000	0.000000
75%	0.000000	0.000000	1.000000	0.000000
max	0.855500	1.000000	1.000000	1.000000





- Negative Count: 1495
- Positive Count: 3397
- Neutral Count: 10009

Homework Assignment 3 (week 3)



- Scoring for each team was performed by transform term_labels into numeric numbers for summation.
  - Postive = 1
  - Negative = -1
  - Neutral = 0

- The dataframe matrix was then grouped by team and summed by it's grouped label_scores.

The sentiment classification gives Minnesota Vikings a highly positive scoring whereas the Tampa Bay Buccaneers a very poor scoring. Though no teams showed a completely negative scoring.

Further looking into each teams records or possible prospects would help give insights into if the overall classifications are accurate and have something backing it up.

<u>Example Data Points to look at:</u>

- Tampa Bay Buccaneers 2018 Record:
  - 5-11
  - Placed 4th in the NFC South

- Minnesota Vikings 2018 Record:
  - 8-7-1
  - Did not qualify for playoffs

Classification for the complete trigram corpus can be found in the attached zip file under './output/vader_trigram_sentiment.txt'.

4 Conclusions

Further analysis should be conducted on this dataset using more of the parameter options available in the vectorization package used for this trial to assess if the best vectorization options were chosen. Many combination variations would yield different results provided the dataset is rich enough in content and representative of the general public and media opinion toward the NFL Teams and its players. Given the subjective nature of sentiment analysis (sometimes know as opinion mining or emotion AI) which is used to systematically identify, extract, quantify, and study affective states and subjective information, accuracy in the text preprocessing steps for vectorization is the most critical component of the overall pipeline workflow.

Also to consider would be adding custom stop words or words specific to football and the NFL that could appear as negative or positive, when in fact it's the opposite for this domain of sports. Fantasy football is an addictive game, fans who could have AI intelligence helping them make chooses or at minimum rule out the noise would benefit greatly from this technology.