2019-1002 IST 736
Text Mining

Final Project

Public Sentiment Toward NFL Team, Coach, Player
Can it predict weekly Fantasy Football outcomes?

Ryan Timbrook
David Madsen
Diego Vales

Course: IST 736 Text Mining
Term: Fall, 2019

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 Fantasy stats and which service provider like yahoo fantasy sports, cbs fantasy sports, and espn fantasy sports 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 its closest rival basketball at 11%; baseball is at 9%, after decades of declining stature. (Norman, 2018) NFL revenue grew an estimated $900 million to $14 billion in 2017; in 2018, it generated about $15 billion. (Soshnick & Novy-Williams, 2019) The league announced in January that it's aiming to boost its annual revenue to $25 billion by 2027. (Florio, 2010) 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. (Houghton, Nowlin, & Walker, 2019)

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." (ESPN Fantasy Insights with Watson, n.d.)

Football sports fans generate a large amount of Twitter tweets which reflect their opinions and feelings about what is happening not only during a game, but also throughout the week as the team prepares for the next game. The aim of this research is to use text mining techniques on public news and media web sites like Twitter to collect and examin the sentiment convedy through these sources, aggregate data relating to NFL teams, players, and coach, to determine if public sentiment alone is a predictor of Player Fantasy Football performance stats.

For example, if Deshaun Watson's Fantasy Football stats forecast for NFL season week 6 is 85 points leading into that week’s game, can the public opinion for that week of negative, neutral, or positive predict if he will achieve that points threshold or not. Thereby giving a player of Fantasy Football an edge in choosing to either keep Deshaun Watson as his or her starting quarterback or bench him for someone else.

1.1 Purpose

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

1.2 Scope

- Gather public data using text mining techniques on:
    - Public opinion toward NFL teams, coaches, and players.
        - Reduce data gathering to one of each for initial POC.
        - Data is in time-series format from the first week of the NFL 2019 session to current schedule week
    - Players weekly Fantasy Football performance stat forecasts on a daily time scale.
- Perform sentiment analysis modeling ML techniques on data set to determine model prediction accuracies.
- Perform unsupervised topic modeling ML techniques on document text to gain insights into public opinion and primary opinion drivers.
- Evaluate weekly sentiment trends aligning with Fantasy Football performance stat forecasts.

2 Initial Data Mining

2.1 About the Data

Gathering data on the NFL Team, Coach, and Player selected for this experiment was mined through the Twitter Developer Platform, Twitter APIs. To have data for analysis which covered the entire 2019 NFL weekly schedule, multiple Twitter API types were required.

Additional details on each of the API capabilities and endpoints can be found in table 2.1.1 below.

2.1.1 *Search Tweets Features Used*

Category	Product name	Supported history	Query capability	Counts endpoint	Data fidelity
Standard	Standard Search API	7 days	Standard operators	Not available	Incomplete
Premium	Search Tweets: 30-day endpoint	30 days	Premium operators	Available	Full
Premium	Search Tweets: Full-archive endpoint	Tweets from as early as 2006	Premium operators	Available	Full

2.1.2 *Tweet Objects*

Tweets are the basic atomic building block of all things Twitter. Tweets are also known as “status updates.” The Tweet object has a long list of ‘root-level’ attributes, including fundamental attributes such as id, created_at, and text. Tweet objects are also the ‘parent’ object to several child objects. Tweet child objects include user, entities, and extended_entities. Geo-tagged tweets will have a place child object

When mining for tweets, tweet objects are returned as JSON objects similar to this example structure:
The JSON will be a mix of ‘root-level’ attributes (here we are highlighting some of the most fundamental attributes), and child objects (which are represented here with the {} notation):

```
{
 "created_at": "Wed Oct 10 20:19:24 +0000 2018",
 "id": 1050118621198921728,
 "id_str": "1050118621198921728",
 "text": "To make room for more expression, we will now count all emojis as equal—including those with gend
er and skin t… https://t.co/MkGjXf9aXm",
 "user": {},
 "entities": {}
}
```

**Tweet Data Dictionary can be found here.

2.1.3 *Premium Search Basic Requirements*

The following is needed:

- A developer account
- A registered app
- A developer environment setup
- Authentication - for cURL need a bearer token, for Twurl need to have Twurl setup

2.1.4 *Accessing the data endpoint*

The data endpoint will provide the full Tweet payload of matched tweets. Use the from: and lang: operators to find Tweets originating from @TwitterDev in English. *For more operators, click here.*

Details on the data endpoint response payload that is returned from the API search request can be found here.

2.1.4.1 Search Endpoints

- **30day**: "https://api.twitter.com/1.1/tweets/search/30day/sandbox.json"
  - Requests Usage Monthly limit: 250
  - Rate Limit Per Request: 100
- **full archive**: "https://api.twitter.com/1.1/tweets/search/fullarchive/devfullarchive.json"
  - Requests Usage Monthly limit: 100
  - Rate Limit Per Request: 500

2.1.4.2 Search Parameters

- query:
  - These were the individual search terms used based on NFL type being processed
    - Team: "Houston Texans"
    - Coach: "Bill Obrien"
    - Player: "Deshaun Watson"

- **fromDate**: shown below in section 2.1.4.2.1
- **toDate**: shown below in section 2.1.4.2.1
- **maxResults**: shown below in section 2.1.4.2.1
- **next_page**: this value is returned in the JSON response payload per request

2.1.4.2.1 Search Date Range Endpoint Mapping

For each NFL type, team, coach, and player, individual search requests were performed specifically to a week date range, which represents the NFL game schedule. For this, different search API endpoints were utilized based on their search time frame capabilities and rate limit restrictions.

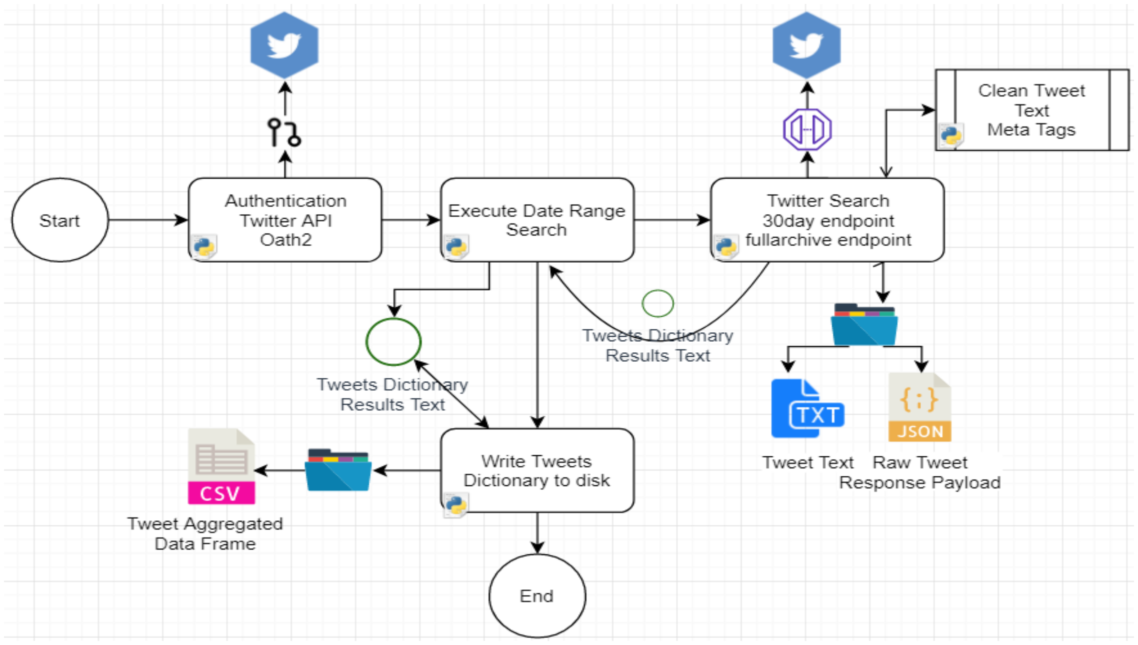
NFL Weekly Schedule	From Date - To Date	Search Endpoint	Max Results
Week 1	('201909010000','201909080000')	fullarchive	500
Week 2	('201909080000','201909150000')	fullarchive	500
Week 3	('201909150000','201909220000')	fullarchive	500
Week 4	('201909220000','201909290000')	fullarchive	500
Week 5	('201909290000','201910060000')	fullarchive	500
Week 6	('201910060000','201910130000')	fullarchive	500
Week 7	('201910130000','201910200000')	fullarchive	500
Week 8	('201910200000','201910270000')	fullarchive	500
Week 9	('201910270000','201911030000')	fullarchive	500
Week 10	('201911030000','201911100000')	30day	100
Week 11	('201911100000','201911170000')	30day	100

Week 12	('201911100000','201911170000')	30day	
Week 13	('201911240000','201912010000')	30day	

2.1.5 ***Twitter Search Function Behavior***

Jupyter Notebook Names:

- **search_twitter_nfl_coach_premium**.ipynb
- **search_twitter_nfl_team_premium**.ipynb
- **search_twitter_nfl_player_premium**.ipynb


Start
Authentication Twitter API Oath2
Execute Date Range Search
Twitter Search 30day endpoint fullarchive endpoint
Clean Tweet Text Meta Tags
Tweets Dictionary Results Text
Tweets Dictionary Results Text
TXT
JSON
Tweet Text
Raw Tweet Response Payload
Write Tweets Dictionary to disk
CSV
Tweet Aggregated Data Frame
End

2.1.5.1 Data Cleaning

At this stage, tweet text were cleaned of hashtags, urls, and @tags. Each of these types was captured and stored into lists. During the final stage of the process after all tweets returned from all requests, these data elements were packaged into a data frame and written to disk as a csv file. Each of the records has an ID attribute that maps it back to the original tweet document it's associated to.

2.1.5.2 Data Modeling

The tweet data collected during the search process captures

- **id**: tweet object response attributed (result["id_str"])
- **created_at**: tweet object response attribute (result["created_at"])
- **date**: aggregated composite value created from created_at value
- **time**: aggregated composite value created from created_at value
- **user**: tweet object response attributed (result["user"]["screen_name"])
- **text**: tweet object response attributed (result["text"])
- **favorite_count**: tweet object response attributed (result["user"]["favourites_count"])
- **year**: aggregated composite value created from created_at value

- **month**: aggregated composite value created from created_at value
- **day_of_month**: aggregated composite value created from created_at value
- **day_of_week**: aggregated composite value created from created_at value

2.1.5.3 Data Export

Four data files are generated as output from this process.

These first two are written out at the end of the process from memory DataFrame objects that were updated throughout the search iterations. It contains a complete list of all tweet documents collected.

Output OS Path Patterns: outputPath = f'{dataDir}/{nfl_type}/{search_on}/v{search_iteration}'

- search_result_tweet_text_data.csv
- search_result_tweet_text_meta.csv

This process, when executed over the three NFL types, outputs three csv data files:

- **coach_search_results_tweet_data.csv**
- **player_search_resutls_tweet_data.csv**
- **team_search_results_tweet_data.csv**

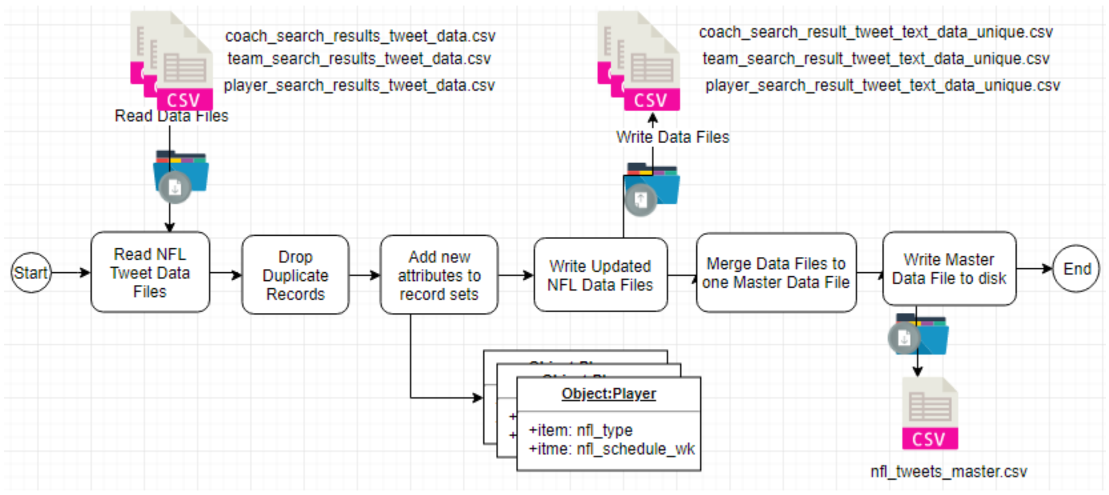
These two files are written to during a search process where the process iterates over a list of tweets returned in each request.
Output OS Path Patterns: outputPath = f'{dataDir}/{nfl_type}/{search_on}/v{search_iteration}/{search_range}'

- tweet_filename=f'{outputPath}/tweet_text.txt'
- raw_filename=f'{outputPath}/tweet_raw.txt'

2.1.6 *Merge NFL Data Sets to Master File*

Jupyter Notebook: **merge_datasets_to_master**.ipynb

This function of the data engineering step reads in the three separate NFL Type data sets and merges them into one data set for model training and analysis. Two additional categorical attributes were added to the data sets during this process. The first was a field called nfl_type. The values are 'coach','team','player'. It's used to segment the data for post-processes granular analysis. The other field is a numeric, categorical attribute that identifies the nfl schedule week; the tweets are associated from a timeline perspective.


coach_search_results_tweet_data.csv
team_search_results_tweet_data.csv
player_search_results_tweet_data.csv
CSV
Read Data Files
coach_search_result_tweet_text_data_unique.csv
team_search_result_tweet_text_data_unique.csv
player_search_result_tweet_text_data_unique.csv
CSV
Write Data Files
Start
Read NFL Tweet Data Files
Drop Duplicate Records
Add new attributes to record sets
Write Updated NFL Data Files
Merge Data Files to one Master Data File
Write Master Data File to disk
End
Object:Player
+item: nfl_type
+itme: nfl_schedule_wk
CSV
nfl_tweets_master.csv

2.1.6.1 Data Output

Four data files are generated as output from this process.

- nfl_tweets_master.csv
- coach_search_result_tweet_text_data_unique.csv
- team_search_result_tweet_text_data_unique.csv
- player_search_result_tweet_text_data_unique.csv

2.1.7 *VEDAR Sentiment Classification*

Jupyter Notebook: **classify_train_nfl_master.ipynb**
python package: vaderSentiment.vaderSentiment **SentimentIntensityAnalyzer**

To create a labeled data set for our supervised sentiment classification algorithms, Multinomial Naive Bayes and Support Vector Machines, to be modeled from, Vadar's Sentiment Intensity Analyzer was used at this stage to score and label each tweet document as either 'positive,' 'negative,' or 'neutral.'

The implementation of Vedar shown below in section 2.1.7.2 takes as input the nfl_tweet_master.csv file generated from the code detailed in section 2.1.6 and outputs the following data files:

- **sentiment_labeled_train_clean_nfl.csv**
  - this file contains vedar sentiment scores on tweets that had been cleaned prior to scoring
  - it contains an additional attribute, 'text_clean', that has the tweet text after cleaning
- **sentiment_labeled_train_nfl.csv**
  - this file contains vedar sentiment scores on tweets that had not been cleaned.

VADER (Valence Aware Dictionary and sEntiment Reasoner) is a lexicon and rule-based sentiment analysis tool that is *specifically attuned to sentiments expressed in social media*.

DESCRIPTION: Empirically validated by multiple independent human judges, VADER incorporates a "gold-standard" sentiment lexicon that is especially attuned to microblog-like contexts.

The VADER sentiment lexicon is sensitive both the **polarity** and the **intensity** of sentiments expressed in social media contexts and is also generally applicable to sentiment analysis in other domains.

2.1.7.1 VEDAR Details

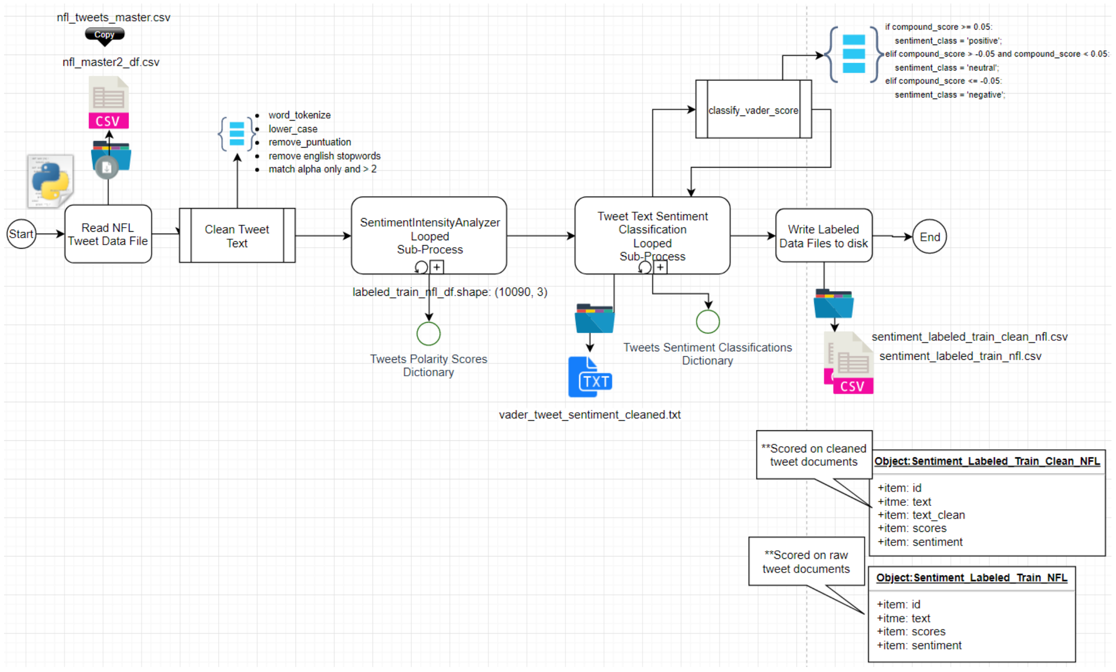
Behind Vader's scoring is its core sentiment analysis engine, vaderSentiment.py.

"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 typically 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

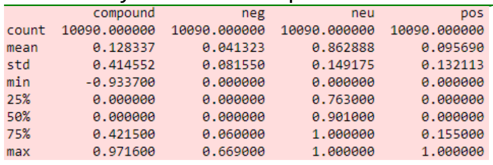
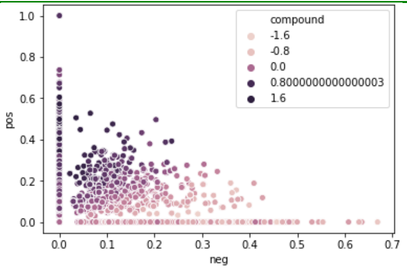
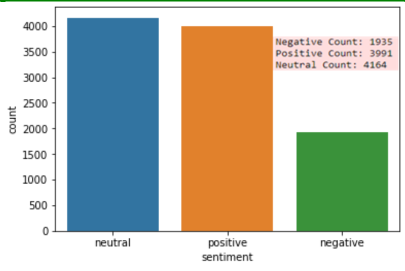
2.1.7.2 VEDAR Implementation


nfl_tweets_master.csv
Copy
nfl_master2_df.csv
CSV
word_tokenize
lower_case
remove_puntuation
remove english stopwords
match alpha only and > 2
Start
Read NFL Tweet Data File
Clean Tweet Text
SentimentIntensityAnalyzer Looped Sub-Process
labeled_train_nfl_df.shape: (10090, 3)
Tweets Polarity Scores Dictionary
classify_vader_score
Tweet Text Sentiment Classification Looped Sub-Process
TXT
vader_tweet_sentiment_cleaned.txt
Tweets Sentiment Classifications Dictionary
Write Labeled Data Files to disk
End
sentiment_labeled_train_clean_nfl.csv
sentiment_labeled_train_nfl.csv
CSV
**Scored on cleaned tweet documents
Object:Sentiment_Labeled_Train_Clean_NFL
+item: id
+itme: text
+item: text_clean
+item: scores
+item: sentiment
**Scored on raw tweet documents
Object:Sentiment_Labeled_Train_NFL
+item: id
+itme: text
+item: scores
+item: sentiment

```
if compound_score >= 0.05:
    sentiment_class = 'positive';
elif compound_score > -0.05 and compound_score < 0.05:
    sentiment_class = 'neutral';
elif compound_score <= -0.05:
    sentiment_class = 'negative';
```

2.1.7.3 VEDAR Results

Table 2.2: Vedar Polarity Scores

Polarity Scores Descriptive Statistics	Polarity Score Scatter Plot	Tweet Data Set Label Distributions
		

```
         compound          neg          neu          pos
count  10090.000000  10090.000000  10090.000000  10090.000000
mean       0.128337      0.041323      0.862888      0.095690
std        0.414552      0.081550      0.149175      0.132113
min       -0.933700      0.000000      0.000000      0.000000
25%        0.000000      0.000000      0.763000      0.000000
50%        0.000000      0.000000      0.901000      0.000000
75%        0.421500      0.060000      1.000000      0.155000
max        0.971600      0.669000      1.000000      1.000000
```

```
Negative Count: 1935
Positive Count: 3991
Neutral Count: 4164
```

Table 2.3: Vedar Polarity Scores Distributions

Polarity Compound Scores	Polarity Positive Score	Polarity Scores Negative

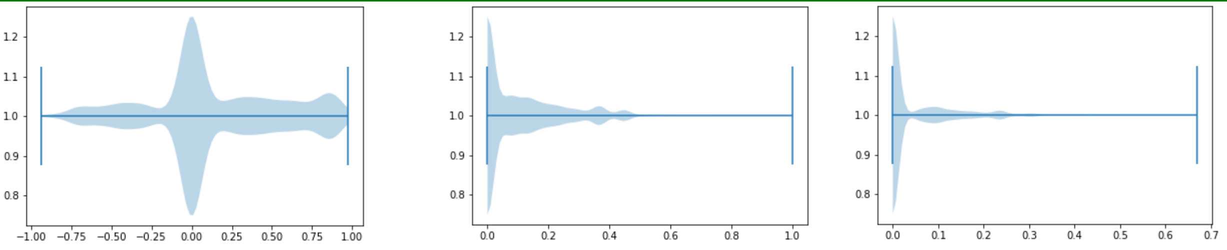
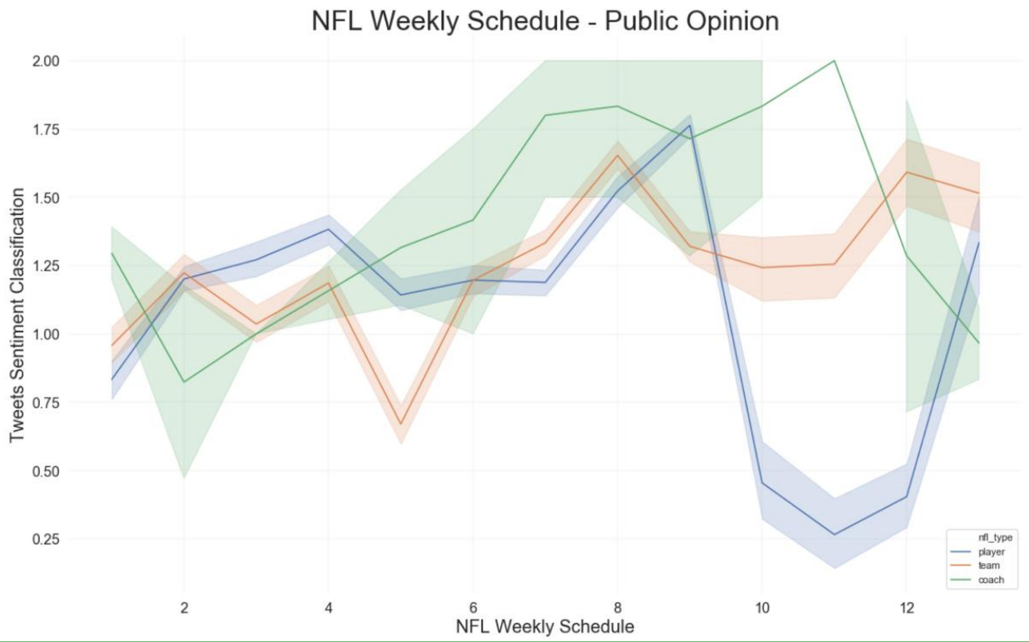


Figure 2.1: Vedar Sentiment Scoring - NFL Trend over 13 Game Weeks

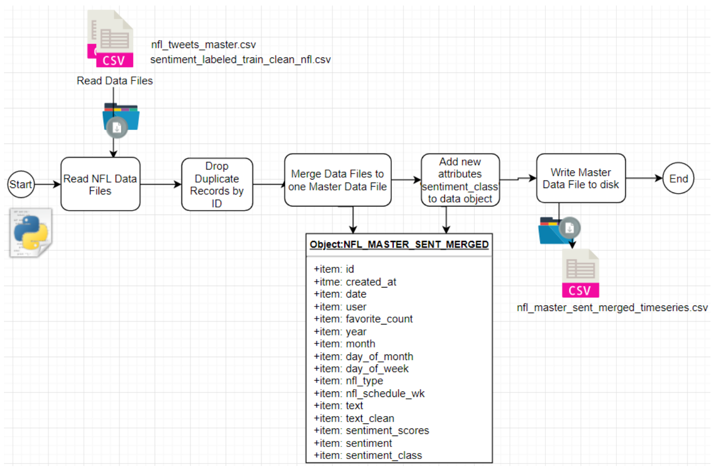


2.1.8 *Merge NFL Labeled Data Set to Master File*

Jupyter Notebook: **nfl_sentiment_analysis_data_merge_master**.ipynb

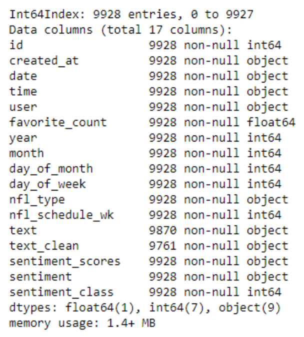
This function of the data engineering step reads in the sentiment_labeled_train_clean_nfl.csv data file generated by the process executed in section 2.1.7 and the nfl_tweets_master.csv data file generated by the process performed in section 2.1.6. And outputs a merged data file to be used for modeling, visualizations, and analysis.

2.1.8.1 Implementation



Data Output:

- nfl_master_sent_merged_timeseries.csv



```
Int64Index: 9928 entries, 0 to 9927
Data columns (total 17 columns):
id                  9928 non-null int64
created_at          9928 non-null object
date                9928 non-null object
time                9928 non-null object
user                9928 non-null object
favorite_count      9928 non-null float64
year                9928 non-null int64
month               9928 non-null int64
day_of_month        9928 non-null int64
day_of_week         9928 non-null int64
nfl_type            9928 non-null object
nfl_schedule_wk     9928 non-null int64
text                9870 non-null object
text_clean          9761 non-null object
sentiment_scores    9928 non-null object
sentiment           9928 non-null object
sentiment_class     9928 non-null int64
dtypes: float64(1), int64(7), object(9)
memory usage: 1.4+ MB
```

3 Sentiment Classification Modeling

This section evaluates two types of classification algorithms with multiple hyperparameter configuration variations on each to determine if one algorithm performs better at predicting sentiment classification on the Twitter NFL Text document data set described in section 2.

The approach is broken into three task categories, defined as:

Given a sentiment labeled data set of Twitter Text documents of NFL Coach, Team, Player, perform classification modeling tasks using both Multinominal Naive Bayes and Support Vector Machines to evaluate which modeling technique and hyperparameter configuration perform the best on unseen-held out data. Labeled tweets are on a scale of three values: negative, neutral, positive.

- Perform three classification tasks
    - Task 1:
        - Build a unigram MNB model and a unigram SVMs model.
        - Print the top 10 indicative words for the most positive category and the most negative category from the MNB and SVMs models, respectively.
        - Report the confusion matrix, precisions, and recalls.
    - Task 2:
        - Build a MNB model and a SVMs model based on both unigram and bigram. For a fair comparison, the same 60% training, 40% testing split is maintained. Likewise, all vectorization parameters are the same as in Task 1.
        - Compare the confusion matrix and other evaluation measures (accuracy, precision, recall).
    - Task 3:
        - Based on the above findings, build the best model by tuning parameters and using an 80/20 training/testing split.

3.1 Analysis and Models

3.1.1 *Models*

Jupyter Notebook: **classification_modeling_mnb_svm.ipynb**

3.1.1.1.1 CountVectorizer

Utilizing the python package **sklearn.feature_extraction.text CountVectorizer** class, this model converts a collection of customer review 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 generally 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. An N-gram is 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.

The data for these vectorization steps is the NFL twitter data output from section 2.1.8.1 section. Each of the vectorization steps below was for Multinomial Naive Baise and Support Vector Machine Classification modeling comparison.

3.1.1.1.2 CountVectorizer Details

Task 1

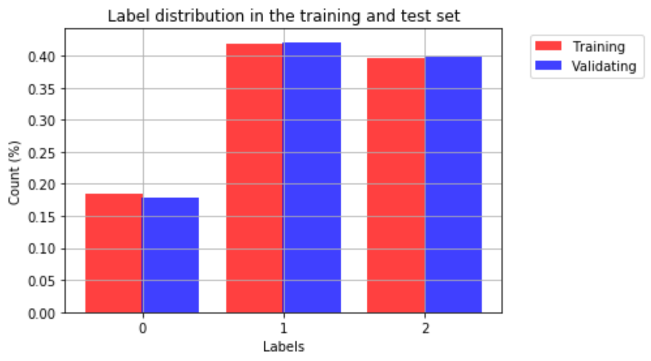
Initialize CountVectorizer unigram vector objects for MNB and SVM modeling

Vectorizer Name:
t1_mnb_count_vec_unigram
vocabulary size: 3227
Parameters:
- input='content'
- ngram_range=(1,1)
- max_features=None
- max_df=1.0
- min_df=2
- analyzer=word
- stop_words='english'

Vectorizer Name:
t1_svm_count_vec_unigram
vocabulary size: 3312
Parameters:
- input='content'
- ngram_range=(1,1)
- max_features=None
- max_df=1.0
- min_df=2
- analyzer=word
- stop_words='english'

Train Data Set Split Configurations:
- train_test_split: test_size=0.4

Figure 2.3: Task 1 Label Distributions



Task 2

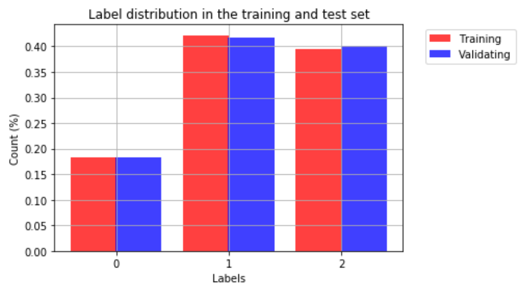
Initialize CountVectorizer bigram vector objects for MNB and SVM modeling

Vectorizer Name:
t2_mnb_count_vec_bigram
vocabulary size: 8304
Parameters:
- input='content'
- ngram_range=(1,2)
- max_features=None
- max_df=1.0
- min_df=2
- analyzer=word
- stop_words='english'

Train Data Set Split Configurations:
- train_test_split: test_size=0.4

Figure 2.4: Task 2 Label Distribution

Vectorizer Name:
t2_svm_count_vec_bigram
vocabulary size: 8665
Parameters:
- input='content'
- ngram_range=(1,2)
- max_features=None
- max_df=1.0
- min_df=2
- analyzer=word
- stop_words='english'



Task 3

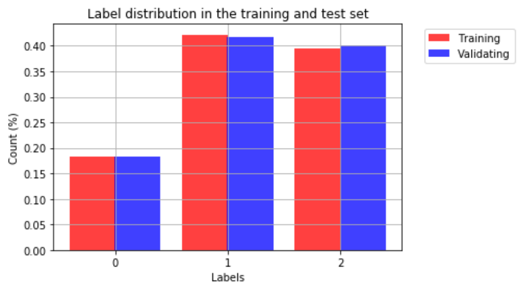
Initialize CountVectorizer bigram vector objects for MNB and SVM modeling

Vectorizer Name:
t3_svm_count_vec_unigram
vocabulary size: 3749
Parameters:
- input='content'
- ngram_range=(1,1)
- max_features=None
- max_df=1.0
- min_df=2
- analyzer=word
- stop_words='english'

Vectorizer Name: t3_svm_count_vec_bigram
Parameters:
- input='content'
- ngram_range=(1,2)
- max_features=None
- max_df=1.0
- min_df=2
- analyzer=word
- stop_words='english'

Train Data Set Split Configurations:
- train_test_split: test_size=0.2

Figure 2.5: Task 3 Label Distribution



3.1.1.2 Classification Models MNB and SVM Comparision

Train Test Split Process
Labels for the datasets were encoded using the sklearn. preprocessing LabelEncoder class.
For prediction evaluation and accuracy measurement, 40% of each dataset was held out as unseen data. The method used was sklearns.model_selection train_test_split class.

Build-Test-Validate-Predict MNB and SVM Models

Model training and validation was performed by using sklearn.model_selection cross_validate. A 10 fold cross-validation measure was used in training and validating each of the vector models training dataset. 40% of each was held out for final, unseen, prediction accuracy evaluation for Task 1 and Task 2; Task 3 was trained at an 80/20 split in order to maximize training data observations while still being able to report on accuracy scoring.

**Note: A complete listing of all model result details can be found in the .output/summary_report_final.xlsx

3.1.1.2.1 Multinominal Naive Bayes (MNB) Models

For our text classification task, we are using the Naive Bayes classifier for multinomial models.
The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts. However, in practice, fractional counts such as tf-idf may also work. - scikit-learn MultinomialNB

For this implementation, we are using scikit-learn v0.21.3 sklearn.naive_bayes MultinomialNB class.
The below steps were taken for each of the CountVectorizer vector models in the given task.

3.1.1.2.2 Support Vector Machine (SVM) Models

Linear Support Vector Classification (LinearSVC)
Python package scikit-learn v0.21.3 sklearn.svm.LinearSVC

Similar to SVC with parameter kernel='linear', but implemented in terms of liblinear rather than libsvm, so it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples.

This class supports both dense and sparse input and the multiclass support is handled according to a one-vs-the-rest scheme.
Read more in the User Guide.
Interpreting LinearSVC models:

- LinearSVC uses a one-vs-all strategy to extend the binary SVM classifier to multi-class problems
- For the NFL Twitter sentiment classification problem, there are three categories 0,1,2 with 0 as negative and 2 positive
- LinearSVC builds five three classifiers, "negative vs. others","neutral vs. others", "positive vs. others", and then pick the most confident prediction as the final prediction.
- Linear SVC also ranks all features based on their contribution to distinguish the two concepts in each binary classifier

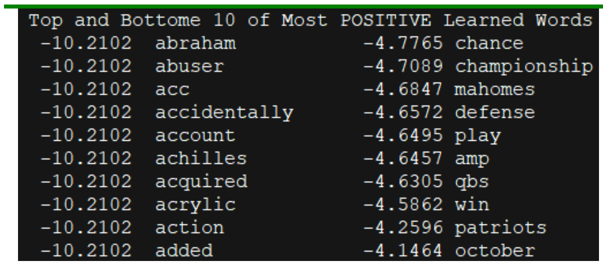
3.1.1.3 Task 1

- Build unigram models
- Print top 10 indicative words for the most positive category and the most negative category
- Report confusion matrix, precision, and recall scores.

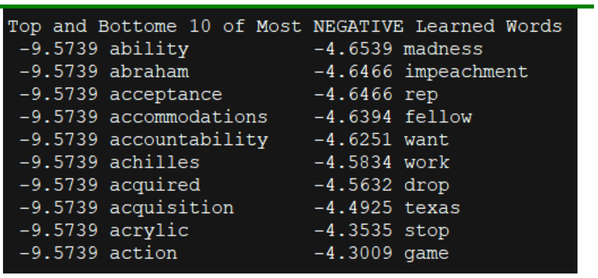
The SVM model performs better than the MNB model by 6%, 91% to 84%.

3.1.1.3.1 MNB Unigram Results

Most POSITIVE WORDS	Most NEGATIVE WORDS

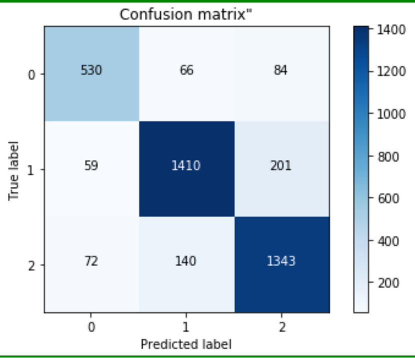


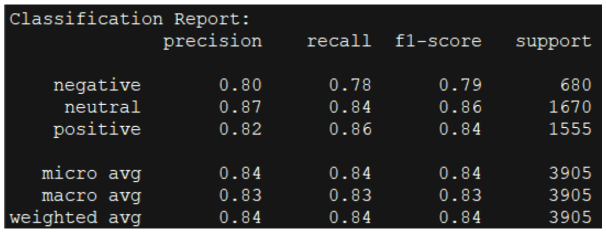
```
Top and Bottome 10 of Most POSITIVE Learned Words
 -10.2102  abraham                -4.7765 chance
 -10.2102  abuser                 -4.7089 championship
 -10.2102  acc                    -4.6847 mahomes
 -10.2102  accidentally           -4.6572 defense
 -10.2102  account                -4.6495 play
 -10.2102  achilles               -4.6457 amp
 -10.2102  acquired               -4.6305 qbs
 -10.2102  acrylic                -4.5862 win
 -10.2102  action                 -4.2596 patriots
 -10.2102  added                  -4.1464 october
```



```
Top and Bottome 10 of Most NEGATIVE Learned Words
 -9.5739 ability              -4.6539 madness
 -9.5739 abraham              -4.6466 impeachment
 -9.5739 acceptance           -4.6466 rep
 -9.5739 accommodations       -4.6394 fellow
 -9.5739 accountability       -4.6251 want
 -9.5739 achilles             -4.5834 work
 -9.5739 acquired             -4.5632 drop
 -9.5739 acquisition          -4.4925 texas
 -9.5739 acrylic              -4.3535 stop
 -9.5739 action               -4.3009 game
```

MNB Model Prediction Accuracy Results: 84%

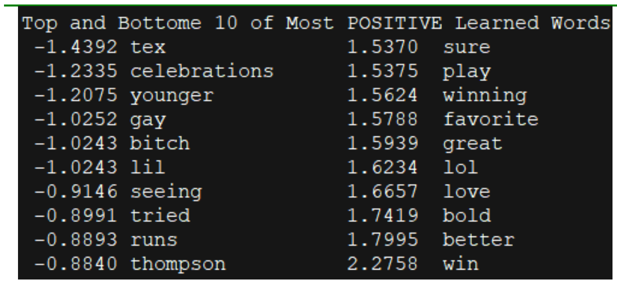




```
Classification Report:
              precision    recall  f1-score   support

    negative       0.80      0.78      0.79       680
     neutral       0.87      0.84      0.86      1670
    positive       0.82      0.86      0.84      1555

   micro avg       0.84      0.84      0.84      3905
   macro avg       0.83      0.83      0.83      3905
weighted avg       0.84      0.84      0.84      3905
```

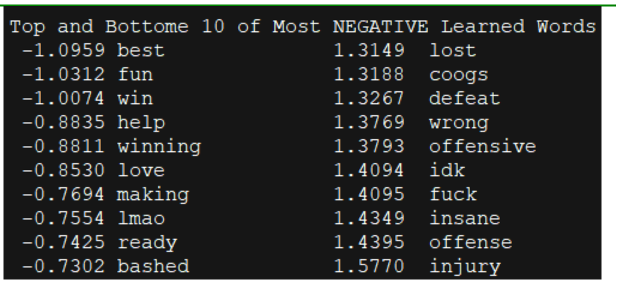
3.1.1.3.2 SVM Unigram Results

Most POSITIVE WORDS



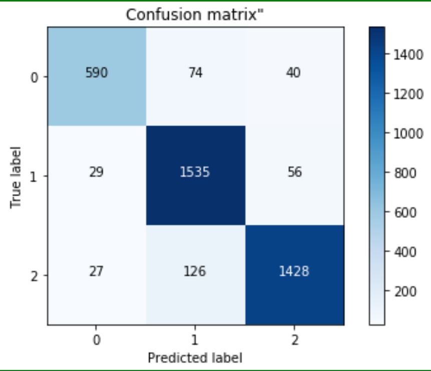
```
Top and Bottome 10 of Most POSITIVE Learned Words
 -1.4392 tex                      1.5370  sure
 -1.2335 celebrations             1.5375  play
 -1.2075 younger                  1.5624  winning
 -1.0252 gay                      1.5788  favorite
 -1.0243 bitch                    1.5939  great
 -1.0243 lil                      1.6234  lol
 -0.9146 seeing                   1.6657  love
 -0.8991 tried                    1.7419  bold
 -0.8893 runs                     1.7995  better
 -0.8840 thompson                 2.2758  win
```

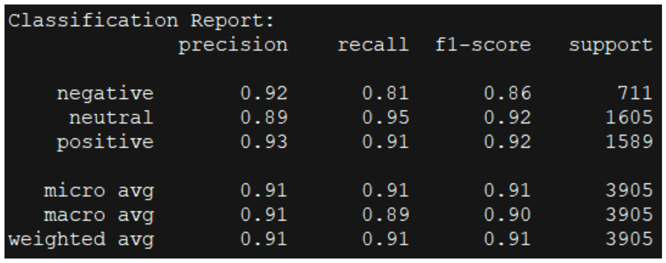
Most NEGATIVE WORDS



```
Top and Bottome 10 of Most NEGATIVE Learned Words
 -1.0959 best                     1.3149  lost
 -1.0312 fun                      1.3188  coogs
 -1.0074 win                      1.3267  defeat
 -0.8835 help                     1.3769  wrong
 -0.8811 winning                  1.3793  offensive
 -0.8530 love                     1.4094  idk
 -0.7694 making                   1.4095  fuck
 -0.7554 lmao                     1.4349  insane
 -0.7425 ready                    1.4395  offense
 -0.7302 bashed                   1.5770  injury
```

SVM Model Prediction Accuracy Results: 91%





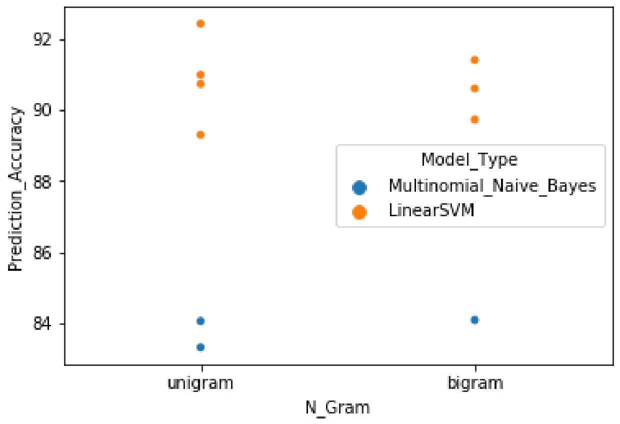
```
Classification Report:
              precision    recall  f1-score   support

    negative       0.92      0.81      0.86       711
     neutral       0.89      0.95      0.92      1605
    positive       0.93      0.91      0.92      1589

   micro avg       0.91      0.91      0.91      3905
   macro avg       0.91      0.89      0.90      3905
weighted avg       0.91      0.91      0.91      3905
```

3.1.1.4 Task 2

- Build bigram models
- Print top 10 indicative words for the most positive category and the most negative category
- Report confusion matrix, precision, and recall scores.
- Compare evaluation measures

The SVM model performs better than the MNB model by 6%, 91% to 84%. Similar results as in the Unigram tests.

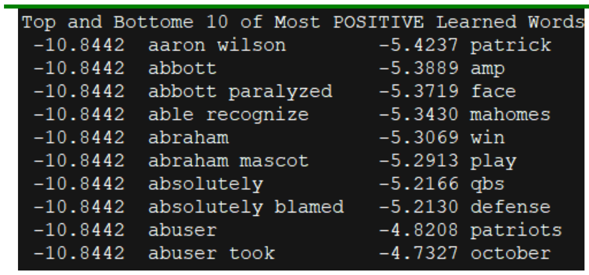
3.1.1.4.1 Model Comparison Prediction Accuracies



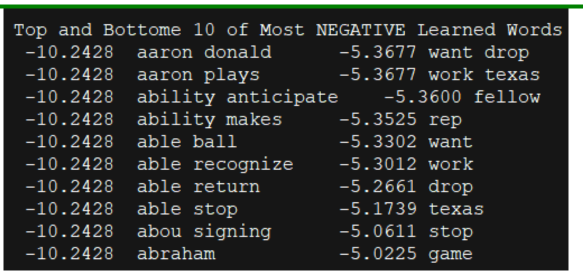
- Overall, LinearSVMs performed better than MNB models.

3.1.1.4.2 MNB Bigram Results

Most POSITIVE WORDS	Most NEGATIVE WORDS

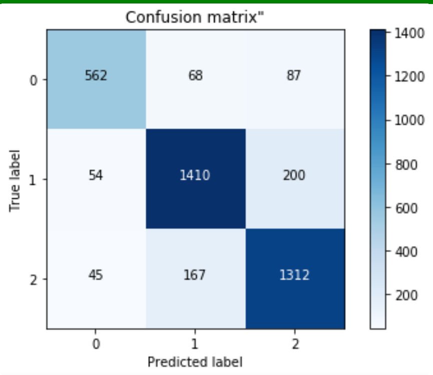


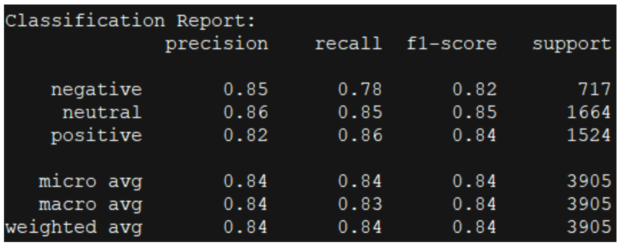
```
Top and Bottome 10 of Most POSITIVE Learned Words
 -10.8442  aaron wilson          -5.4237 patrick
 -10.8442  abbott                -5.3889 amp
 -10.8442  abbott paralyzed      -5.3719 face
 -10.8442  able recognize        -5.3430 mahomes
 -10.8442  abraham               -5.3069 win
 -10.8442  abraham mascot        -5.2913 play
 -10.8442  absolutely            -5.2166 qbs
 -10.8442  absolutely blamed     -5.2130 defense
 -10.8442  abuser                -4.8208 patriots
 -10.8442  abuser took           -4.7327 october
```



```
Top and Bottome 10 of Most NEGATIVE Learned Words
 -10.2428  aaron donald        -5.3677 want drop
 -10.2428  aaron plays         -5.3677 work texas
 -10.2428  ability anticipate     -5.3600 fellow
 -10.2428  ability makes       -5.3525 rep
 -10.2428  able ball           -5.3302 want
 -10.2428  able recognize      -5.3012 work
 -10.2428  able return         -5.2661 drop
 -10.2428  able stop           -5.1739 texas
 -10.2428  abou signing        -5.0611 stop
 -10.2428  abraham             -5.0225 game
```

MNB Bigram Model Prediction Accuracy Results: 84%

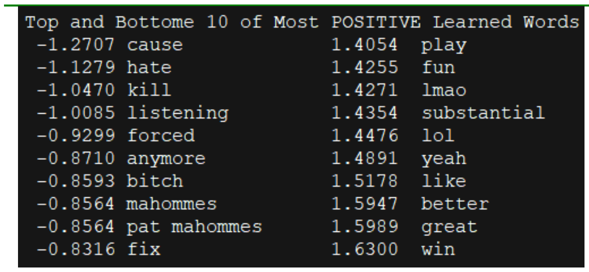




```
Classification Report:
              precision    recall  f1-score   support

    negative       0.85      0.78      0.82       717
     neutral       0.86      0.85      0.85      1664
    positive       0.82      0.86      0.84      1524

   micro avg       0.84      0.84      0.84      3905
   macro avg       0.84      0.83      0.84      3905
weighted avg       0.84      0.84      0.84      3905
```

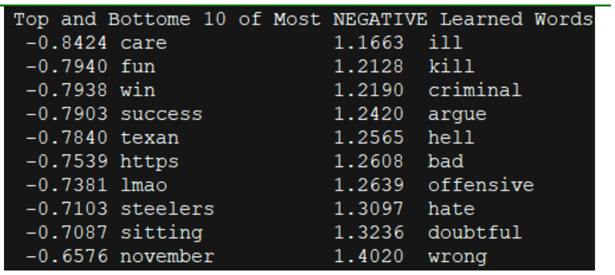
3.1.1.4.3 SVM Bigram Results

Most POSITIVE WORDS



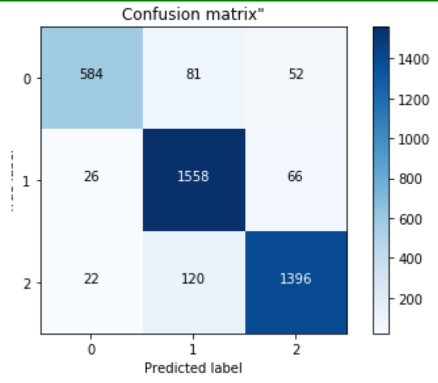
```
Top and Bottome 10 of Most POSITIVE Learned Words
 -1.2707 cause                1.4054  play
 -1.1279 hate                 1.4255  fun
 -1.0470 kill                 1.4271  lmao
 -1.0085 listening            1.4354  substantial
 -0.9299 forced               1.4476  lol
 -0.8710 anymore              1.4891  yeah
 -0.8593 bitch                1.5178  like
 -0.8564 mahommes             1.5947  better
 -0.8564 pat mahommes         1.5989  great
 -0.8316 fix                  1.6300  win
```

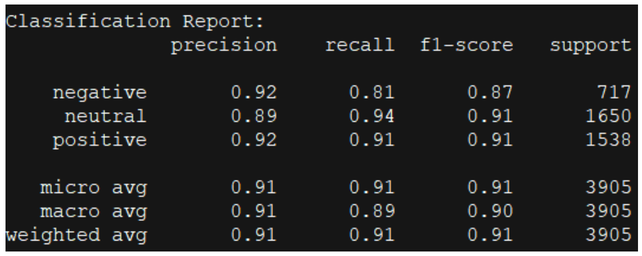
Most NEGATIVE WORDS



```
Top and Bottome 10 of Most NEGATIVE Learned Words
 -0.8424 care                  1.1663  ill
 -0.7940 fun                   1.2128  kill
 -0.7938 win                   1.2190  criminal
 -0.7903 success               1.2420  argue
 -0.7840 texan                 1.2565  hell
 -0.7539 https                 1.2608  bad
 -0.7381 lmao                  1.2639  offensive
 -0.7103 steelers              1.3097  hate
 -0.7087 sitting               1.3236  doubtful
 -0.6576 november              1.4020  wrong
```

SVM Bigram Model Prediction Accuracy Results: 91%




```
Classification Report:
              precision    recall  f1-score   support

    negative       0.92      0.81      0.87       717
     neutral       0.89      0.94      0.91      1650
    positive       0.92      0.91      0.91      1538

   micro avg       0.91      0.91      0.91      3905
   macro avg       0.91      0.89      0.90      3905
weighted avg       0.91      0.91      0.91      3905
```

3.1.1.5 Task 3

- Build the best model (SVM Unigram) by tuning parameters using 80/20 training data set
- Report parameters used to train the model and its cross-validation accuracy

The best SVM vectorizer is the Unigram.

To determine the best LinearSVM hyperparameters to configure the model with GridSearchCV was used. GridSearchCV is a parameter estimation module using grid search with cross-validation.
Hyper-parameters are parameters that are not directly learnt within estimators. In scikit-learn they are passed as arguments to the constructor of the estimator classes.

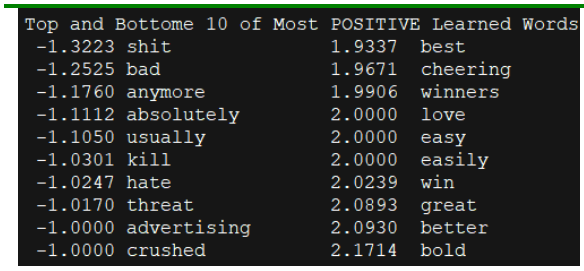
GridSearchCV was ran with the following parameter ranges:
{
'C': [0.1, 1, 10, 100, 1000],
 'loss': ['hinge','squared_hinge'],
 }

- Fitting 3 folds for each of 10 candidates, totalling 30 fits
- grid.best_params_: {'C': 1, 'loss': 'hinge'}

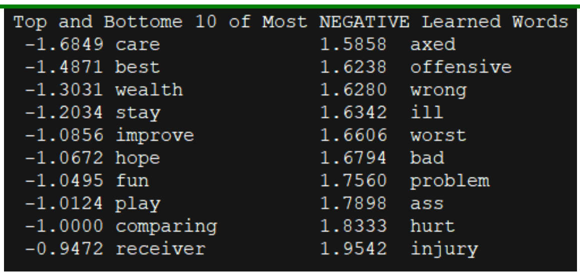
GridSearchCV(cv='warn', error_score='raise-deprecating', estimator=LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True, intercept_scaling=1, loss='squared_hinge', max_iter=1000, multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,verbose=0),fit_params=None, iid='warn', n_jobs=None, param_grid={'C': [0.1, 1, 10, 100, 1000], 'loss': ['hinge', 'squared_hinge']}, pre_dispatch='2*n_jobs', refit=True, return_train_score='warn', scoring=None, verbose=3)

Most POSITIVE WORDS	Most NEGATIVE WORDS

Final Project

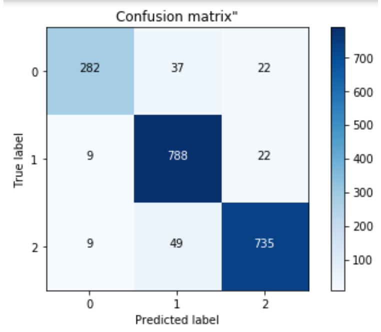


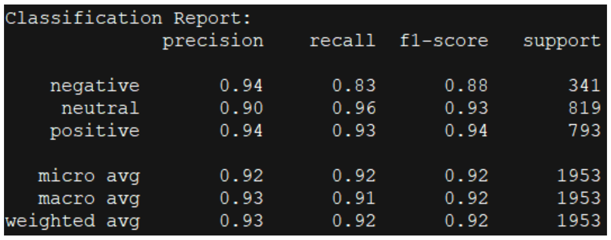
```
Top and Bottome 10 of Most POSITIVE Learned Words
 -1.3223 shit                    1.9337  best
 -1.2525 bad                     1.9671  cheering
 -1.1760 anymore                 1.9906  winners
 -1.1112 absolutely              2.0000  love
 -1.1050 usually                 2.0000  easy
 -1.0301 kill                    2.0000  easily
 -1.0247 hate                    2.0239  win
 -1.0170 threat                  2.0893  great
 -1.0000 advertising             2.0930  better
 -1.0000 crushed                 2.1714  bold
```



```
Top and Bottome 10 of Most NEGATIVE Learned Words
 -1.6849 care                    1.5858  axed
 -1.4871 best                    1.6238  offensive
 -1.3031 wealth                  1.6280  wrong
 -1.2034 stay                    1.6342  ill
 -1.0856 improve                 1.6606  worst
 -1.0672 hope                    1.6794  bad
 -1.0495 fun                     1.7560  problem
 -1.0124 play                    1.7898  ass
 -1.0000 comparing               1.8333  hurt
 -0.9472 receiver                1.9542  injury
```

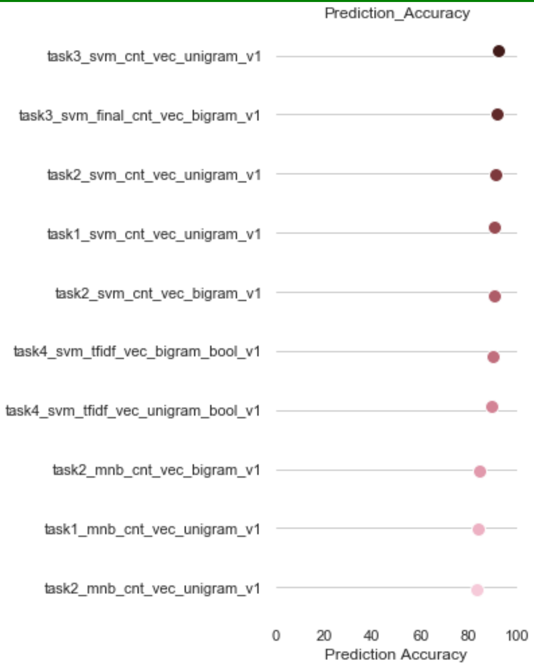
SVM Unigram Model Prediction Accuracy Results: 93%

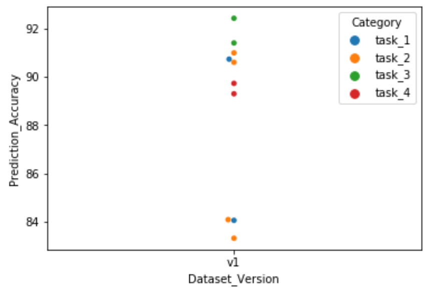


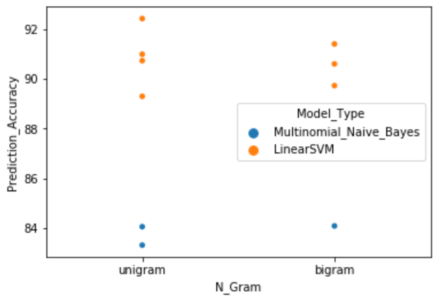


```
Classification Report:
             precision    recall  f1-score   support

    negative      0.94      0.83      0.88       341
     neutral      0.90      0.96      0.93       819
    positive      0.94      0.93      0.94       793

   micro avg      0.92      0.92      0.92      1953
   macro avg      0.93      0.91      0.92      1953
weighted avg      0.93      0.92      0.92      1953
```

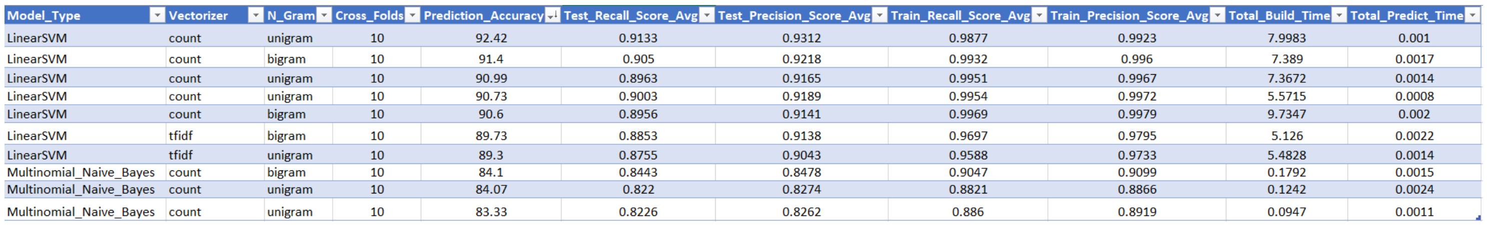
3.1.1.5.1 Model Accuracy Comparison Summary







Final Project



Model_Type	Vectorizer	N_Gram	Cross_Folds	Prediction_Accuracy	Test_Recall_Score_Avg	Test_Precision_Score_Avg	Train_Recall_Score_Avg	Train_Precision_Score_Avg	Total_Build_Time	Total_Predict_Time
LinearSVM	count	unigram	10	92.42	0.9133	0.9312	0.9877	0.9923	7.9983	0.001
LinearSVM	count	bigram	10	91.4	0.905	0.9218	0.9932	0.996	7.389	0.0017
LinearSVM	count	unigram	10	90.99	0.8963	0.9165	0.9951	0.9967	7.3672	0.0014
LinearSVM	count	unigram	10	90.73	0.9003	0.9189	0.9954	0.9972	5.5715	0.0008
LinearSVM	count	bigram	10	90.6	0.8956	0.9141	0.9969	0.9979	9.7347	0.002
LinearSVM	tfidf	bigram	10	89.73	0.8853	0.9138	0.9697	0.9795	5.126	0.0022
LinearSVM	tfidf	unigram	10	89.3	0.8755	0.9043	0.9588	0.9733	5.4828	0.0014
Multinomial_Naive_Bayes	count	bigram	10	84.1	0.8443	0.8478	0.9047	0.9099	0.1792	0.0015
Multinomial_Naive_Bayes	count	unigram	10	84.07	0.822	0.8274	0.8821	0.8866	0.1242	0.0024
Multinomial_Naive_Bayes	count	unigram	10	83.33	0.8226	0.8262	0.886	0.8919	0.0947	0.0011

4 Topic Modeling

4.1 Analysis and Models

4.1.1 *Data Transformation and Cleaning*

Four datasets were used in the topic modeling analysis. Each set is a collection of tweets containing timestamps, user, favorites, and the tweet. All columns except the tweet were removed. The tweets in each file could be categorized by teams, players, coaches, and a complete collection of all of the tweets. The tweets were cleaned by removing Unicode characters (emojis). These symbols (which will be discussed in a future section) are essential in the model's ability to determine concise topics. When this determination was made, the characters were returned to the datasets. There were a total of 8 datasets used in the analysis.

4.1.2 *Models*

Multiple models have created to fine-tune the parameters. Each dataset was vectorized using CountVectorizer (sklearn) and models were generated with LatentDirichletAllocation (sklearn). Stop words were removed as well as any estranged single characters. Initially, symbols representing emojis were removed, but after examing many results, models including emojis were created. The models with these characters displayed better topics that were easier to discern than those generated without emojis.

4.1.2.1 Model Details

Vectorizing parameters were kept constant for all approaches (files vs. tweets) to be modeled. Initial LDA models generated 10 topics, but examing these results showed that there wasn't a concise set of topics being identified. These results had many duplicate word frequency results. Future model iterations cut topics down. The final value of 5 was settled upon because the models generated had topics that could be discerned and the run time to generate the models was acceptable. Attempts at generating less than 5 topics had long runtimes and often had to be terminated.

4.1.2.2 Model Parameters

Count Vectorizer - files	
Content	filename
Analyzer	word
Atop words	english
Lowercase	TRUE
Max df	0.95
Min df	2
Max features	500

Count Vectorizer - content	
Content	content
Analyzer	word
Atop words	english
Lowercase	TRUE
Max df	0.95
Min df	2
Max features	1000

LDA - All	
topics	5
iterations	10
learn	50
random	0

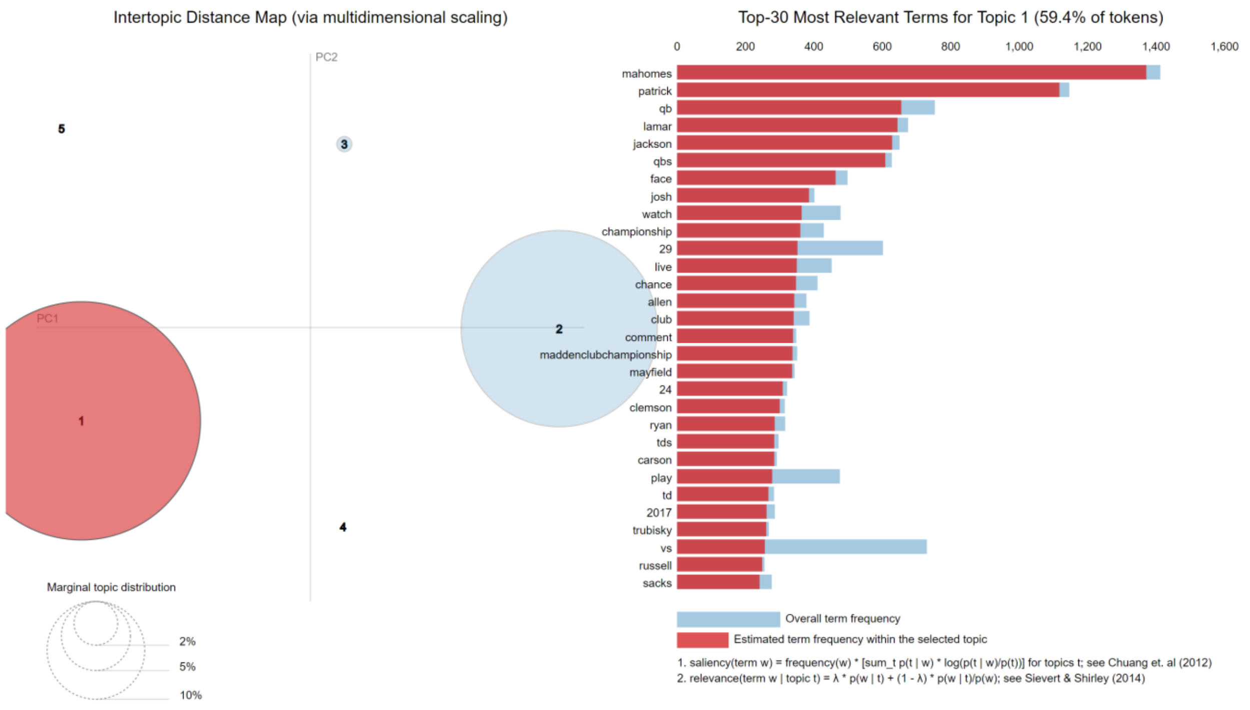
4.1.2.3 Model Results

For both approaches, 2 distinct topics were generated: Young Quarterbacks, Sunday Night Matchup: Patriots at Houston.

The first topic can be discerned with the most relevant terms: Mahomes, Patrick, qb, Lamar, Jackson, Mayfiel d, Russel, Carson, Ryan, Josh, Tribisky. These are all names (or portions of names) of young quarterbacks in t he NFL.

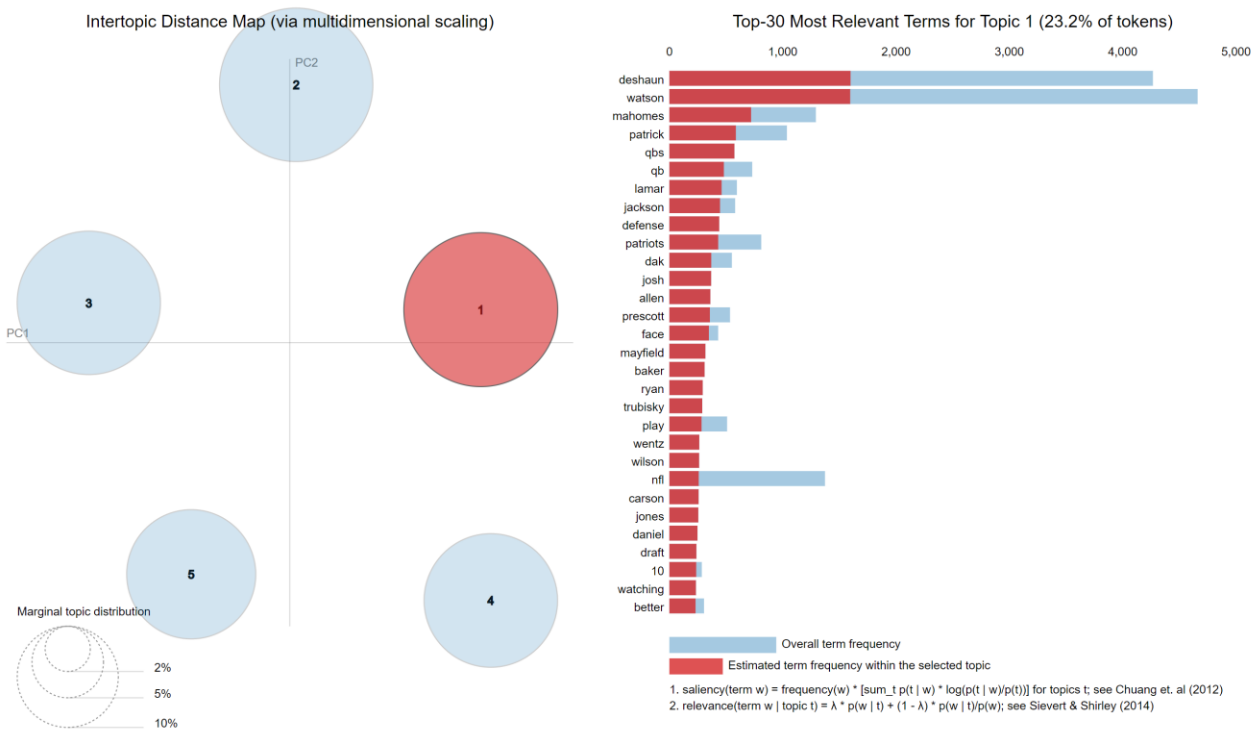
The second topic was the Sunday night game for week 13, New England at Houston: Deshaun, Watson, Hous ton, Patriots, Tom, Brady, Home.

4.1.2.4 Topic Models – Files



Final Project

4.1.2.5 Topic Models – Individual Tweets



5 Player Stats Outcome Prediction

5.1 Analysis and Models

5.1.1 *Data Transformation and Cleaning*

Data used was previously cleaned for other experiments however, the data was not labeled for prediction.

5.1.1.1 Labeling Twitter Data

Fantasy points over the course of the season can be found in several places. ESPN's data was chosen for data labeling purposes. (Scoring Leaders, n.d.) Labels are derived from the week to week change in fantasy points scored.

Table 1 Labels Generated from Fantasy Performance

Label	Game to Game Change in Fantasy Points
dd	Down 10 or more points
d	Down less than 10 points
e	Even, no game to game change
u	Up less than 10 points
uu	Up 10 or more points

Tweet texts are labeled based on their 'created_at' timestamp. A tweet created after the start of a game will be given the point change label based on the next game. For instance, a tweet about Deshaun Watson created after October 6, 2019 at 12:00 PM CDT, the start time of Watson's game against the Atlanta Falcons but before October 13, 2019 12:00 PM CDT, the start time of the game against the Kansas City Chiefs is labeled 'dd' as Watson's fantasy points against Atlanta were 41.7 and against Kansas City. 29.4, a difference of 12.3 fantasy points.

As there is no game to game change to reference at the start of the season (preseason is intentionally omitted), any tweet before the start of the first game are omitted. For the purpose of training and testing classifiers, tweets happening after the most recent game are also omitted however an operationalized model would necessarily perform prediction based on the latter tweets.

Simplified labels are also explored. The simplified set eliminates the 'dd' and 'uu' labels by merging them with the 'd' and 'u' labels respectively. This gives the following breakdown of simplified labels.

Table 2 Simplified Labels for Game to Game Fantasy Point Performance

Label	Game to Game Change in Fantasy Points
d	Fewer points scored

e	Even, no game to game change
u	More points scored

5.1.1.2 Labeling Sentiment

Sentiment data is provided from previous experiments using equivalent timestamps to the original tweet data. Therefore, the same labeling strategy can be applied to label the sentiment data.

5.1.1.3 Preprocessing Tweets

Previous experience with the Scikit-learn LinearSVC support vector machine classifier has shown better performance can be achieved with relative term frequencies rather than raw term frequencies when processing sparse document matrices for classification. Tweet text is vectorized using the Scikit-learn TfidfVectorizer. (Pedregosa, et al., 2011) The basic usage of this vectorizer creates a sparse document matrix of the relative frequency each term is used in a document weighted by the base 10 log of the inverse of the frequency of documents in which it appears. As mentioned, only the relative frequency is needed so the inverse document frequency is not used (use_idf=False). Other notable settings include 'latin-1' encoding (for emoji and other special characters), use of frequency instead of binary vectors, minimum document frequency of one and 'english' stop words.

5.1.2 *Predicting Game to Game Performance Using Multinomial Naïve Bayes*

An attempt is made to predict game to game performance from vectorized tweet text using the Multinomial Naïve Bayes model.

5.1.2.1 Model Details

As already mentioned, twitter data leading up to any one particular game is given the same label based on player performance in that game. The desire is to train a model based on these labels then predict some future week's performance by labeling tweets leading up to that game with the majority label as the "winner" and thus the performance prediction for that week.
This is a bit of a nonstandard approach in that the test data will always be completely unbalanced, it will always have one and only one label indicative of that week's performance.

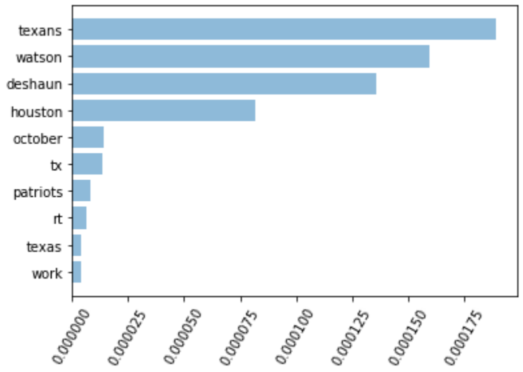
5.1.2.2 Model Parameters

Standard parameters are used. Vectorization has already been noted, and the alpha parameter for the model is not adjusted. Two different datasets are attempted: data leading up to the December 1 game between the Houston Texans and New England Patriots as an "up" week, and the November 17 game between the Houston Texans and Baltimore Ravens as the "down" game, both based on the fantasy performance of Deshaun Watson.

5.1.2.3 Model Results

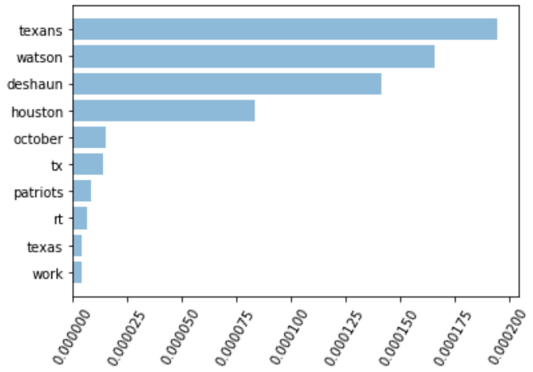
5.1.2.3.1 "Up" Scoring

The week to week scoring "up" experiment was not successful. Of 428 tweets collected for the period between the December 1 game and the previous game, 319 were classified as predicting a down week while only 109 correctly predicted an up week, an accuracy of only 25% and a weighted F1 average of 41%. The desired outcome is to have more than 50% of tweets correctly classified as being part of an “up” week. The following features were deemed important for the “up” classification.



5.1.2.3.2 “Down” Scoring

Possibly shining some light on the failures of the “up” week are the “successes” of the “down” week and its determining features.



In contrast to the up week, the down week using somewhat less data came in with an accuracy of 92% and weight F1 average of 96% identifying 369 tweets as indicative of a “down” week and 33 “up”. This suggests the multinomial naïve bayes classifier may be ineffective on this dataset likely due to the imbalance favoring “down” weeks.+

5.1.3 *Predicting Game to Game Performance using Linear Support Vector Machine*

5.1.3.1 Model Details

A linear kernel support vector machine is utilized in an attempt to better predict outcome based on raw tweet data. Two attempts are made. First, a simple fit fo the vectored text is attempted for an up week and a down week as done with the multinomial naïve bayes already described. A second attempt is then made first doing a grid search over a 10-fold cross validation looking for the best C parameter then utilizing that parameter in another 10-fold cross validation. The best predictor (by F1 score) is selected and the accuracy is gauged on the performance of that predictor on the test week.

5.1.3.2 Model Parameters

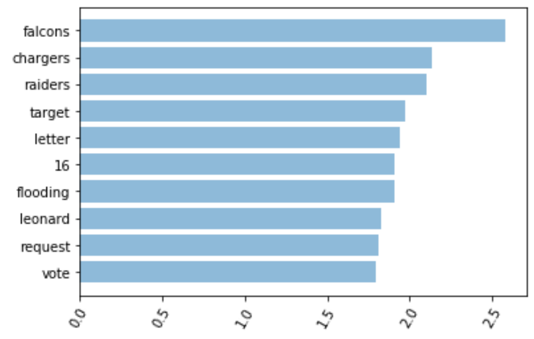
This has already been previously stated. For the simple model, a value of C=1 (default) is used. For the cross validation step in the "up" scoring, the grid search still suggests C=1 is the better value however for the "down", the grid search has zeroed in on a C value of 0.25 which will be used in the cross validation step.

5.1.3.3 Model Results

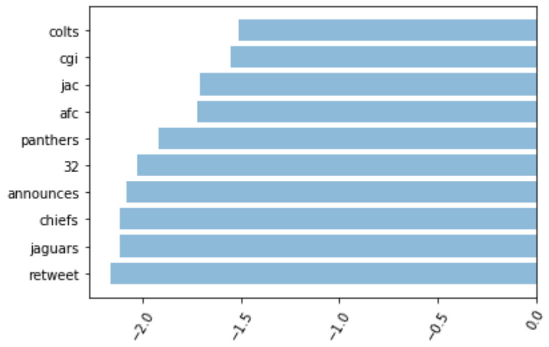
5.1.3.3.1 "Up" Scoring

The simple LinearSVC model is more disappointing than the MultinomialNB model. Correctly classifying only 51 of the 428 tweets, this model comes in at only 12% accuracy with a weighted average F1 of 21%. The following are the most and least influential features.

Most Influential Features



Least Influential Features

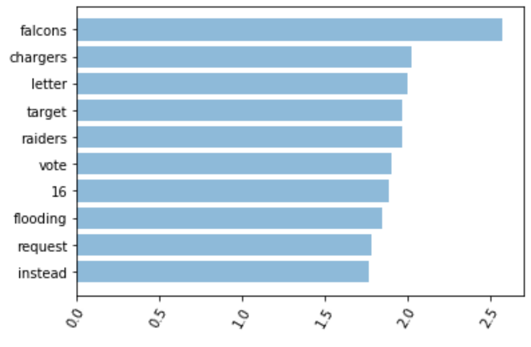


Cross validation performed slightly better correctly classifying 94 tweets for an accuracy of 22% and weighted F1 average of 36%. This is compared to a F1 score from the cross validation of 87%. This suggests the tweet data in and of itself may simply be a poor indicator of performance.

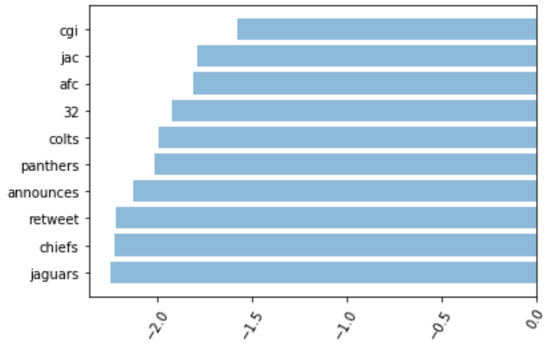
5.1.3.3.2 "Down" Scoring

Again, predicting the down week is far more successful! The cross validation estimator correctly classified 370 of 402 tweets for an accuracy of 92% and a weighted F1 of 96%. The standard fit and predict still predicted 360 correctly with an accuracy of 90% and an F1 of 94%. Features were very similar.

Most Influential Features



Least Influential Features



5.1.4 *Predicting Game to Game Performance Using Sentiment Scoring*

5.1.4.1 Model Details

LinearSVC is again utilized with the simplest parameters to investigate. The same labeling scheme is applied to tweet sentiment scores which are then used to train linear SVC models from both the “up” week and the “down” week.

5.1.4.2 Model Parameters

No special parameters are utilized here.

5.1.4.3 Model Results

Results are very similar to the results of the term frequency models already described. The model was ineffective at predicting an “up” week, only correctly labeling 45 of 425 tweets for an accuracy of 11% and a F1 of 19%. The “down” week, of course, seems like a success correctly classifying 385 tweets for an accuracy of 96% and a weighted F1 of 98%. Again, this shows an ineffective model that is likely to always choose the direction in which a player most frequently trends.

5.1.5 *Predicting Fantasy Point Scores from Tweet Text*

5.1.5.1 Model Details

A linear regression model is trained on the relative term frequencies over the course of the season attempting to predict the actual fantasy point score of a player (Deshaun Watson) for a given game. The expectation is the two games cannot be successfully predicted; the first due to lack of data, and the second due to a single training label fixing the model on that score. After the second game, the hope is the model will slowly converge on the actual fantasy point score for the player.

5.1.5.2 Model Parameters

No specific parameters are utilized for this model except for those already outlined in the data preprocessing phase.

5.1.5.3 Model Results

The model performed somewhat poorly. The following table illustrates the models progress throughout the current NFL season through the December 1 game.

Date	Actual Score	Average Prediction
September 15	12.9	30.7
September 22	25.8	22.4
September 29	11.6	26.9
October 6	41.7	14.1
October 13	29.4	14.3
October 20	15.5	19.6
October 27	27.8	22.5
November 3	19.7	42.0
November 17	4.0	21.8
November 21	18.9	4.5
December 1	28.9	17.9

The model seems to follow two patterns over time, first a pattern of being too slow to react to flutations in the player's scoring pattern then possibly settling into a pattern of overfitting on the previous week. In any case, this is likely not a highly useful model for sentiment scoring.

5.1.6 *Predicting Fantasy Point Scores from Sentiment Scoring*

5.1.6.1 Model Details

Again, an attempt is made to use linear regression to predict a player's fantasy point score from tweet data however in this case, only the positive, neutral and negative scores produced by VADER scoring tools. (Hutto & Gilbert, 2014)

5.1.6.2 Model Parameters

Again, there are no special parameters to note here.

5.1.6.3 Model Results

Date	Actual Score	Average Prediction
September 15	12.9	30.7
September 22	25.8	22.6
September 29	11.6	23.2
October 6	41.7	21.1
October 13	29.4	24.1
October 20	15.5	25.0
October 27	27.8	23.3
November 3	19.7	24.0
November 17	4.0	24.0
November 21	18.9	23.5
December 1	28.9	23.4

Sentiment scoring provided many instances of very few featured. This likely contributed to the very lackluster results. As with the tweet text model, an initial predition of 30.7 is given which is expected. From there one notes the model quickly queros in on a range from 23-24 points. The lack or prediction in fluctuation of score makes this sadly ineffective.

6 Conclusion

The most time-intensive, challenging, yet most critical aspect of this research is in mining the data. To be able to accurately apply machine learning algorithms to this type of unstructured, continuous-timeseries data for the purposes of trend analysis, mechanisms have to first be in place that can search social media sources like Twitter for relevant tweets on a daily and or historical bases. Setting up these programs should be well thought out and designed such that the data, in its various forms, is thought of as the final product. How the data is stored, cleaned and propagated through its pipeline will determine the quality of analysis and experimentation that can be done on it.

Technologies such as VADER (Hutto & Gilbert, 2014) for sentiment scoring, the Natural Language Toolkit (NLTK) (Bird, Loper, & Klein, 2009) for natural language processing and SciKit-Learn (Pedregosa, et al., 2011) for general machine learning have helped to commoditize these disciplines. The techniques made available to the average programmer today offer a wide range of options to attempt to extract value from data that is available either freely (albeit sometimes in limited quantities) or at a known subscription fee.

The National Football League (NFL) remains one of the most popular sports in America (Norman, 2018) and Fantasy Football has grown by leaps and bounds. In this age, popularity almost inevitably leads to an explosion of unstructured data across social networks such as Facebook and Twitter. This freely provided user opinion when correctly gathered and curated can provide a wealth of information about how those who follow both the NFL and fantasy sports feel about a particular team, coach, or player at any given week, especially during the active season. This data can be collected, and the sentiment of the fan base can be measured and tracked over time.

It is possible to look at this same user-provided data to determine the subjects about which changes to user sentiment are driven. The addition of topic modeling to this data may be a useful tool for the purposes of marketing. Not only could the NFL determine what the socially vocal fans are discussing, but it can determine the sentiment within those categories. The implications of this may hold well from a fantasy perspective as well, as these topics/sentiments can be used for assessing the cost of adding players to a fantasy team in games that are week to week.

The validity of predicting player performance based on sentiment is, at the moment, inconclusive. While there is undoubtedly immediate benefit from analyzing social media sentiment for the NFL (it can be used to determine marketing strategy), nothing in the presented analysis can conclusively determine that a player’s performance can be obtained from the consciousness of the ‘Twittersphere.’ Future efforts to determine the true validity of using sentiment in the prediction of a players' performance need to include data from each of the fantasy football service providers. Each provider uses different statistical methods and AI to forecast player performance. While this analysis did not show immediate promise based on the limited slice of data used, it is possible that the addition of new data sources and the altering of methods may lead sentiment to be a variable in predictive models.

7 Appendix: References

1. Twitter Developer Portal - Product APIs: https://developer.twitter.com/en/products/products-overview
    a. Twitter Search Tweets Overview: https://developer.twitter.com/en/docs/tweets/search/overview

8 Bibliography

Bird, S., Loper, E., & Klein, E. (2009). *Natural Language Processing with Python.* O'Reilly Media Inc.

ESPN Fantasy Insights with Watson. (n.d.). Retrieved December 5, 2019, from IBM: https://www.ibm.com/sports/fantasy/

Florio, M. (2010, April 5). *Goodell wants NFL to be earning $25 billion per year by 2027.* Retrieved from PFT: https://profootballtalk.nbcsports.com/2010/04/05/goodell-wants-nfl-to-be-earning-25-billion-per-year-by-2027/

Houghton, D. M., Nowlin, E. L., & Walker, D. (2019). From Fantasy to Reality: The Role of Fantasy Sports in Sports Betting and Online Gambling. *Journal of Public Policy & Marketing, 38*(3), 332-353.

Hutto, C. J., & Gilbert, E. E. (2014). VADER: A Parsimonious Rule-based Model for Sentiment Analysis of Social Media Text. *Eighth International Conference on Weblogs and Social Media (ICWSM-14)*, *June 2014.* Ann Arbor, MI.

Norman, J. (2018, January 4). *Football Still Americans' Favorite Sport to Watch.* Retrieved December 5, 2019, from Gallup: https://news.gallup.com/poll/224864/football-americans-favorite-sport-watch.aspx

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research, 12*, 2825-2830.

Scoring Leaders. (n.d.). Retrieved December 4, 2019, from ESPN: https://fantasy.espn.com/football/leaders?lineupSlot=0%2C1%2C2%2C3%2C4%2C5%2C6%2C7%2C8%2C9%2C10%2C11%2C12%2C13%2C14%2C15%2C16%2C17%2C18%2C19%2C23%2C24&proTeamId=34

Soshnick, S., & Novy-Williams, E. (2019, January 28). *NFL Is Bullish on Its $25 Billion Revenue Goal Ahead of Super Bowl.* Retrieved from Bloomberg: https://www.bloomberg.com/news/articles/2019-01-28/nfl-bullish-about-25-billion-revenue-goal-as-super-bowl-nears