**Ryan Timbrook**

**David Madsen**

**Diego Vales**

**Course:** IST 736 Text Mining

**Term:** Fall, 2019

Table of Contents

[1 Introduction 3](#_Toc26446243)

[1.1 Purpose 3](#_Toc26446244)

[1.2 Scope 3](#_Toc26446245)

[2 Initial Data Mining 4](#_Toc26446246)

[2.1 About the Data 4](#_Toc26446247)

[2.1.1 Dataset Info 4](#_Toc26446248)

[2.1.2 Data Exploration & Cleaning 4](#_Toc26446249)

[3 Sentiment Classification Modeling 7](#_Toc26446250)

[3.1 Analysis and Models 7](#_Toc26446251)

[NFL Teams - 7](#_Toc26446252)

[3.1.1 Data Transformation and Cleaning 7](#_Toc26446253)

[3.1.2 Models 7](#_Toc26446254)

[4 Topic Modeling 8](#_Toc26446255)

[4.1 Analysis and Models 8](#_Toc26446256)

[NFL Teams - 8](#_Toc26446257)

[4.1.1 Data Transformation and Cleaning 8](#_Toc26446258)

[4.1.2 Models 8](#_Toc26446259)

[5 Player Stats Outcome Prediction 9](#_Toc26446260)

[5.1 Analysis and Models 9](#_Toc26446261)

[NFL Teams - 9](#_Toc26446262)

[5.1.1 Data Transformation and Cleaning 9](#_Toc26446263)

[5.1.2 Models 9](#_Toc26446264)

[6 Conclusion 10](#_Toc26446265)

[7 Appendix: References 11](#_Toc26446266)

# 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 to determine if sentiment opinion alone is a predictor of Player Fantacy Football performance stats.

An example would be, if Deshaun Watson's Fantacy Football stat forecast for week-6 is 85 pts leading into that weeks 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 Fantacy Football an edge in choosing to either keep Deshaun Watson as his/her starting quarterback or bench him for someone else.

## 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.

## 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 timeseries 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 determin model prediction accuracies.
* Perform unsuppervised topic modeling ML techniques on document text to gain insights into public opinion primary opinion drivers.
* Evaluate weekly sentimnent trends aligning with Fantascy Football performance stat forecasts.

# Initial Data Engineering

## 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. In order 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.

### Search Tweets Features Used

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Category** | **Product name** | **Supported history** | **Query capability** | **Counts endpoint** | **Data fidelity** |
| **Standard** | [Standard Search API](https://developer.twitter.com/en/docs/tweets/search/overview/standard) | 7 days | [Standard operators](https://developer.twitter.com/en/docs/tweets/search/guides/standard-operators) | Not available | Incomplete |
| **Premium** | [Search Tweets: 30-day endpoint](https://developer.twitter.com/en/docs/tweets/search/overview/premium) | 30 days | [Premium operators](https://developer.twitter.com/en/docs/tweets/search/guides/premium-operators) | Available | Full |
| **Premium** | [Search Tweets: Full-archive endpoint](https://developer.twitter.com/en/docs/tweets/search/overview/premium) | Tweets from as early as 2006 | [Premium operators](https://developer.twitter.com/en/docs/tweets/search/guides/premium-operators) | Available | Full |

### 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. Tweets that are geo-tagged 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 gender‍‍‍ ‍‍and skin t… https://t.co/MkGjXf9aXm",

"user": {},

"entities": {}

}

\*\*Tweet Data Dictionary can be found [here](https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/tweet-object).

### Premium Search Basic Requirements

The following is needed:

* [A developer account](https://developer.twitter.com/en/dashboard)
* [A registered app](https://developer.twitter.com/en/apps)
* [A developer environment setup](https://developer.twitter.com/en/account/environments)
* Authentication - for cURL need a [bearer token](https://developer.twitter.com/en/docs/basics/authentication/guides/bearer-tokens), for Twurl need to have [Twurl setup](https://developer.twitter.com/en/docs/tutorials/using-twurl.html)

### 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*](https://developer.twitter.com/en/docs/tweets/search/guides/premium-operators.html)*.*

Details on the data endpoint response payload that is returned from the API search request can be found [here](https://developer.twitter.com/en/docs/tweets/search/quick-start/premium-30-day).

#### Search Endpoints

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

#### Search Parameters

* query:
  + These were the individual search terms used based on nfl type being processed
    - Team: "houston texans"
    - Coach: "bill obrian"
    - 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

##### Search Date Range Endpoint Mapping

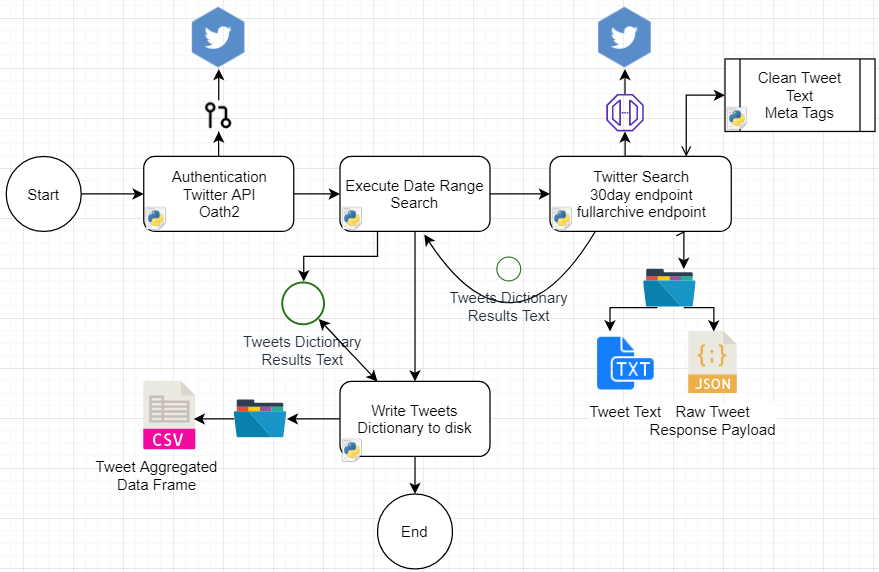
For each NFL type, team, coach, and player, individual search requests were performed specific to a week date range which represents the NFL game schedule. For this, different search API endpoints were utilized based on their search time frame capabilities and rate limit restrictions.

|  |  |  |  |
| --- | --- | --- | --- |
| NFL Weekly Schedule | From Date - To Date | Search Endoint | 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 |  |

### 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



#### Data Cleaning

At this stage, tweet text was cleaned of hashtags, urls, and @tags. Each of these types were 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 dataframe 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.

#### 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

#### 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.

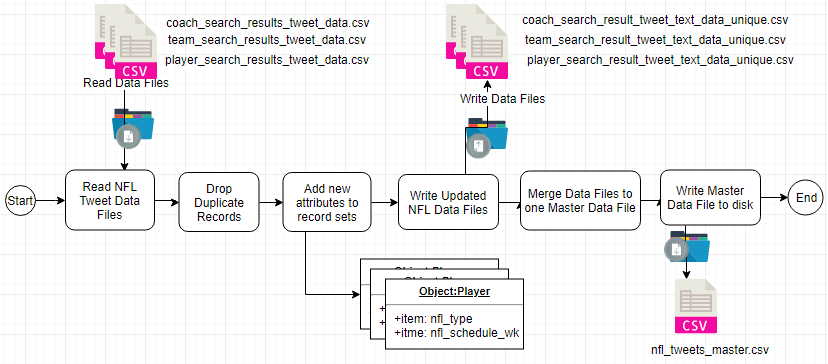
Output OS Path Patterns: outputPath = f'{dataDir}/{nfl\_type}/{search\_on}/v{search\_iteration}/{search\_range}'

* tweet\_filename=f'{outputPath}/tweet\_text.txt'
* raw\_filename=f'{outputPath}/tweet\_raw.txt'

### 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 additonal 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 grainular analysis. The other field is a numeric categorical attribute that identifies the nfl schedule week the tweets are associated to from a timeline perspective.



#### Data Output

Four data files are generated as output from this proces.

* 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

### VEDAR Sentiment Classification

Jupyter Notebook: **classify\_train\_nfl\_master.ipynb**

python package: vaderSentiment.vaderSentiment **SentimentIntensityAnalyzer**

In order to create a labeled data set for our suppervised sentiment classification algorathms, 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.

#### VEDAR Details

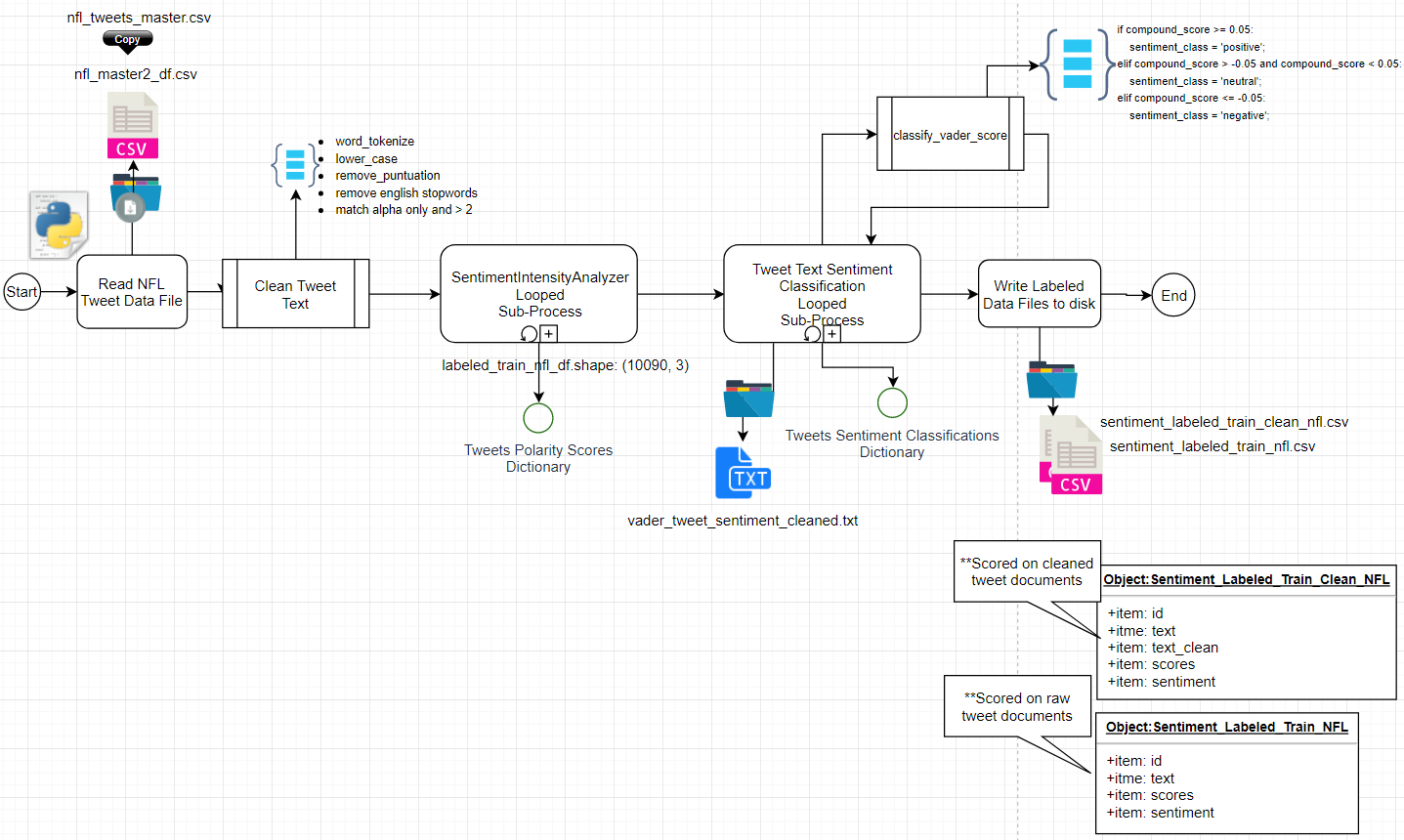
Behind Vader's scoring is its core sentiment analysis engine, vaderSentiment.py.

"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 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>

#### VEDAR Implementation



#### VEDAR Results

Table 2.2: Vedar Polarity Scores

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

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

|  |
| --- |
|  |

### Merge NFL Labeled Data Set to Master File

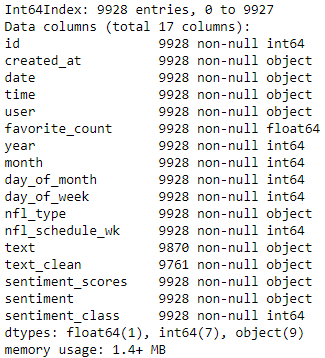
Jupyter Notebook: **nfl\_sentiment\_analysis\_data\_merge\_master**.ipynb

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 executed in section 2.1.6. and outputs a merged data file to be used for modeling, visualizations and analysis.

#### Implementation

Data Output:

* nfl\_master\_sent\_merged\_timeseries.csv



# Sentiment Classification Modeling

This section evaluates two types of classification algorithms with multiple hyperperameter 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 fair comparison, the same 60% training, 40% testing split is maintained. Likewise, all vectorization parametes 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.

## Analysis and Models

### Data Transformation and Cleaning

### Models

Jupyter Notebook: **classification\_modeling\_mnb\_svm.ipynb**

##### 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 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. 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.

The data for these vectorization steps is the NFL twitter data described in section. Each of the vectorization steps below were for Multinomial Naive Baise and Support Vector Machine Classification modeling comparision.

##### 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'   **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' | **Train Data Set Split Configurations:**   * train\_test\_split: test\_size=0.4   **Figure 2.4: Task 2 Label Distribution** |

##### 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** |

#### 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 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 a 80/20 split inorder 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

##### 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 taking for each of the CountVectorizer vector models in the given task.

##### Support Vector Machine (SVM) Models

##### *Linear Support Vector Classification (LinearSVC)*

Python package scikit-learn v0.21.3 [sklearn.svm.LinearSVC](https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html#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](https://scikit-learn.org/stable/modules/svm.html#svm-classification).

**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 classifier, "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

#### Task 1

* Build unigram models
* Print top 10 indicative words for 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%.

##### MNB Unigram Results

|  |  |
| --- | --- |
| Most POSITIVE WORDS | Most NEGATIVE WORDS |
|  |  |

##### MNB Model Prediction Accuracy Results: 84%

|  |  |
| --- | --- |
|  |  |

##### SVM Unigram Results

|  |  |
| --- | --- |
| Most POSITIVE WORDS | Most NEGATIVE WORDS |
|  |  |

##### SVM Model Prediction Accuracy Results: 91%

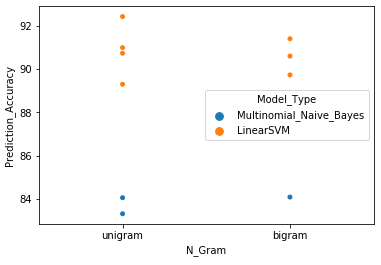
|  |  |
| --- | --- |
|  |  |

#### Task 2

* Build bigram models
* Print top 10 indicative words for most positive category and the most negative category
* Report confusion matrix, precision, and recall scores.
* Compare evaluation measures, was adding bigrams helpful?

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

##### Model Comparison Prediction Accuracies



* Overall, LinearSVMs performed better than MNB models.

##### MNB Bigram Results

|  |  |
| --- | --- |
| Most POSITIVE WORDS | Most NEGATIVE WORDS |
|  |  |

##### MNB Bigram Model Prediction Accuracy Results: 84%

|  |  |
| --- | --- |
|  |  |

##### SVM Bigram Results

|  |  |
| --- | --- |
| Most POSITIVE WORDS | Most NEGATIVE WORDS |
|  |  |

##### SVM Bigram Model Prediction Accuracy Results: 91%

|  |  |
| --- | --- |
|  |  |

#### Task 3

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

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)

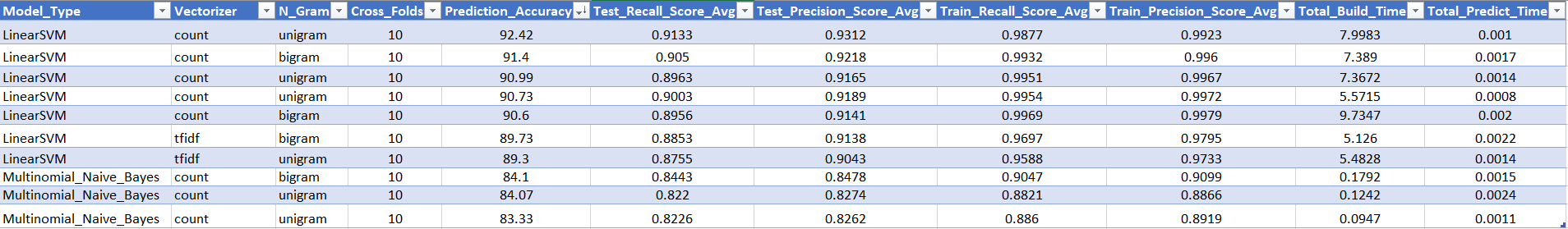
|  |  |
| --- | --- |
| Most POSITIVE WORDS | Most NEGATIVE WORDS |
|  |  |

##### SVM Unigram Model Prediction Accuracy Results: 93%

|  |  |
| --- | --- |
|  |  |

##### Model Accuracy Comparison Summary

|  |  |
| --- | --- |
|  |  |



# Topic Modeling

## Analysis and Models

### Data Transformation and Cleaning

### Models

#### Model Details

#### Model Parameters

#### Model Results

# Player Stats Outcome Prediction

## Analysis and Models

### NFL Teams -

### Data Transformation and Cleaning

### Models

#### Model Details

#### Model Parameters

#### Model Results

# Conclusion

# Appendix: References

1. Twitter Developer Portal - Product APIs: <https://developer.twitter.com/en/products/products-overview>
   1. Twitter Search Tweets Overview: <https://developer.twitter.com/en/docs/tweets/search/overview>
2. VEDAR Sentiment Intensity Analyzer - <https://github.com/cjhutto/vaderSentiment>
3. [Using VADER to handle sentiment analysis with social media text](http://t-redactyl.io/blog/2017/04/using-vader-to-handle-sentiment-analysis-with-social-media-text.html)