2019-1002 IST 736 Text Mining

Homework Assignment 6 (week 6)

Ryan Timbrook
NetID: RTIMBROO
Course: IST 736 Text Mining
Term: Fall, 2019
Topic: Use Benoulli and Multinomial Naive Bayes algorithm to build models to classify text documents of customer reviews

Homework Assignment 6 (week 6)

Table of Contents

1 Introduction

Every business needs to keep a pulse on how they are perceived in the public eye. Today, more than ever, due to the social media phenomenon explosion along with the expansion of technology being at everyone's fingertips it's critical for any organization to have a solid understanding of customers they are reaching or trying to reach think positively about them or not. Along with that, if not as important or more important is can we distinguish what's true or not; If what people or anything capable of posting a comment or review about an organization online is authentic.

According to Moz. The marketing firm found that one negative article can lose a company as many as 22 percent of its customers. Just four such articles can drive off 70 percent of potential customers — something any business would struggle to bounce back from.

Bad reviews are bad news for brands, large or small. The one thing companies can do is to listen to their customers. Since companies typically have large amounts of customers, any of whom could be prone to posting comments on the internet, they need to use sophisticated technology to narrow the playing field and find potential negative customers or false statements being written about them on-line before the damage is too great to fix.

1.1 Purpose

Build supervised learning models using both Benoulli and Multinominal Naive Bayes algorithm that classify text documentation of customer reviews by sentiment and authenticity.

1.2 Scope

Given a labeled data set of customer reviews, perform two classification tasks useing both Benoulli and Multinominal Naive Bayes to build the models. Compare the performance results of these models.

- Perform two classification tasks
  - Sentiment (positive or negative)
  - Authenticity (real or fake)
- For each classification task, build Benoulli Naive Bayes and Multinominal Naive Bayes, using 10 fold cross validation, models for each of the following variations:
  - CountVectorizer
  - TfidfVectorizer
  - Unigram
  - Bigram
  - Dataset Tokenization Configurations:
    - V1:
      - lower_case=False, stop_words=False, remove_punc=False, remove_non_alphabetic=True, stemming=False
    - V2:
      - lower_case=True, stop_words=False, remove_punc=False, remove_non_alphabetic=True, stemming=False
    - V3:
      - lower_case=True, stop_words=True, remove_punc=True, remove_non_alphabetic=True, stemming=False

        - V4:
            - lower_case=True, stop_words=True, remove_punc=True, remove_non_alphabetic=True, stemming=True
    - Report on the top performing model variation

2 Analysis and Models

2.1 About the Data

The deception data set is made up of 92 individual text documents of customer reviews. Each document is labled with two classifications, lie and sentiment. Each of the classification lables are binary. For the lie label, an 'f' stands for false, the customer review is not a lie, and a 't' stands for true, the customer review is a lie. The sentiment label 'n' stands for negative, the customer review has a negative tone, and the 'p' for postive, the costomer has a postive tone in their review.

2.1.1 *Dataset Info*

The initial shape of the deception data set is (92, 3). Where it has 92 rows, and 3 columns. It's initial size is 276. Figure 2.1 below is a small representation of how the data looks when first loaded for exploration.

Figure 2.1: Deception Data Set sample:

lie	sentiment	review
f	n	'i really like this buffet restaurant in Marshall street. they have a lot of selection of american
f	n	'OMG. This restaurant is horrible. The receptionist did not greet us
t	n	'Restaurant : Samrat Food Ordered : Dal Tadka
f	p	'WQR is the best! A group of us went out last Friday to celebrate a friend\'s birthday and we had a blast. Great ambiance
f	p	'Gannonâ€™s Isle Ice Cream served the best ice cream and you better believe it! The place is ideally situated and it is easy to get too. The ice cream is delicious
t	p	'Can\'t say too much about it. Just

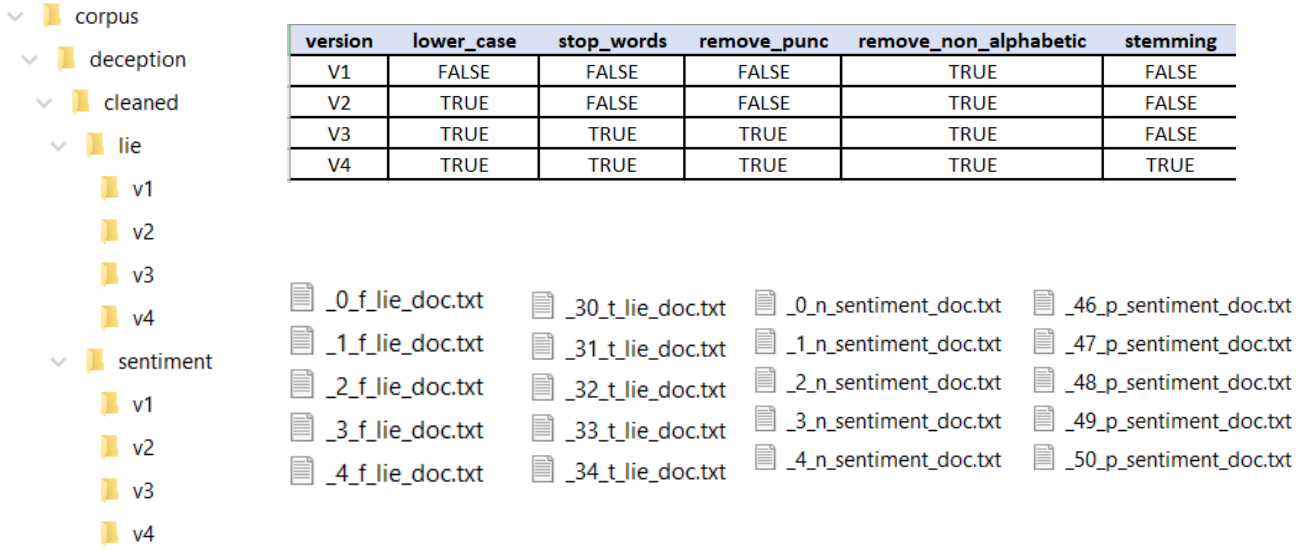
2.1.2 *Data Exploration & Cleaning*

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

Focusing on the goal of vectorizing customer review text as individual documents to be used as a corpus for analyzing customer's sentiment and authenticity toward restaurants, the following text preparation pipeline steps were taken:

- Split the initial customer review data set into two datasets for classification, lie detection and sentiment analysis.
  - Both datasets started with 92 records and 2 columns
  - Two records were removed from both sets where it was found that the review was a single '?' character.
- For both datasets, tokenize each document
  - The **nltk word_tokenize** class was used for this step
- For both datasets, perform Bag Of Words exploration
  - Both before cleaning and after cleaning of document word tokens
- Fore both datasets, 4 versions of cleaned data are created for model evaluation. Each version is stored as it's own corpus. Images below represent the corpus directory structure generated during the tokenization cleaning and preprocessing steps.

Homework Assignment 6 (week 6)



version	lower_case	stop_words	remove_punc	remove_non_alphabetic	stemming
V1	FALSE	FALSE	FALSE	TRUE	FALSE
V2	TRUE	FALSE	FALSE	TRUE	FALSE
V3	TRUE	TRUE	TRUE	TRUE	FALSE
V4	TRUE	TRUE	TRUE	TRUE	TRUE

- *Note: Due to the small sample size, additional trial runs, taking into account all of the below steps, were taken by resampling the original dataset at 1,000 with replacement and 10,000 with replacement. The only noticible observed effect on the model prediction outcomes was that they became more normalized at 50/50.*

This section will cover the cleaning and vectorization steps taken as pre-processing methods for the classification models to be discussed in section 3.

2.1.2.1 Cleaning Steps Taken:

For both lie and sentiment classification, the following tokenization / corpus generation steps were taken:

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

(note - each of the bellow steps is controlled via a Boolean True or False conditional statement, based on the below configuration, four versions of the document corpus were create and modeled on.)

Table 2.1: Corpus Tokenization Configurations

version	lower_case	stop_words	remove_punc	remove_non_alphabetic	stemming
V1	FALSE	FALSE	FALSE	TRUE	FALSE
V2	TRUE	FALSE	FALSE	TRUE	FALSE
V3	TRUE	TRUE	TRUE	TRUE	FALSE
V4	TRUE	TRUE	TRUE	TRUE	TRUE

Where specified in Table 2.1:

- Punctuation was removed using the python string. punctuation values
  i. '!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
- Non-Alphabetic tokens were removed using the python string method isalpha()
- Lowercase all of the token characters
- Stop words were removed using the NLTK English stopwords list
  i. additionally, this step allows the addition of custom stop words to be added to the list for fine-tuning.

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

2.1.2.1.1 Initial Tokenization - Lie Review Text

Using the nltk word_tokenize class, each line of the individual retaurant review text document was tokenized. Figure 2.2 is a Word Cloud of all of the tokenized words for the Lie classification dataset. Figure's 2.3 and 2.4 are representations of just those observations that were classified as either True or False.

From this viewpoint it's hard to recognize if any meaningful words that could descern the classification stand out.

Figure 2.2: Lie BoW Word Cloud

Figure 2.3: Lie True BoW Word Cloud

Figure 2.4: Lie False BoW Word Cloud

2.1.2.1.2 Vectorization Preprocessing Steps - Lie Review Text

After runing the four tokenization methods specified above in section 2.1.2.1, the following feature counts remained for each scenario:

- Total Feature Count Prior to Vectorization Preprocessing: **8152**
- Total Feature Counts per Version After Vectorization Preprocessing:

version	category	label	feature count	unique feature count
V1	lie	t	3362	1021
V1	lie	f	3629	1035
V2	lie	t	3362	950
V2	lie	f	3629	944
V3	lie	t	1648	832
V3	lie	f	1789	831
V4	lie	t	1648	733
V4	lie	f	1789	734

Post-pre-processing: Each cleaned text was saved to its own file to be used as a corpus of documents in the vectorization process. The file name has the following format: <_{docIndex}_{t_Or_f}_lie_doc.txt. The corpus directory path for the lie documents is: ./corpus/deception/cleaned/lie/v{number}/ - see section 2.1.2 for details.

Below in figure's 2.5 and 2.6 are representations of the Lie reviews after tokenization cleaning's been performed.

From this dataset, nothing substantial is jumping out that would indicate flags for lieing or not.

Figure 2.5: Cleaned Lie 'True' Feature BoW Reviews

Figure 2.6: Cleaned Lie 'False' Feature BoW Reviews

2.1.2.1.3 Initial Tokenization - Sentiment Review Text

All of the above steps taken for Lie review text tokenization were also followed for the sentiment review documents.

- Initial Sentiment Review Token Count: **8079**
- Feature's that are part of a document labeled as positive: **3352**
- Features that are part of a document labeled as negative: **4742**

Figure 2.7: Sent BoW Word Cloud

Figure 2.8: Sent Pos BoW Word Cloud

Figure 2.9: Sent Neg BoW Word Cloud

2.1.2.1.4 Vectorization Preprocessing Steps - Sentiment Review Text

All of the above steps taken for Lie review text vectorization preprocessing were also followed for the sentiment review documents.

- Total Feature Count Prior to Vectorization Preprocessing: **8079**
- Total Feature Counts per Version After Vectorization Preprocessing:

version	category	label	feature count	unique feature count
V1	sentiment	p	2749	850
V1	sentiment	n	3941	1050
V2	sentiment	p	2749	789
V2	sentiment	n	3941	966
V3	sentiment	p	1468	724
V3	sentiment	n	1993	896
V4	sentiment	p	1468	644
V4	sentiment	n	1993	799

Post-pre-processing: Each cleaned text was saved to its own file to be used as a corpus of documents in the vectorization process. The file name has the following format: <_{docIndex}_{p_Or_n}_sentiment_doc.txt. The corpus directory path for the lie documents is: ./corpus/deception/cleaned/sentiment/v{number}/ -- see section 2.1.2 for details.

3 Models

3.1.1 *CountVectorizer - Vectorize Team Text Documents*

Utilizing the python package **sklearn.feature_extraction.text CountVectorizer** class, this model converts a collection of 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.

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

- if data set is Lie:
    - path=f'{corpusDir}/deception/cleaned/lie/v{num}
    - A collection of 90 documents are retrieved from each version of the cleaned corpus containing customer review text labeled for authentisity
- if dataset is Sentiment:
    - path=f'{corpusDir}/deception/cleaned/sentiment/v{num}
    - A collection of 92 documents are retrieved containing customer review text labeled for sentiment analysis

For this experiment unigram and bigram models were vectorized as frequency count for Multinomial Naive Baise modeling and binary for Benoulli Naive Baise and saved to file for analysis and future additive modeling. This was done for both the Lie and Sentiment document corpuses. Classification modeling was conducted for each of these vectors to assess which had the best classification results. Each vector can be found in the ./output directory for reference.

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

3.1.1.1 CountVectorizer Unigram Model

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

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

3.1.1.1.1 CountVectorizer Unigram Details

CountVectorizer Unigram Parameters for Multinomial NB:

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

Both fit and transform methods were performed on this model.

CountVectorizer Unigram Parameters for Benoulli NB:

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

Both fit and transform methods were performed on this model.

3.1.1.2 CountVectorizer Bigram Model

3.1.1.2.1 CountVectorizer Bigram Details

CountVectorizer Bigram Parameters for Multinomial NB:

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

Both fit and transform methods were performed on this model.

CountVectorizer Bigram Parameters for Benoulli NB:

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

3.1.1.2.2 CountVectorizer Results

Vector Document Term Matrix Dimensions

Vector	Corpus	Size	Rows	Columns(vocabulary)	Saved File Name
lie_cnt_vec_unigram_v1	lie	3803	90	590	./output/v1/lie_cnt_vec_unigram_v1.txt
lie_cnt_vec_unigram_v2	lie	3803	90	590	./output/v2/lie_cnt_vec_unigram_v2.txt
lie_cnt_vec_unigram_v3	lie	2141	90	482	./output/v3/lie_cnt_vec_unigram_v3.txt
lie_cnt_vec_unigram_v4	lie	2253	90	464	./output/v4/lie_cnt_vec_unigram_v4.txt
lie_cnt_vec_unigram_bool_v1	lie	3803	90	590	./output/v1/lie_cnt_vec_unigram_bool_v1.txt
lie_cnt_vec_unigram_bool_v2	lie	3803	90	590	./output/v2/lie_cnt_vec_unigram_bool_v2.txt
lie_cnt_vec_unigram_bool_v3	lie	2141	90	482	./output/v3/lie_cnt_vec_unigram_bool_v3.txt
lie_cnt_vec_unigram_bool_v4	lie	2253	90	464	./output/v4/lie_cnt_vec_unigram_bool_v4.txt
lie_cnt_vec_bigram_v1	lie	5907	90	1240	./output/v1/lie_cnt_vec_bigram_v1.txt
lie_cnt_vec_bigram_v2	lie	5907	90	1240	./output/v2/lie_cnt_vec_bigram_v2.txt
lie_cnt_vec_bigram_v3	lie	2519	90	646	./output/v3/lie_cnt_vec_bigram_v3.txt
lie_cnt_vec_bigram_v4	lie	2687	90	651	./output/v4/lie_cnt_vec_bigram_v4.txt
lie_cnt_vec_bigram_bool_v1	lie	5907	90	1240	./output/v1/lie_cnt_vec_bigram_bool_v1.txt
lie_cnt_vec_bigram_bool_v2	lie	5907	90	1240	./output/v2/lie_cnt_vec_bigram_bool_v2.txt
lie_cnt_vec_bigram_bool_v3	lie	2519	90	646	./output/v3/lie_cnt_vec_bigram_bool_v3.txt
lie_cnt_vec_bigram_bool_v4	lie	2687	90	651	./output/v4/lie_cnt_vec_bigram_bool_v4.txt
sent_cnt_vec_unigram_v1	sentiment	3702	90	574	./output/v1/sent_cnt_vec_unigram_v1.txt
sent_cnt_vec_unigram_v2	sentiment	3702	90	574	./output/v2/sent_cnt_vec_unigram_v2.txt
sent_cnt_vec_unigram_v3	sentiment	2161	90	486	./output/v3/sent_cnt_vec_unigram_v3.txt
sent_cnt_vec_unigram_v4	sentiment	2271	90	468	./output/v4/sent_cnt_vec_unigram_v4.txt
sent_cnt_vec_unigram_bool_v1	sentiment	3702	90	574	./output/v1/sent_cnt_vec_unigram_bool_v1.txt
sent_cnt_vec_unigram_bool_v2	sentiment	3702	90	574	./output/v2/sent_cnt_vec_unigram_bool_v2.txt
sent_cnt_vec_unigram_bool_v3	sentiment	2161	90	486	./output/v3/sent_cnt_vec_unigram_bool_v3.txt
sent_cnt_vec_unigram_bool_v4	sentiment	2271	90	468	./output/v4/sent_cnt_vec_unigram_bool_v4.txt
sent_cnt_vec_bigram_v1	sentiment	5741	90	1208	./output/v1/sent_cnt_vec_bigram_v1.txt
sent_cnt_vec_bigram_v2	sentiment	5741	90	1208	./output/v2/sent_cnt_vec_bigram_v2.txt
sent_cnt_vec_bigram_v3	sentiment	2536	90	650	./output/v3/sent_cnt_vec_bigram_v3.txt
sent_cnt_vec_bigram_v4	sentiment	2702	90	655	./output/v4/sent_cnt_vec_bigram_v4.txt
sent_cnt_vec_bigram_bool_v1	sentiment	5741	90	1208	./output/v1/ sent_cnt_vec_bigram_bool_v1.txt
sent_cnt_vec_bigram_bool_v2	sentiment	2536	90	650	./output/v2/ sent_cnt_vec_bigram_bool_v2.txt
sent_cnt_vec_bigram_bool_v3	sentiment	2702	90	655	./output/v3/ sent_cnt_vec_bigram_bool_v3.txt
sent_cnt_vec_bigram_bool_v4	sentiment	3702	90	574	./output/v4/ sent_cnt_vec_bigram_bool_v4.txt

3.1.2 *TfidfVectorizer - Vectorize Team Text Documents*

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

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

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

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

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

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

- if data set is Lie:
  - path=f'{corpusDir}/deception/cleaned/lie/v{num}/
  - A collection of 90 documents are retrieved containing customer review text labeled for authentisity
- if dataset is Sentiment:
  - path=f'{corpusDir}/deception/cleaned/sentiment/v{num}/
  - A collection of 90 documents are retrieved containing customer review text labeled for sentiment analysis

For this experiment unigram and bigram models were vectorized as frequency count for Multinomial Naive Baise modeling and binary for Benoulli Naive Baise and saved to file for analysis and future additive modeling. This was done for both the Lie and Sentiment document corpuses. Classification modeling was conducted for each of these vectors to assess which had the best classification results. Each vector can be found in the ./output directory for reference.

3.1.2.1 TfidfVectorizer Unigram Model

3.1.2.1.1 TfidfVectorizer Unigram Details

TfidfVectorizer Unigram Parameters for Multinomial NB:

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

Both fit and transform methods were performed on this model.

TfidfVectorizer Unigram Parameters for Benoulli NB:

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

Both fit and transform methods were performed on this model.

3.1.2.2 TfidfVectorizer Bigram Model

3.1.2.2.1 TfidfVectorizer Bigram Details

TfidfVectorizer Bigram Parameters for Multinomial NB:

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

Both fit and transform methods were performed on this model.

TfidfVectorizer Bigram Parameters for Benoulli NB:

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

Both fit and transform methods were performed on this model.

3.1.2.2.2 TfidfVectorizer Results

Vector Document Term Matrix Dimensions

Vector	Corpus	Size	Rows	Columns(vocabulary)	Saved File Name
lie_tfidf_vec_unigram_v1	lie	3803	90	590	./output/v1/lie_tfidf_vec_unigram_v1.txt
lie_tfidf_vec_unigram_v2	lie	3803	90	590	./output/v2/lie_tfidf_vec_unigram_v2.txt
lie_tfidf_vec_unigram_v3	lie	2141	90	482	./output/v3/lie_tfidf_vec_unigram_v3.txt
lie_tfidf_vec_unigram_v4	lie	2253	90	464	./output/v4/lie_tfidf_vec_unigram_v4.txt
lie_tfidf_vec_unigram_bool_v1	lie	3803	90	590	./output/v1/lie_tfidf_vec_unigram_bool_v1.txt
lie_tfidf_vec_unigram_bool_v2	lie	3803	90	590	./output/v2/lie_tfidf_vec_unigram_bool_v2.txt
lie_tfidf_vec_unigram_bool_v3	lie	2141	90	482	./output/v3/lie_tfidf_vec_unigram_bool_v3.txt
lie_tfidf_vec_unigram_bool_v4	lie	2253	90	464	./output/v4/lie_tfidf_vec_unigram_bool_v4.txt
lie_tfidf_vec_bigram_v1	lie	5907	90	1240	./output/v1/lie_tfidf_vec_bigram_v1.txt
lie_tfidf_vec_bigram_v2	lie	5907	90	1240	./output/v2/lie_tfidf_vec_bigram_v2.txt
lie_tfidf_vec_bigram_v3	lie	2519	90	646	./output/v3/lie_tfidf_vec_bigram_v3.txt
lie_tfidf_vec_bigram_v4	lie	2687	90	651	./output/v4/lie_tfidf_vec_bigram_v4.txt
lie_tfidf_vec_bigram_bool_v1	lie	5907	90	1240	./output/v1/lie_tfidf_vec_bigram_bool_v1.txt
lie_tfidf_vec_bigram_bool_v2	lie	5907	90	1240	./output/v2/lie_tfidf_vec_bigram_bool_v2.txt
lie_tfidf_vec_bigram_bool_v3	lie	2519	90	646	./output/v3/lie_tfidf_vec_bigram_bool_v3.txt
lie_tfidf_vec_bigram_bool_v4	lie	2687	90	651	./output/v4/lie_tfidf_vec_bigram_bool_v4.txt
sent_tfidf_vec_unigram_v1	sentiment	3702	90	574	./output/v1/sent_tfidf_vec_unigram_v1.txt
sent_tfidf_vec_unigram_v2	sentiment	3702	90	574	./output/v2/sent_tfidf_vec_unigram_v2.txt
sent_tfidf_vec_unigram_v3	sentiment	2161	90	486	./output/v3/sent_tfidf_vec_unigram_v3.txt
sent_tfidf_vec_unigram_v4	sentiment	2271	90	468	./output/v4/sent_tfidf_vec_unigram_v4.txt
sent_tfidf_vec_unigram_bool_v1	sentiment	3702	90	574	./output/v1/sent_tfidf_vec_unigram_bool_v1.txt

sent_tfidf_vec_unigram_bool_v2	sentiment	3702	90	574	./output/v2/sent_tfidf_vec_unigram_bool_v2.txt
sent_tfidf_vec_unigram_bool_v3	sentiment	2161	90	486	./output/v3/sent_tfidf_vec_unigram_bool_v3.txt
sent_tfidf_vec_unigram_bool_v4	sentiment	2271	90	468	./output/v4/sent_tfidf_vec_unigram_bool_v4.txt
sent_tfidf_vec_bigram_v1	sentiment	5741	90	1208	./output/v1/sent_tfidf_vec_bigram_v1.txt
sent_tfidf_vec_bigram_v2	sentiment	5741	90	1208	./output/v2/sent_tfidf_vec_bigram_v2.txt
sent_tfidf_vec_bigram_v3	sentiment	2536	90	650	./output/v3/sent_tfidf_vec_bigram_v3.txt
sent_tfidf_vec_bigram_v4	sentiment	2702	90	655	./output/v4/sent_tfidf_vec_bigram_v4.txt
sent_tfidf_vec_bigram_bool_v1	sentiment	5741	90	1208	./output/v1/ sent_tfidf_vec_bigram_bool_v1.txt
sent_tfidf_vec_bigram_bool_v2	sentiment	2536	90	650	./output/v2/ sent_tfidf_vec_bigram_bool_v2.txt
sent_tfidf_vec_bigram_bool_v3	sentiment	2702	90	655	./output/v3/ sent_tfidf_vec_bigram_bool_v3.txt
sent_tfidf_vec_bigram_bool_v4	sentiment	3702	90	574	./output/v4/ sent_tfidf_vec_bigram_bool_v4.txt

3.1.3 *Multinominal Naive Bayes Models*

For our text classification problems 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 and TfidfVectorizer vector models created in section 3.1.1 and 3.1.2. For efficiency the vector models were packaged into collection objects to be itterated over in a loop. The loop passes through a function that performs all of the MNB modeling pre-processing and execution tasks. Output results for both Lie and Sentiment modeling are written to a report for evaluation.

3.1.3.1 Data Transformation

3.1.3.1.1 Train Test Split Process

Labels for both datasets, Lie and Sentiment were encoded using the sklearn.preprocessing LabelEncoder class. After running the fit_transform method, these label catgories are transformed into binary, 0 or 1 values.

For prediction evaluation and accuracy measurement, 30% of each dataset was held out as unseen data. The method used was sklearns.model_selection train_test_split class.

3.1.3.2 Build-Test-Validate-Predict MNB 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. 30% of each was held out for final, unseen, prediction accuracy evaluation.

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

3.1.3.3 Top 5 MNB Lie Models Results

3.1.3.3.1 Lie Prediction Accuracy Results

Experiment_Model_Name	Prediction_Accuracy	Total_Build_Time
lie_cnt_vec_unigram_v3	51.85	0.0859
lie_tfidf_vec_unigram_v2	51.85	0.0896
lie_cnt_vec_bigram_v4	51.85	0.0948
lie_cnt_vec_unigram_v1	48.15	0.0780
lie_tfidf_vec_unigram_v3	48.15	0.0893

Averaged Prediction over dataset version:
V4 Avg Prediciton Accuracy: 37.96
V3 Avg Prediciton Accuracy: 43.51
V2 Avg Prediciton Accuracy: 44.44
V1 Avg Prediciton Accuracy: 41.66

3.1.3.4 Top 5 MNB Sentiment Models Results

3.1.3.4.1 Sentiment Prediction Accuracy Results

Experiment_Model_Name	Prediction_Accuracy	Total_Build_Time
sent_cnt_vec_bigram_v2	62.96	0.0795
sent_cnt_vec_bigram_v3	55.56	0.0833
sent_cnt_vec_unigram_v3	51.85	0.1050
sent_tfidf_vec_unigram_v1	51.85	0.0736
sent_cnt_vec_unigram_v2	48.15	0.1152

Averaged Prediction over dataset version:
V4 Avg Prediciton Accuracy: 40.73
V3 Avg Prediciton Accuracy: 46.29
V2 Avg Prediciton Accuracy: 50.0
V1 Avg Prediciton Accuracy: 43.51

It doesn't appear that the corpus version of tokenized text has an overwhelming influence on the model's accuracy results. Further filtering and issolation of the individual attributes of the models will need to be performed to get a more grainular outlook.

The top performing Multinomial NB for sentiment classification was 62.96% and generated from a model with the following configurations:

- CountVectorizer
- Bigram
- **Trained on a corpus generated without stop_words being removed and without removing punctuation.**

version	lower_case	stop_words	remove_punc	remove_non_alphabetic	stemming
V1	FALSE	FALSE	FALSE	TRUE	FALSE
V2	TRUE	FALSE	FALSE	TRUE	FALSE
V3	TRUE	TRUE	TRUE	TRUE	FALSE
V4	TRUE	TRUE	TRUE	TRUE	TRUE

3.1.4 *Bernoulli Naive Bayes Models*

Naive Bayes classifier for multivariate Bernoulli models.

Like MultinomialNB, this classifier is suitable for discrete data. The difference is that while MultinomialNB works with occurrence counts, BernoulliNB is designed for binary/boolean features.

For this implementation we are using scikit-learn v0.21.3 sklearn.naive_bayes BernoulliNB class.
The below steps were taking for each of the CountVectorizer vector models and TfidfVectorizer vector models created in section 3.1.1 and 3.1.2. For efficiency the vector models were packaged into collection objects to be itterated over in a loop. The loop passes through a function that performs all of the BNB modeling pre-processing and execution tasks. Output results for both Lie and Sentiment modeling are written to a report for evaluation.

3.1.4.1 Top 5 BNB Lie Models Results

3.1.4.1.1 Lie Prediction Accuracy Results

Experiment_Model_Name	Prediction_Accuracy	Total_Build_Time
lie_cnt_vec_bigram_bool_v4	66.67	0.0998
lie_tfidf_vec_unigram_bool_v2	55.56	0.1246
lie_cnt_vec_bigram_bool_v1	55.56	0.1185
lie_cnt_vec_unigram_bool_v4	51.85	0.1105
lie_tfidf_vec_bigram_bool_v4	51.85	0.1051

Averaged Prediction over dataset version:
V4 Avg Prediciton Accuracy: 50.0
V3 Avg Prediciton Accuracy: 38.8
V2 Avg Prediciton Accuracy: 43.5
V1 Avg Prediciton Accuracy: 45.3

3.1.4.2 Top 5 BNB Sentiment Models Results

3.1.4.2.1 Sentiment Prediction Accuracy Results

Experiment_Model_Name	Prediction_Accuracy	Total_Build_Time
sent_tfidf_vec_unigram_bool_v4	74.07	0.1146
sent_tfidf_vec_bigram_bool_v3	59.26	0.1804
sent_cnt_vec_unigram_bool_v1	51.85	0.1246
sent_cnt_vec_unigram_bool_v3	51.85	0.1065
sent_cnt_vec_bigram_bool_v2	51.85	0.1012

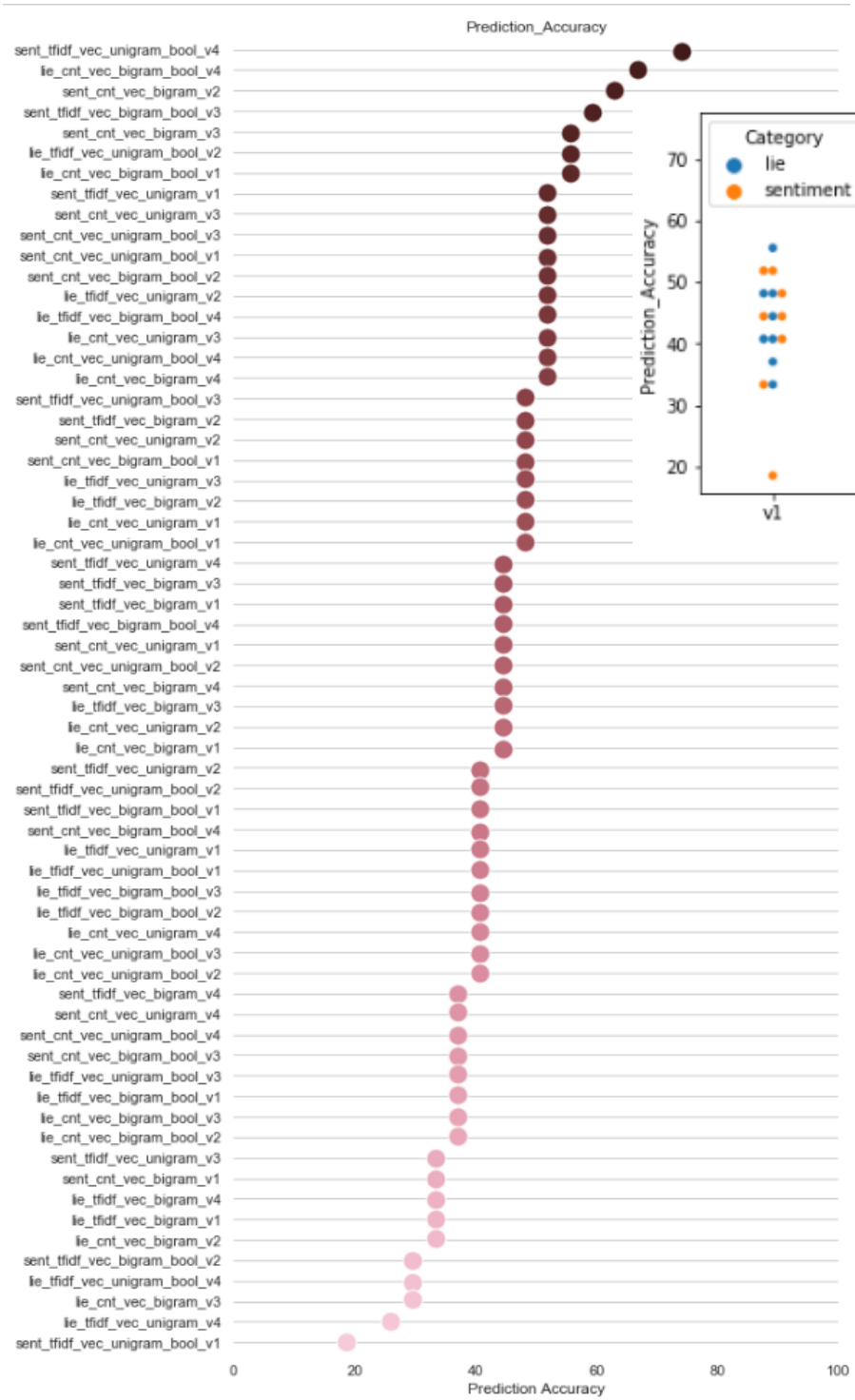
Averaged Prediction over dataset version:
V4 Avg Prediciton Accuracy: 49.07
V3 Avg Prediciton Accuracy: 49.00
V2 Avg Prediciton Accuracy: 41.66
V1 Avg Prediciton Accuracy: 39.81

The best Bernoulli accuracy prediction model for the sentiment classification task was 74.7% and was performed by a model with the following configurations:

- TfidfVectorizer
- Unigram
- Binary
- **Trained on a corpus generated by stemming**

version	lower_case	stop_words	remove_punc	remove_non_alphabetic	stemming
V1	FALSE	FALSE	FALSE	TRUE	FALSE
V2	TRUE	FALSE	FALSE	TRUE	FALSE
V3	TRUE	TRUE	TRUE	TRUE	FALSE
V4	TRUE	TRUE	TRUE	TRUE	TRUE

3.1.5 *Naive Bayes Models Overall Results*



Bernoulli Naive Bayes has the highest overall prediction accuracy score of 74.7% on the sentiment classification task; beating the Multinomial Naive Bayes which had a best score of 63%.

Bernoulli Naive Bayes also had the highest prediction accuracy on the lie detection classification task of 66.7%, winning out over Multinomial Naive Bayes at 52%.

4 Conclusions

Of the 64 models evaluated, eight of them scored above a 55% predicted accuracy rating on unseen customer reviews with a high of 74% on a sentiment classification task. Comparing Multinomail Naive Bayes with Benoulli Naive Bayes, Benoulli had the highest accuracy score in both sentiment and lie detection classification.

Sentiment analysis and lie detection in text data is a very challenging natural language processing task. The samples provided were insufficient for a model to learn how to detect negative versus positive sentiment and if someone was telling the truth or not in what they wrote. More context into how the text data was labeled would aid in further studies using this along with with additional data feeds to build more accurate models.

Additionally, there are labeled datasets that are used as gold standards for building and training models for baseline evaluation of accuracy. Starting with those datasets, then adding-context specific domain knowledge into this dataset would greatly enhance the accuracy potential of these models on customer reviews and authenticity.

5 Appendix: