2019-1002 IST 736 Text Mining

Homework Assignment 4 (week 4)

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

Homework Assignment 4 (week 4)

Table of Contents

Homework Assignment 4 (week 4)

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 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, use Multinominal Naive Bayes to build the models.

- Perform two classification tasks
  - Sentiment (positive or negative)
  - Authenticity (real or fake)
- For each of the models, evaluate them using 10-fold cross validation method.
- For each modles, report the 20 most indicative words that the models have learnt.
- Compare the difficulty level of sentiment classification vs. lie detection.

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 will have 92 records and 2 columns
- 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, clean each documents tokens:
  - Remove punctuation
  - Remove non-alphabetic
  - Lower case text
  - Remove stop words
- *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:

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

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

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

- 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.

Homework Assignment 4 (week 4)

Note: Multiple trial runs were taken for both Lie and Sentiment in changing the above parameters and evaluating its impact on the end states model prediction on unseen data. No significant enhancements to the predictions accuracy were found. For our final trial and report the above cleaning steps were all set to True.

- Total Feature Count Prior to Vectorization Preprocessing: **8152**
- Total Feature Count After Vectorization Preprocessing**: 3439**
- Feature's that are part of a document labeled as true: **1650**
    - True unique features: **833**
- Features that are part of a document labeled as false**: 1789**
    - False unique features: **831**

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.

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 Count After Vectorization Preprocessing**: 3463**

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.

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. Where ever input='filename' for vectorization parameter this collection of files is used.

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

For this experiment unigram, bigram and trigram models were vectorized 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:

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

Both fit and transform methods were performed on this model.

3.1.1.2 CountVectorizer Bigram Model

3.1.1.2.1 CountVectorizer Bigram Details

CountVectorizer Bigram Parameters:

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

Both fit and transform methods were performed on this model.

3.1.1.3 CountVectorizer Trigram Model

3.1.1.3.1 CountVectorizer Trigram Details

CountVectorizer Bigram Parameters:

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

Both fit and transform methods were performed on this model.

3.1.1.3.2 CountVectorizer Results

Vector Document Term Matrix Dimensions

Vector	Corpus	Size	Rows	Columns(vocabulary)	Saved File Name
lie_cnt_vec_unigram	lie	2606	92	1225	./output/lie_cnt_feature_vector_unigram.txt
lie_cnt_vec_bigram	lie	5482	92	3912	./output/lie_cnt_feature_vector_bigram.txt

lie_cnt_vec_trigram	lie	8307	92	6711	./output/lie_cnt_feature_vector_trigram.txt
sent_cnt_vec_unigram	sent	2630	92	1234	./output/sent_cnt_feature_vector_unigram.txt
sent_cnt_vec_bigram	sent	5531	92	3953	./output/sent_cnt_feature_vector_bigram.txt
sent_cnt_vec_trigram	sent	8380	92	6779	./output/sent_cnt_feature_vector_trigram.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
  - A collection of 92 documents are retrieved containing customer review text labeled for authentisity
- if dataset is Sentiment:
  - path=f'{corpusDir}/deception/cleaned/sentiment
  - A collection of 92 documents are retrieved containing customer review text labeled for sentiment analysis

For this experiment unigram, bigram and trigram models were vectorized 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:

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

Both fit and transform methods were performed on this model.

3.1.2.2 TfidfVectorizer Bigram Model

3.1.2.2.1 TfidfVectorizer Bigram Details

TfidfVectorizer Bigram Parameters:

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

Both fit and transform methods were performed on this model.

3.1.2.3 TfidfVectorizer Trigram Model

3.1.2.3.1 TfidfVectorizer Trigram Details

TfidfVectorizer Bigram Parameters:

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

Both fit and transform methods were performed on this model.

3.1.2.3.2 TfidfVectorizer Results

Vector Document Term Matrix Dimensions

Vector	Corpus	Size	Rows	Columns(vocabulary)	Saved File Name
lie_tfidf_vec_unigram	lie	2606	92	1225	.output/lie_tfidf_feature_vector_unigram.txt
lie_tfidf_vec_bigram	lie	5482	92	3912	.output/lie_tfidf_feature_vector_bigram.txt
lie_tfidf_vec_trigram	lie	8307	92	6711	.output/lie_tfidf_feature_vector_trigram.txt

sent_tfidf_vec_unigram	sent	2630	92	1234	.output/sent_tfidf_feature_vector_unigram.txt
sent_tfidf_vec_bigram	sent	5531	92	3953	.output/sent_tfidf_feature_vector_bigram.txt
sent_tfidf_vec_trigram	sent	8380	92	6779	.output/sent_tfidf_feature_vector_trigram.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.

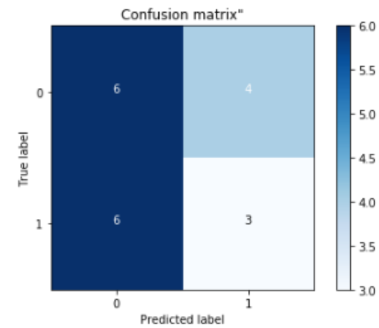
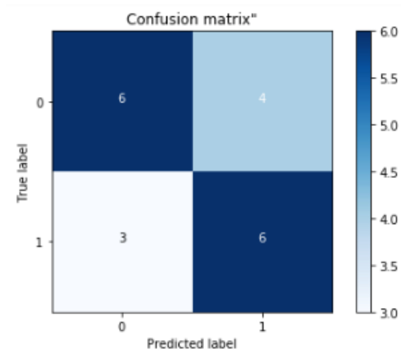
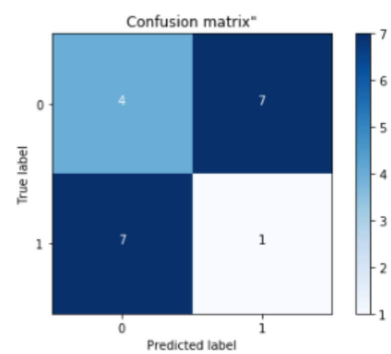
For prediction evaluation and accuracy measurement, 20% of each dataset was held out as unseen data. The method used was sklearns.model_selection train_test_split class.

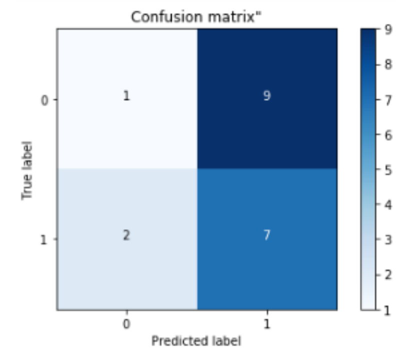
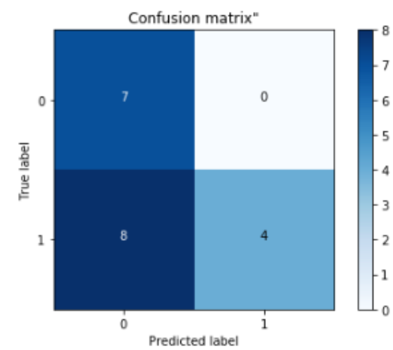
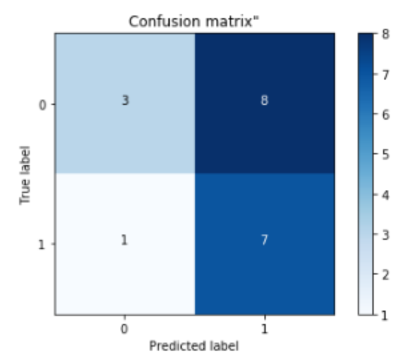
3.1.3.2 Build-Test-Validate-Predict MNB Models

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. 20% of each was held out for final, unseen, prediction accuracy evaluation.

3.1.3.3 MNB Lie Models Results

3.1.3.3.1 Lie Prediction Accuracy Results

Model	Total points to Label	Misslabeled Points	Percent Mislabeled	Percent Accurately Labeled	Confusion Matrix
lie_cnt_unigram	19	10	0.5263	0.4736	
lie_tfidf_unigram	19	7	0.3684	**0.63157**	
lie_cnt_bigram	19	14	0.73684	0.2631	

lie_tfidf_bigram	19	11	0.57894	0.42105	
lie_cnt_trigram	19	8	0.42105	**0.57894**	
lie_tfidf_trigram	19	9	0.47368	0.52631	

3.1.3.3.2 lie_cnt_unigram Model Scoring Details

Build Time: ---------------------------- [0.17730290000054083]
Fit Time:------------------------------- [array([0.01195526, 0.00299144, 0.00498724, 0.0039885 , 0.00398946, 0.00598431, 0.00697517, 0.00498867, 0.00598311, 0.00398755])]
Score Time:----------------------------- [array([0.00498843, 0.00299168, 0.00598621, 0.00598478, 0.00498629, 0.01096988, 0.00498724, 0.00498605, 0.00498676, 0.00399017])]
Test Recall Scores:--------------------- [array([0.125 , 0.625 , 0.375 , 0.75 , 0.375 ,

0.5 , 0.16666667, 0.33333333, 0.5 , 0.83333333])]
Test Precision Scores:------------------ [array([0.1 , 0.78571429, 0.36666667, 0.75 , 0.21428571,
0.5 , 0.1 , 0.2 , 0.5 , 0.875])]
Train Recall Scores:-------------------- [array([1. , 1. , 1. , 1. , 1. ,
1. , 0.98484848, 1. , 1. , 1.])]
Train Precision Scores:----------------- [array([1. , 1. , 1. , 1. , 1. ,
1. , 0.98529412, 1. , 1. , 1.])]

Predict Time:--------------------------- [0.0017864000001281966]

3.1.3.3.3 lie_tfidf_unigram Model Scoring Details

Build Time: ---------------------------- [0.40670009999939793]
Fit Time:------------------------------- [array([0.00798273, 0.00698137, 0.00698209, 0.01694679, 0.00698137,
0.01657081, 0.01695704, 0.0079782 , 0.0199461 , 0.00698209])]
Score Time:----------------------------- [array([0.01395845, 0.00797868, 0.00749469, 0.0089767 , 0.00897527,
0.00797749, 0.01096892, 0.03490543, 0.01395941, 0.00898218])]
Test Recall Scores:--------------------- [array([0.375 , 0.25 , 0.25 , 0.25 , 0.5 ,
0.5 , 0.45833333, 0.33333333, 0.5 , 0.5])]
Test Precision Scores:------------------ [array([0.36666667, 0.16666667, 0.16666667, 0.25 , 0.5 ,
0.5 , 0.45 , 0.2 , 0.5 , 0.25])]
Train Recall Scores:-------------------- [array([1. , 1. , 1. , 1. , 1. ,
1. , 0.98484848, 1. , 1. , 1.])]
Train Precision Scores:----------------- [array([1. , 1. , 1. , 1. , 1. ,
1. , 0.98529412, 1. , 1. , 1.])]

Predict Time:--------------------------- [0.004308300000047893]

3.1.3.3.4 lie_cnt_bigram Model Scoring Details

Build Time: ---------------------------- [0.35490219999974215]
Fit Time:------------------------------- [array([0.01495814, 0.01197028, 0.01296616, 0.01097107, 0.01695466,
0.01238418, 0.01296568, 0.01296473, 0.01496148, 0.01295948])]
Score Time:----------------------------- [array([0.0209434 , 0.00797725, 0.00498652, 0.00598645, 0.00797987,
0.00598454, 0.00797892, 0.00897598, 0.00797725, 0.00598335])]
Test Recall Scores:--------------------- [array([0.75 , 0.375 , 0.5 , 0.375 , 0.375 ,
0.33333333, 0.58333333, 0.29166667, 0.66666667, 0.66666667])]
Test Precision Scores:------------------ [array([0.75 , 0.36666667, 0.5 , 0.36666667, 0.21428571,
0.16666667, 0.58333333, 0.29166667, 0.8 , 0.8])]
Train Recall Scores:-------------------- [array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])]
Train Precision Scores:----------------- [array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])]

Predict Time:--------------------------- [0.0026583999997455976]

3.1.3.3.5 lie_tfidf_bigram Model Scoring Details

Build Time: ---------------------------- [0.5036607000001823]
Fit Time:------------------------------- [array([0.00997782, 0.03390336, 0.0159657 , 0.01296592, 0.01396227,
0.01096606, 0.02693105, 0.01794839, 0.00997138, 0.0089767])]
Score Time:----------------------------- [array([0.01695156, 0.02792645, 0.01894069, 0.00997281, 0.02393794,
0.01496029, 0.01696515, 0.01296782, 0.00798798, 0.00797844])]
Test Recall Scores:--------------------- [array([0.5 , 0.375 , 0.625 , 0.375 , 0.625 ,

0.5 , 0.58333333, 0.5 , 0.66666667, 0.33333333])]
Test Precision Scores:------------------ [array([0.5 , 0.36666667, 0.63333333, 0.36666667, 0.63333333,
0.5 , 0.58333333, 0.5 , 0.66666667, 0.2])]
Train Recall Scores:-------------------- [array([1. , 1. , 1. , 1. , 1. ,
1. , 0.96969697, 1. , 1. , 1.])]
Train Precision Scores:----------------- [array([1. , 1. , 1. , 1. , 1. ,
1. , 0.97142857, 1. , 1. , 1.])]

Predict Time:--------------------------- [0.003743399999621033]

3.1.3.3.6 lie_cnt_trigram Model Scoring Details

Build Time: ---------------------------- [0.7361665000007633]
Fit Time:------------------------------ [array([0.04088783, 0.01695514, 0.02992082, 0.02592635, 0.02493787,
0.02506924, 0.0249331 , 0.01894927, 0.02293825, 0.03590441])]
Score Time:---------------------------- [array([0.01296544, 0.00797844, 0.00897765, 0.02194524, 0.02194047,
0.01296329, 0.00997186, 0.0110321 , 0.00797939, 0.04345536])]
Test Recall Scores:--------------------- [array([0.375 , 0.375 , 0.625 , 0.375 , 0.45833333,
0.29166667, 0.25 , 0.54166667, 0.25 , 0.33333333])]
Test Precision Scores:------------------ [array([0.36666667, 0.36666667, 0.78571429, 0.36666667, 0.45 ,
0.29166667, 0.2 , 0.55 , 0.2 , 0.2])]
Train Recall Scores:-------------------- [array([0.98333333, 0.98333333, 0.98333333, 0.98333333, 0.98387097,
0.98387097, 1. , 0.98387097, 0.98387097, 0.98387097])]
Train Precision Scores:----------------- [array([0.98611111, 0.98611111, 0.98611111, 0.98611111, 0.98611111,
0.98611111, 1. , 0.98611111, 0.98611111, 0.98648649])]

Predict Time:--------------------------- [0.012066700000104902]

3.1.3.3.7 lie_tfidf_trigram Model Scoring Details

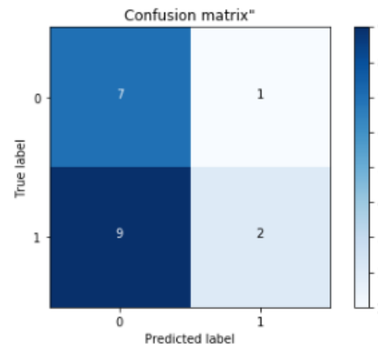
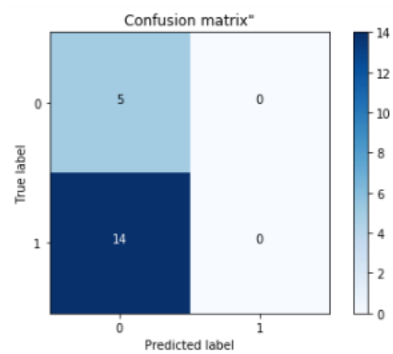
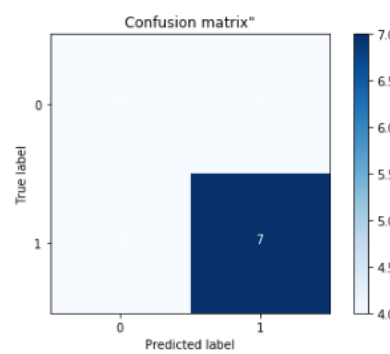
Build Time: ---------------------------- [0.4189678000002459]
Fit Time:------------------------------ [array([0.01516294, 0.00997496, 0.01097107, 0.0169549 , 0.01596236,
0.01296496, 0.01097178, 0.01396298, 0.01507187, 0.01296592])]
Score Time:---------------------------- [array([0.00697994, 0.0079782 , 0.00797772, 0.01296568, 0.01196384,
0.01695561, 0.01097775, 0.01196885, 0.01195526, 0.01201487])]
Test Recall Scores:--------------------- [array([0.25 , 0.5 , 0.25 , 0.625 , 0.625 ,
0.375 , 0.5 , 0.5 , 0.33333333, 0.5])]
Test Precision Scores:------------------ [array([0.25 , 0.5 , 0.16666667, 0.78571429, 0.78571429,
0.25 , 0.28571429, 0.28571429, 0.2 , 0.25])]
Train Recall Scores:-------------------- [array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])]
Train Precision Scores:----------------- [array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])]

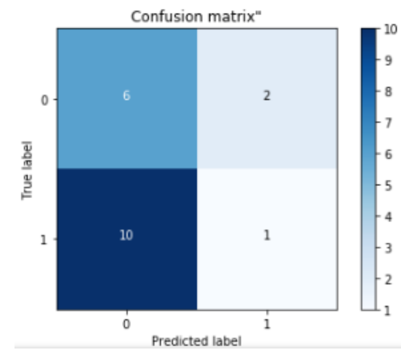
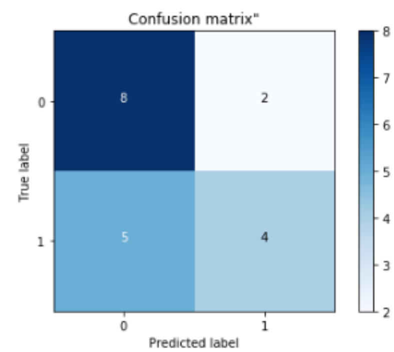
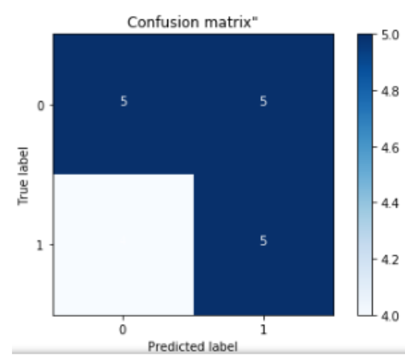
Predict Time:--------------------------- [0.008878900000127032]

3.1.3.4 MNB Sentiment Models Results

3.1.3.4.1 Sentiment Prediction Accuracy Results

Homework Assignment 4 (week 4)

Model	Total points to Label	Misslabeled Points	Percent Mislabeled	Percent Accurately Labeled	Confusion Matrix
sent_cnt_unigram	19	10	0.5263	0.4736	
sent_tfidf_unigram	19	14	0.73684	0.26315	
sent_cnt_bigram	19	8	0.42105	0.57894	 [[4 4] [4 7]]

sent_tfidf_bigram	19	12	0.63157	0.36842	
sent_cnt_trigram	19	7	0.36842	**0.63157**	
sent_tfidf_trigram	19	9	0.47368	0.52631	

3.1.3.4.2 sent_cnt_unigram Model Scoring Details

Build Time: ---------------------------- [0.12292340000021795]
Fit Time:------------------------------- [array([0.00498557, 0.00299191, 0.00299096, 0.00299239, 0.00398946,
0.00398946, 0.00499439, 0.00299287, 0.0039885 , 0.00299144])]
Score Time:----------------------------- [array([0.00498557, 0.00398946, 0.00498676, 0.00498438, 0.00398874,
0.00299168, 0.00497746, 0.00299215, 0.00299287, 0.00398922])]
Test Recall Scores:--------------------- [array([0.5 , 0.375 , 0.5 , 0.375 , 0.5 ,

Homework Assignment 4 (week 4)

0.29166667, 0.83333333, 0.125 , 0.33333333, 0.5])]
Test Precision Scores:------------------ [array([0.5 , 0.36666667, 0.25 , 0.36666667, 0.5 ,
0.29166667, 0.9 , 0.125 , 0.33333333, 0.5])]
Train Recall Scores:-------------------- [array([0.96774194, 0.96774194, 0.98387097, 0.96774194, 0.96774194,
0.96875 , 0.96875 , 0.984375 , 0.96875 , 0.96875])]
Train Precision Scores:----------------- [array([0.97222222, 0.97222222, 0.98571429, 0.97222222, 0.97222222,
0.97222222, 0.97222222, 0.98571429, 0.97297297, 0.97297297])]

Predict Time:--------------------------- [0.001436200000171084]

3.1.3.4.3 sent_tfidf_unigram Model Scoring Details

Build Time: ---------------------------- [0.10605889999987994]
Fit Time:------------------------------- [array([0.00299168, 0.00299168, 0.00299215, 0.0039897 , 0.0039897 ,
0.004987 , 0.00299525, 0.00199509, 0.00199485, 0.00199461])]
Score Time:----------------------------- [array([0.0049839 , 0.00498629, 0.00398922, 0.00398922, 0.00398898,
0.0039885 , 0.00199437, 0.00199389, 0.00199413, 0.00299168])]
Test Recall Scores:--------------------- [array([0.625 , 0.375 , 0.66666667, 0.5 , 0.375 ,
0.5 , 0.375 , 0.5 , 0.5 , 0.66666667])]
Test Precision Scores:------------------ [array([0.8125 , 0.21428571, 0.83333333, 0.28571429, 0.25 ,
0.28571429, 0.25 , 0.28571429, 0.28571429, 0.83333333])]
Train Recall Scores:-------------------- [array([0.98214286, 0.96428571, 0.94827586, 0.98275862, 0.96551724,
1. , 0.98275862, 1. , 0.96551724, 0.96551724])]
Train Precision Scores:----------------- [array([0.98648649, 0.97435897, 0.9625 , 0.98684211, 0.97435897,
1. , 0.98684211, 1. , 0.97435897, 0.97435897])]

Predict Time:--------------------------- [0.001564199999847915]

3.1.3.4.4 sent_cnt_bigram Model Scoring Details

Build Time: ---------------------------- [0.17514979999941716]
Fit Time:------------------------------- [array([0.00698233, 0.00897717, 0.00797772, 0.00498772, 0.00797939,
0.00797796, 0.00498676, 0.00498629, 0.00598526, 0.00697994])]
Score Time:----------------------------- [array([0.0039885 , 0.00498724, 0.00398874, 0.00299168, 0.00299072,
0.00299191, 0.00199485, 0.00299263, 0.00398874, 0.00398946])]
Test Recall Scores:--------------------- [array([0.75 , 0.375 , 0.375 , 0.75 , 0.625 ,
0.41666667, 0.16666667, 0.5 , 0.66666667, 0.5])]
Test Precision Scores:------------------ [array([0.75 , 0.36666667, 0.36666667, 0.75 , 0.63333333,
0.41666667, 0.1 , 0.28571429, 0.66666667, 0.5])]
Train Recall Scores:-------------------- [array([0.96774194, 0.96774194, 0.96774194, 0.98387097, 0.98387097,
0.96875 , 0.96875 , 0.96875 , 0.96875 , 0.96875])]
Train Precision Scores:----------------- [array([0.97222222, 0.97222222, 0.97222222, 0.98571429, 0.98571429,
0.97222222, 0.97222222, 0.97222222, 0.97297297, 0.97297297])]

Predict Time:--------------------------- [0.0013465000001815497]

3.1.3.4.5 sent_tfidf_bigram Model Scoring Details

Build Time: ---------------------------- [0.15233990000069753]
Fit Time:------------------------------- [array([0.00598288, 0.00500178, 0.00598335, 0.00698304, 0.00598574,
0.00498748, 0.00598431, 0.00598383, 0.00598168, 0.00498891])]
Score Time:----------------------------- [array([0.00399852, 0.0039742 , 0.00398946, 0.00399017, 0.00398922,
0.00398922, 0.0039897 , 0.00398946, 0.0039885 , 0.00298977])]

Test Recall Scores:--------------------- [array([0.5 , 0.625 , 0.75 , 0.5 , 0.375 ,
0.41666667, 0.54166667, 0.58333333, 0.5 , 0.5])]
Test Precision Scores:------------------ [array([0.25 , 0.78571429, 0.83333333, 0.25 , 0.21428571,
0.41666667, 0.55 , 0.58333333, 0.25 , 0.25])]
Train Recall Scores:-------------------- [array([1. , 0.98387097, 0.98387097, 0.98387097, 0.98387097,
0.984375 , 0.984375 , 0.984375 , 0.984375 , 0.984375])]
Train Precision Scores:----------------- [array([1. , 0.98571429, 0.98571429, 0.98571429, 0.98571429,
0.98571429, 0.98571429, 0.98571429, 0.98611111, 0.98611111])]

Predict Time:--------------------------- [0.0012077999999974668]

3.1.3.4.6 sent_cnt_trigram Model Scoring Details

Build Time: ---------------------------- [0.28495100000054663]
Fit Time:------------------------------- [array([0.01296425, 0.01395988, 0.0149579 , 0.01296496, 0.01097107,
0.01096988, 0.01297235, 0.00797868, 0.00797939, 0.00698185])]
Score Time:----------------------------- [array([0.00598431, 0.0039897 , 0.00698209, 0.00498939, 0.00498748,
0.00598431, 0.00498104, 0.00299168, 0.00199461, 0.0039885])]
Test Recall Scores:--------------------- [array([0.375 , 0.375 , 0.25 , 0.75 , 0.5 ,
0.75 , 0.16666667, 0.16666667, 0.16666667, 0.33333333])]
Test Precision Scores:------------------ [array([0.21428571, 0.36666667, 0.16666667, 0.83333333, 0.5 ,
0.75 , 0.1 , 0.125 , 0.125 , 0.2])]
Train Recall Scores:-------------------- [array([1. , 1. , 1. , 1. , 1. ,
1. , 0.98484848, 1. , 1. , 1.])]
Train Precision Scores:----------------- [array([1. , 1. , 1. , 1. , 1. ,
1. , 0.98529412, 1. , 1. , 1.])]

Predict Time:--------------------------- [0.0019415999995544553]

3.1.3.4.7 sent_tfidf_trigram Model Scoring Details

Build Time: ---------------------------- [0.18237069999941014]
Fit Time:------------------------------- [array([0.00797677, 0.00897408, 0.00897884, 0.00897741, 0.0069828 ,
0.00698066, 0.0079782 , 0.00701332, 0.00498509, 0.00498915])]
Score Time:----------------------------- [array([0.00498581, 0.00499034, 0.00498724, 0.00498462, 0.00398898,
0.00498605, 0.00398874, 0.00295877, 0.00299168, 0.00299001])]
Test Recall Scores:--------------------- [array([0.25 , 0.375 , 0.25 , 0.375 , 0.5 ,
0.625 , 0.75 , 0.83333333, 0.16666667, 0.83333333])]
Test Precision Scores:------------------ [array([0.16666667, 0.36666667, 0.16666667, 0.21428571, 0.25 ,
0.63333333, 0.8 , 0.875 , 0.125 , 0.875])]
Train Recall Scores:-------------------- [array([1. , 1. , 1. , 1. , 1. ,
1. , 0.98484848, 1. , 1. , 1.])]
Train Precision Scores:----------------- [array([1. , 1. , 1. , 1. , 1. ,
1. , 0.98529412, 1. , 1. , 1.])]

Predict Time:--------------------------- [0.002271399999699497

4 Conclusions

Of the twelve models evaluated, only four of them scored above a 55% predicted accuracy rating on unseen customer reviews. 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.