Capstone Proposal

Carson Dahlberg - August 9th, 2018

Definition

Project Overview – Natural Language Processing (NLP)

Alan Turing published his famous article "Computing Machine Intelligence", in 1950, formalizing criteria to evaluate machine intelligence. A successful Turing Test[1] exhibited a machine performing so well in real-time natural conversation with a human, that a human evaluator would be unable to reliably distinguish between the program and a human. Turing firmly believed that machines could learn and achieve this level of intelligence.

Since Turing's time, and actually quite recently, these intelligent machines have grown considerably in viability[2]. Once, in the area of natural language processing (NLP), they were merely novelties, slow and not scalable, with little more than entertainment value. Today, they continue their rapid transition to highly functional applications, effectively creating value. Their usefulness is especially salient when placed in the context of intractable data size and frequency coupled with insurmountable demands from the limited human capacity of even domain experts.

My personal motivation for choosing this domain is that intelligent applications' potential is very intriguing, and the constant innovation is exciting. As Aurelien Geron, Sr. Artificial Intelligence Engineer states in his book, "Machine Learning has conquered the industry: it is now at the heart of much of the magic in today's high-tech products, ranking your web search results, powering your smartphone's speech recognition, and recommending videos."[3] I want to lead the charge, where I work, to unleash intelligent apps on our unstructured textual data, solving real-world problems and creating value by reduce costs, eliminate boring tasks, and aid/empower expert humans.

Problem Statement - Classifying Sentiment

The goal is to explore using machine learning models for solving the NLP-related tasks involving sentiment analysis (classifying polarity as negative or positive). The scope of this project will focus on creating practical models which not only succeed in predicting sentiment, but meet the requirement of explainability.

Balancing these requirements is challenging since human understanding of explainability is generally quite different than how a model explains a prediction given a dataset. The "best" model will likely be chosen based off of compromises between resources like funding, time allocated for research to production, the complexity and explability of the model, and technical requirements like time for prediction. Quick, simple, and explainable models can go a long way to allay the fear of reputational risk of a data science business champion. Simple models can create momentum of interest and buy-in. This, in turn, can be parlayed into more grandiose projects.

---

[1] [PDF] Turing, A. M., Computing Machinery and Intelligence - Mind, A Quarterly Review of Psychology and Philosophy (October, 1950).

[2] [WEB] Bird, S., Klein, E., Loper, E., Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit, Preface page ix. O'Reilly Media (June, 2009).

[3] Geron, A., Hands-On Machine Learning with Scikit-Learn & TensorFlow. O'Reilly Media (2017).

The task involved in classifying sentiment of social media reviews (movies, restaurants, products, and services) are:

1. find a dataset balanced between classes (performance evaluation less complicated)
2. create a corpus to train models, clean up/preprocess the text data for machine learning
3. represent the data as various vectors and embeddings for modelling sentiment
4. apply logistic regression to learn to predict the probability that text is positive/negative sentiment

Evaluation Metrics

We will compare the strengths, weaknesses, and trade offs when considering which models to select. We will create a framework for solving sentiment problems in the real world – including being able to communicate model's limitations, building an intuition, and explaining trade offs to inform the business' decision making.

- **Quantifiable Tasks** - the models should be able to predict the polarity of the text sentiment (positive versus negative).

In the equations below, ***tp*** is true positives and ***tn*** is true negatives.

- **Accuracy** – the percentage of text classified correctly (positive/negative sentiment).

$$Accuracy = \frac{tn + tp}{tp + tn + fp + fn}$$

- **Precision** – the percentage of all text classified as belonging to a given class that actually belong to that class.

$$Precision = \frac{tp}{tp + fp}$$

- **Recall** – the percentage of text associated with a particular class that were classified as such. I will examine both Positive and Negative Recall.

$$Recall = \frac{tp}{tp + fn}$$

- **F-beta Score** - is a metric that considers both precision and recall:

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$$

  - This is called F-score for simplicity. Precision and Recall wikipedia
    - It becomes the **F**0.5 **score** when β =0.5, more emphasis is placed on **precision**.
    - It becomes the **F**2 **score** when β =2, more emphasis is placed on **recall**.

- Maximum precision (no false positive/ type I errors) and maximum recall (no false negatives/ type II errors). Precision is measuring exactness/quality/relevant results while recall measures completeness/most relevant results returned. One way that I think about the comparison is that precision is exclusive (focused on the results being correct but not getting all correct labels) and recall is inclusive (focus on getting all correct labels back with the tradeoff of the wide net capturing more false positive).
- **Justification for using F0.5 score -** For the engineered features, we will use β =0.5 to focus on quality/relevant results. This is because it is more experimental and sentiment is already subjective, so we'd like the polarity to be relevant. This makes interpretation easier. For the n-gram features we will use the **F1 score** to balance between the two as to get more results with some false positive versus few results with no false positives.

- **Error Analysis**
  - **Variation** – investigate if particular models encountered errors with particular topics or tasks. This might be visible between different datasets (for example, perhaps a model doesn't generalize well to long texts).
  - **Confusion Matrix** – This would make sense to use if can accurately tag topics and zero in on whether particular topics are misinterpreted consistently/identify where a model can be improved/ is inherently weak (maybe several attempts do not improve the model).
- **Qualitative Investigation of Misinterpreted Classes**
  - user purpose – questions/requests mistaken as compliment/criticism?
  - Semantics/sentence structure/subtle language cues – jokes, sarcasm, rhetoric and literary devices
  - Atypical word usage like swearing used colloquially as a compliment
  - Errors due to mixed/conflicted sentiment
  - Discussing multiple topics (potentially with mixed sentiment)

Analysis

Data Exploration

Respecting the reviewer's time to evaluate the project, I will only work on a subset of data, in order to keep the project size to a sensible size.

Data Guidelines in General – the standardized dataset selected is balanced between negative and positive sentiment, allowing us to focus on modeling and making performance measurement easier. Data Pipelines will be created to maintain consistent/repeatable data being sent to the models between training and testing. These pipelines will consist of noise removal and normalizing text, like removing HTML, removing non-ascii, lower casing, removing punctuation, replacing numbers with words, removing stop words, stemming and lemmatization. They will also create data representations as vectors and embeddings for the model to predict classes. In addition, hand-crafted features will be created to try to capture the information lost from processing the data, such as removing punctuation.

Dataset Details - Sentiment Labelled Sentences Dataset – from UCI Machine Learning Repository.

- This was created for the Paper 'From Group to Individual Labels using Deep Features', Kotzias et. al,. KDD 2015.
- `**score**` - sentences labelled with positive or negative sentiment, with scores of either 1 (for positive) or 0 (for negative).
- `**source**` - sentences were extracted from reviews of products, movies, and restaurants from three different websites/fields: **imdb.com**, **amazon.com**, and **yelp.com**. For each website, there are 500 positive and 500

negative sentences, selected randomly from their respective larger datasets, having a clearly positive or negative connotation. The goal was for no neutral sentences to be selected.

- Here are some example text, from different sources:

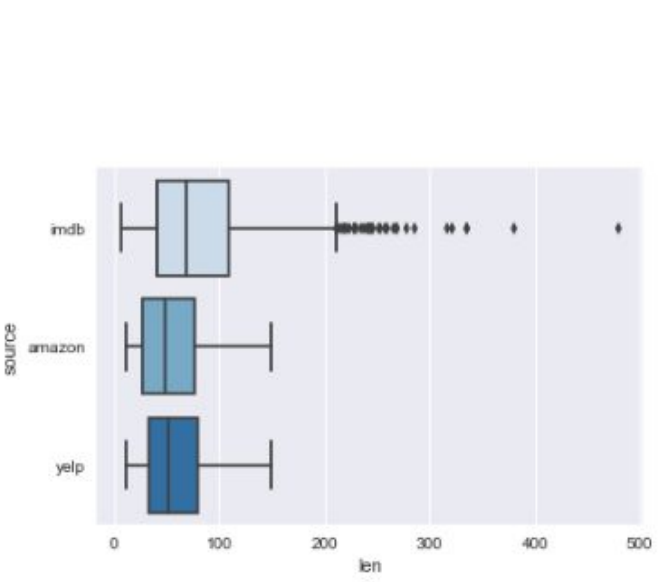
	document	score	source
0	So there is no way for me to plug it in here in the US unless I go by a converter.	0	amazon
1	Good case, Excellent value.	1	amazon
2	Great for the jawbone.	1	amazon
3	Tied to charger for conversations lasting more than 45 minutes.MAJOR PROBLEMS!!	0	amazon
4	The mic is great.	1	amazon

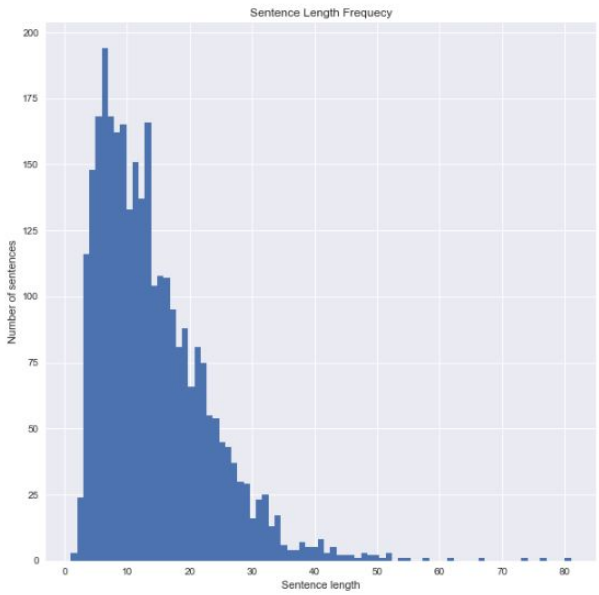
	document	score	source
0	A very, very, very slow-moving, aimless movie about a distressed, drifting young man.	0	imdb
1	Not sure who was more lost - the flat characters or the audience, nearly half of whom walked out.	0	imdb
2	Attempting artiness with black & white and clever camera angles, the movie disappointed - became even more ridiculous - as the acting was poor and the plot and lines almost non-existent.	0	imdb
3	Very little music or anything to speak of.	0	imdb
4	The best scene in the movie was when Gerardo is trying to find a song that keeps running through his head.	1	imdb

	document	score	source
0	Wow... Loved this place.	1	yelp
1	Crust is not good.	0	yelp
2	Not tasty and the texture was just nasty.	0	yelp
3	Stopped by during the late May bank holiday off Rick Steve recommendation and loved it.	1	yelp
4	The selection on the menu was great and so were the prices.	1	yelp

Exploration Visualization

- Observation - regarding length of documents, amazon and yelp are more similar, with imdb having the most outliers. This can be visualized in the boxplot below:

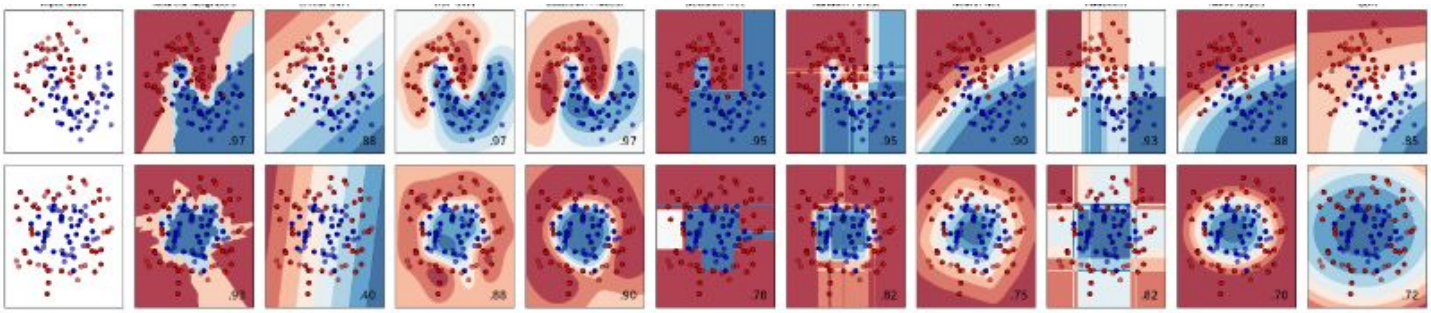




(above left: boxplots of document lengths per source, above right: sentence length frequency - all sources)

- I will select and label a small sample size of sentences from **UrbanDictionary.com** to examine model efficacy. Below is an example.

	document	score
0	Expensive restaurants with horrible food that people still go to for the prime location or atmosphere.	0
1	It has a beautiful view of the Eiffel Tower!	1
2	Nah, that place is a total touristaurant, my food was awful last time I ate there.	0
3	I had a great day, climbed a hill and didn't check facebook.	1
4	That corgi-shaped macaron is expensive because it's got instagram tax.	1



Algorithms and Techniques

Summary

We experiment with algorithms that have shown efficacy with practical application of NLP tasks. All algorithms are fed the same preprocessed data discussed in the following **Data Preprocessing** section. For the text representations, we will use count-based, probabilistic language models which create probability distributions given a sequence of tokens, which include Bag-of-Words (BoWs) word-level counts as well as rescaled TF-IDF (term frequency-inverse document frequency), and Bag-of-Characters (BoCs character-level ngrams). These distributions encode some of the complexities of language, as well as providing a way to capture and reduce information in the corpora contains. We will also incorporate a word2vec pretrained model which has learned 300 vector embeddings to represent similarity between words, which can be quite useful.

These will all be combined with a supervised **logistic regression** model for binary prediction, usinf k-fold (5) cross-validation as well as grid search techniques over the hypothesis space in order to discover optimal models with the given dataset. Finally, a voting classifier will be constructed incorporating the best-of-class models (BoWs, BoCs, TF-IDF BoWs, word2vec all with Logistic Regression) in order to achieve higher performance metrics than attainable by said individual models.

Bag-of-N-grams

Scikit-Learn's **CountVectorizer** is a count-based implementation of BoWs word-level or BoCs character-level ngrams. It has shown to be a solid performer for its simplicity which makes it easily interpretable. Text documents are described by word occurrences, completely ignoring the relative position information of the words. Hence, one limitation is that it needs help capturing context. This is done by representing word order by combining tokens into combinations of n-grams (two or more combined character or word-level tokens). Each of these n-grams is treated as a feature. Hence as the number of n-grams grows, so to the order of complexity, which in turn can affect prediction times. The CountVectorizer Class then builds a vocabulary based on the parameters passed like minimum and maximum document frequencies. These are in turn used as features which are fed to the sklearn **LogisticRegression** Class Classifier. The hyperparameters chosen can be seen in the Baseline Bag-of-Words + Logistic Regression in the Implementation and Refinement Section. Key hyperparameters are:

- **analyzer**='word' | 'char_wb('word' used for word n-grams and 'char_wb' for only using text inside of words to build character n-grams)
- **max_df**=0.7 (when building the vocabulary, this will ignore terms that have a document frequency higher than 70%)
- **max_features**=10000 (number of n-grams to build)
- **min_df**=1 (this value is the 'cut-off' where the term will be ignored if occurring less than the threshold), we are including all words)
- **ngram_range**=(1, 1) the baseline is uni-gram
- **preprocessor**=None (all preprocessing done in the Data Preprocessing stage of pipeline, allowing for experimentation)
- **stop_words**='english' (sklearn has more stop words that appear more appropriate for this task and we'll utilize them).
- **token_pattern**='(?u)\\b\\w\\w+\\b' (simple regex pattern taking all word tokens separated by word boundaries)

Scikit-Learn's **TfidfVectorizer** will be used for TF-IDF (term frequency-inverse document frequency), which will take the vectorized count and rescale the values to take advantage of looking for documents that are unique to text but less common in general. This is completed by rescaling the data. Instead of dropping features (words, etc.) that are assumed to be unimportant, another approach is to keep them all and rescale them. The rescaling is according to how informative they are expected to be. The basic assumption is that the more important terms are the ones which appear often in a particular document, but not in many documents in the corpus. Some intuition behind this idea is that if this condition holds, the term is likely very descriptive of that document. Here are some details on the variants of tf-idf on wikipedia.

We can apply this with sklearn, which uses the following in the TfidfVectorizer:

$$\text{tfidf}(w, d) = \text{tf} \log\left(\frac{N+1}{N_w+1}\right) + 1$$

- *N* is the number of docs in training set
- Nw is the num of docs in the training set
- *w* is the word
- *tf* is the word frequency, the num times *w* appears in the document *d*

We'll use sklearn's TfidfVectorizer & Pipeline to apply this concept with the following hyperparameters:

- **analyzer** = 'word' (this is used to create word-level n-grams)
- **max_features** = 10,000 (number of n-grams to build)
- **sublinear_df** = True (replaces term frequency with 1 + log(tf)
- **min_df** = 5 (min num of docs a word must be present in to be kept)
- **max_df** = default 1 (max num of docs a word must be present in to be kept, this keeps all)
- **norm** = l2 (to ensure all feature vectors have euclidean norm of 1)
- **ngram_range** = (1, 2) (for both unigrams and bigrams initially and for grid searching [(1,1), (1,2), (1,3)])
- **stop_words** = "english" (to reduce noisy features)

Word Embeddings

Finally, we will employ vector space word representation created by Tom Mikolov et al, from Google's research arm, called **word2vec**.[4] The key difference between word2vec and the the Bag-of-N-Grams representations are that the Bag-of-N-Grams treats every term as atomic, meaning it has no connection to the rest of the text for additional meaning. What Tom Mikolov et al achieved was an efficient was to train typical neural net architectures to learn vector representations of words using an extremely large corpus, first 1.6 billion words, and then later on +100 billion words from Google's News data.

Using the novel architecture and approach of skip-gram models, they created the ability for the model to learn semantic and syntactic representations of words which could then be used for similarity comparisons. These representations also

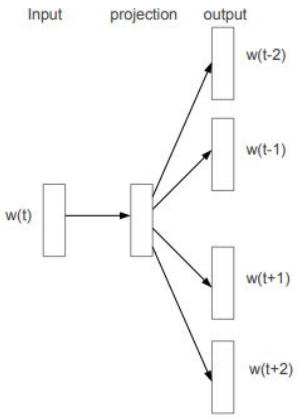
[4] [pdf] Mikolov, T., Chen, K., Corrado, G., Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space.

find vectors for phrases in which their meaning is more than the combined compositionality of the individual terms. This additive compositionality was demonstrated in Table 5, on page 5 of their second paper[5], shown here.

Czech + currency	Vietnam + capital	German + airlines	Russian + river	French + actress
koruna	Hanoi	airline Lufthansa	Moscow	Juliette Binoche
Check crown	Ho Chi Minh City	carrier Lufthansa	Volga River	Vanessa Paradis
Polish zolty	Viet Nam	flag carrier Lufthansa	upriver	Charlotte Gainsbourg
CTK	Vietnamese	Lufthansa	Russia	Cecile De

Table 5: Vector compositionality using element-wise addition. Four closest tokens to the sum of two vectors are shown, using the best Skip-gram model.

The skip-gram model (architecture shown on the right figure) has the job of finding the word representations that are useful for predicting surrounding words in sentences and other text documents. To do this, a vocabulary is created for reference, similar to the Bag-of-N-Grams. Frequently occurring n-grams like “New York Times” and “Toronto Maple Leafs” are combined into single tokens, allowing for a more concise/manageable vocabulary.

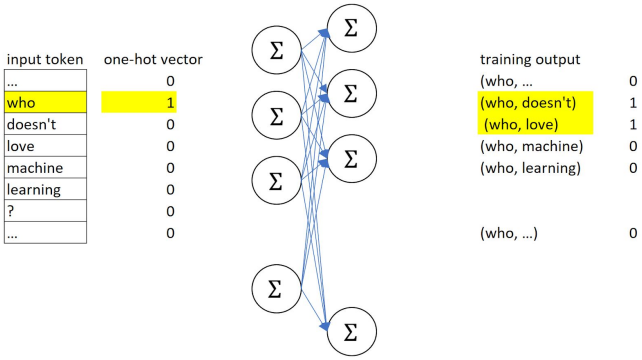


A rolling window, of size n, of tokens pairs are fed to the model. The model then learns statistics from the number of times the pairs occur in the training corpus. Below is a basic example of how the data pairs are created with a 2-word window.

training text						pairs
Who	doesn't	love	machine	learning	?	(who, doesn't), (who, love)
Who	doesn't	love	machine	learning	?	(doesn't, who), (doesn't, love), (doesn't, machine)
Who	doesn't	love	machine	learning	?	(love, who), (love, doesn't), (love, machine), (love, learning)
Who	doesn't	love	machine	learning	?	(machine, doesn't), (machine, love), (machine, learning), (machine, ?)

(Above left: example of skip-gram window word pairs, Above right: skip-gram model architecture)

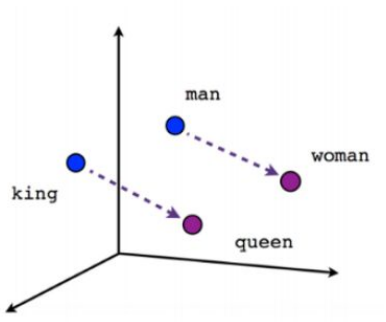
These word pairs are fed in as one-hot encoding vectors for the skip-gram model, which outputs probability distributions. The words represent the rows and the features the columns. To capture the idea, with our very basic example text, the figure to the right shows the skip-gram model.



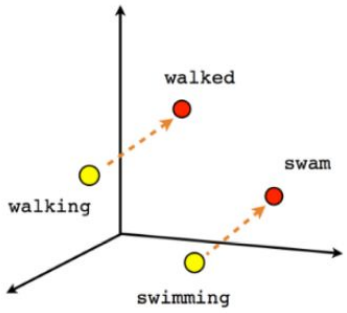
In their research, they initially chose 1.6 Billion words and then the model learned 50, 100, and 300 features (from hidden neurons). They also use several modifications such as applying a simple subsampling approach for the training output to “counter the balance between rare and infrequent words over large text.”

We will gain access to **word2vec** via Gensim’s Python-based nlp package. The pre-trained model that is a hefty 1.5GB in size containing 300 vectors for a vocabulary of over 3 million words and phrases, will allow us to capture semantic meaning via similarity distance metrics using the vectors, as graphically shown below.

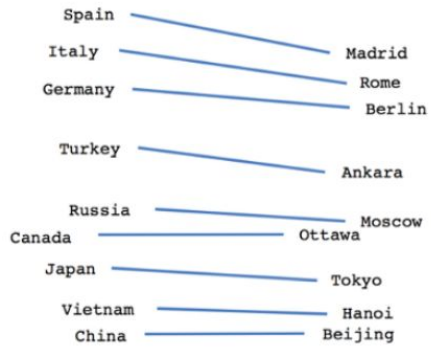
[5] [pdf] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed Representations of Words and Phrases and their Compositionality. Accepted to NIPS 2013.



Male-Female



Verb tense



Country-Capital

Helper functions were created to compute the average vector of each sentence which will be input for the **LogisticRegression** learner to predict sentiment. The average vector can be described as:

For a text T, composed of words $w_1, \ldots, w_n$ and word2vec embeddings $vec_{w_1}, \ldots, vec_{w_i}$:

$$T = \frac{1}{n} \sum_{i=1}^{n} vec_{w_i}$$

Logistic Regression

For consistency of comparison, except for the engineered features experiments, the same preprocessing (see Data Preprocessing section) will be applied to BoWs, BoCs, TF-IDF, and word2vec representations. These representations will also be coupled with the same **LogisticRegression** classifier to model sentiment.

Logistic Regression is a linear classifier that models the probabilities of classes with the logistic function. The logistic function (also referred to as the Sigmoid Function), can take any real number and then map it to a range between 0 and 1 (translating to 0 - 100% probability). For linear models:

$y = X\beta + \epsilon$, where

- X : data
- y : target variable
- β : Coefficients
- ϵ : Observation noise

If we were to use linear regression for this task, it would not give good results because it would lend too much weight from the decision frontier. The sigmoid function corrects this:

$$S(z) = \frac{1}{1 + e^{-z}}$$

- s(z) = output between 0 and 1 (probability estimate)
- z = input to the function (your algorithm's prediction e.g. mx + b)
- e = base of natural log (Euler's number)

is fit instead, which essentially helps us learn a step function between the two classes, to map predicted values to probabilities which are constrained between 0 and 1 (equation from sklearn docs):

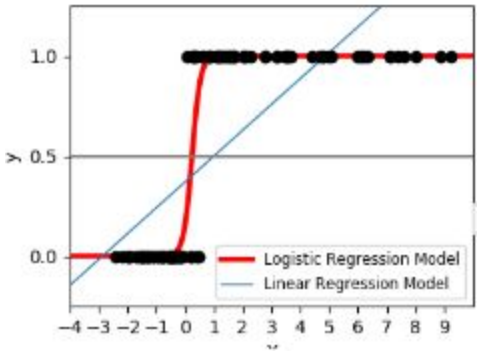
$$y = \text{sigmoid}(X\beta - \text{offset}) + \epsilon = \frac{1}{1 + \exp(-X\beta + \text{offset})} + \epsilon$$

The job of the classifier is to discover the best value for the coefficients. As stated in sklearn, as an optimization problem, binary class L2 penalized logistic regression minimizes the following cost function (the solver "liblinear" uses a coordinate descent (CD) algorithm):

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1).$$

The cost function introduces a cost term for the addition of features (model complexity) and pushes coefficients towards zero (reducing the cost term). It is using cross-entropy (aka log loss) instead of MSE to avoid resulting in several local minima, which could impede the convergence upon a solution with gradient descent (an iterative process that makes a prediction for all instances based on the current weights, and calculates new weights, until a desired accuracy is met or a limit of iteration is met). It is the process of updating the weights that is what the model is truly learning.

The chart on the right shows the decision function learned from the sigmoid curve (image and equations from sklearn documentation). The decision boundary is the threshold which demarcates which class is predicted:



- p≥0.5,class=1
- p<0.5,class=0

To summarize the process:

1. we are modeling the probability of a class given inputs X
    a. p(X) = p(Y=1| X), for p≥0.5 (which is plotted)
    b. p(X) = p(Y=0| X), for p<0.5
2. transforming the predictions using the sigmoid function
    a. p(X) = e^(b0 + b1*X + ...) / (1 + e^(b0 + b1*X + ...))
3. calculating/predicting the log odds
    a. ln(p(X) / 1 – p(X)) = b0 + b1 * X + ...
    b. ln(odds) = b0 + b1 * X + ...
4. OR predicting the odds
    a. odds = e^(b0 + b1 * X)
5. and/or predicting the class
    a. p≥0.5,class=1
    b. p<0.5,class=0

In our project, the logistic function is learned to model the probability of a text belonging to positive/negative sentiment given the input variables (the character and word-level n-grams representations of the corpus of training text). The model then uses this learned function over the input for the text (features present in the text) to predict and map to a class/class probability.

Scikit-Learn's LogisticRegression default hyperparameters were used for the baselines and are listed below. Only the regularization coefficient, C.

	baseline	**grid search**
• **penalty**:	l2	no
• **tol** (tolerance for stopping criteria):	1e-4	no
• **C** (inverse of regularization strength):	1.0	[0.001, 0.01, 0.1, 1, 10, 100]
• **fit_intercept**:	True	no

- **intercept_scaling**: 1 no
- **class_weight** (classes balanced): None no
- **random_state**: 42 no
- **solver** (algo used for optimization): liblinear no

Benchmarks

- Naïve models – results of consistently guessing same class
- Simple Models – baseline of default settings Scikit-Learn's Logistic Regression on Bag-of-Words (only uni-grams) without hyperparameter tuning

Methodology

Data Preprocessing

One of the challenges with textual data is to represent it in a form that both enables the models to learn as well as being manageable. This must be balanced with losing information in the preprocessing. The steps used in the data pipeline for this project are standard ones for the practicing industry. The data preprocessing steps are shown with toy examples of the pipeline.

First is the cleanup, which deals with headers and footers, HTML, XML, markup an metadata, while attempting to extract data from formats like JSON, and removing square brackets.

	document
1454	I've also had problems with the phone reading the memory card in which I always turn it on and then off again.
1136	Very good stuff for the price.
2137	This is one of the best bars with food in Vegas.
2276	I liked the patio and the service was outstanding.
1198	The pleather case doesn't fit.
1167	The look of it is very sharp and the screen is nice and clear, with great graphics.

Next, contractions like 'don't' are replaced with 'do not' and tokenization is applied. Note that words and punctuation are given their own token.

	document
1454	[I, have, also, had, problems, with, the, phone, reading, the, memory, card, in, which, I, always, turn, it, on, and, then, off, again, .]
1136	[Very, good, stuff, for, the, price, .]
2137	[This, is, one, of, the, best, bars, with, food, in, Vegas, .]
2276	[I, liked, the, patio, and, the, service, was, outstanding, .]
1198	[The, pleather, case, does, not, fit, .]
1167	[The, look, of, it, is, very, sharp, and, the, screen, is, nice, and, clear, ,, with, great, graphics, .]

To reduce the data, while maintaining most of the documents' meaning, normalization is applied. Demonstrated below are the effects after removing non-ascii characters from word tokens, converting all characters to lowercase, removing punctuation, replacing numbers with their equivalent words, and removing stop words (which are like the plumbing for languages and don't really capture much meaning but their removing aids processing speed). Additionally, verbs are lemmatized.

	document
1454	[also, problems, phone, read, memory, card, always, turn]
1136	[good, stuff, price]
2137	[one, best, bar, food, vegas]
2276	[like, patio, service, outstanding]
1198	[pleather, case, fit]
1167	[look, sharp, screen, nice, clear, great, graphics]

The models will then take these preprocessed tokens and transform them into count-based representations like Bag-of-Words (really n-grams) which is a term-matrix for the corpus, and a Term Frequency - Inverse Document Frequency (TF-IDF) rescaled version which helps to reduce much of the shortcomings of the Bag-of-Words.

Here is a small representation of the Bag-of-Words, displaying the head, every 100th feature, and tail, from a vocabulary of 3,777 words. The vocabulary is represented in a dictionary format with the key: value format of word token: count. This representation will be utilized to a Logistic Regression learning which will assign weights to each token in proportion to the frequency of the token's appearance in the corpora (for this project, it is the training data).

```
features head: ['15lb', '18th', '20th', '20the', '2mp']

every 100th feature:
['15lb', 'aluminum', 'authentic', 'better', 'brownish', 'cell', 'click', 'construct', 'customer', 'dif
ficult', 'dull', 'entire', 'far', 'footage', 'giant', 'happen', 'howeverthe', 'instal', 'kabuki', 'lil
t', 'marion', 'mistake', 'nice', 'outdoor', 'periodically', 'possibility', 'puzzlesolving', 'relax',
'rpger', 'selfindulgent', 'sit', 'spicier', 'sublimely', 'teen', 'track', 'unfortunate', 'violence',
'windows']

features tail: ['zero', 'zillion', 'zombie', 'zombiestudents', 'zombiez']
```

Hand-crafted, sentence-level features will be used as inputs to models in an attempt to explore recapturing information lost by data preprocessing steps. To accomplish this, helper functions were created to be applied to a Pandas DataFrame of rows of text. These functions reside in the `helpers.py` project file. From the NLTK online book, using Python to represent the following is natural and intuitive:

- **{*w* | *w* E V & *P*(*w*)}**

which means, "The set of all w such that w is an element of V (vocabulary) and w has property P." This can be represented in Python using a list generator:

- **[w for w in V if p(w)]**

and is the form used to produce the sentence level features

Sentence Level Features include:

- **is_capitalized** - True/False if the sentence starts with a capitalized letter
- **all_caps_sent** - True/False if the sentence is all caps
- **all_caps_word_cnt** - Integer number of words in the sentence which are all caps
- **is_all_lower** - True/False if the sentence is all lowercase
- **num_chars** - Integer number of characters in the sentence
- **word_count** - Integer number of total words (including duplicates) in the sentence
- **word_to_char_frac** - this is average number of characters per word per sentence
- **vocab_size** - Integer number of unique words in the sentence
- **diversity_score** - Float number fraction of total words / total unique words (aka lexical diversity)
- **frac_unusual** - Float number fraction of unusual English words which are defined here as unusual or mispelled words using Word Corpus from the /user/share/dict/words Unix file used by spellcheckers

- **frac_non_stop_words** - Float number fraction of stop words in a sentence

There were more that features I wanted to add, but did not have time:

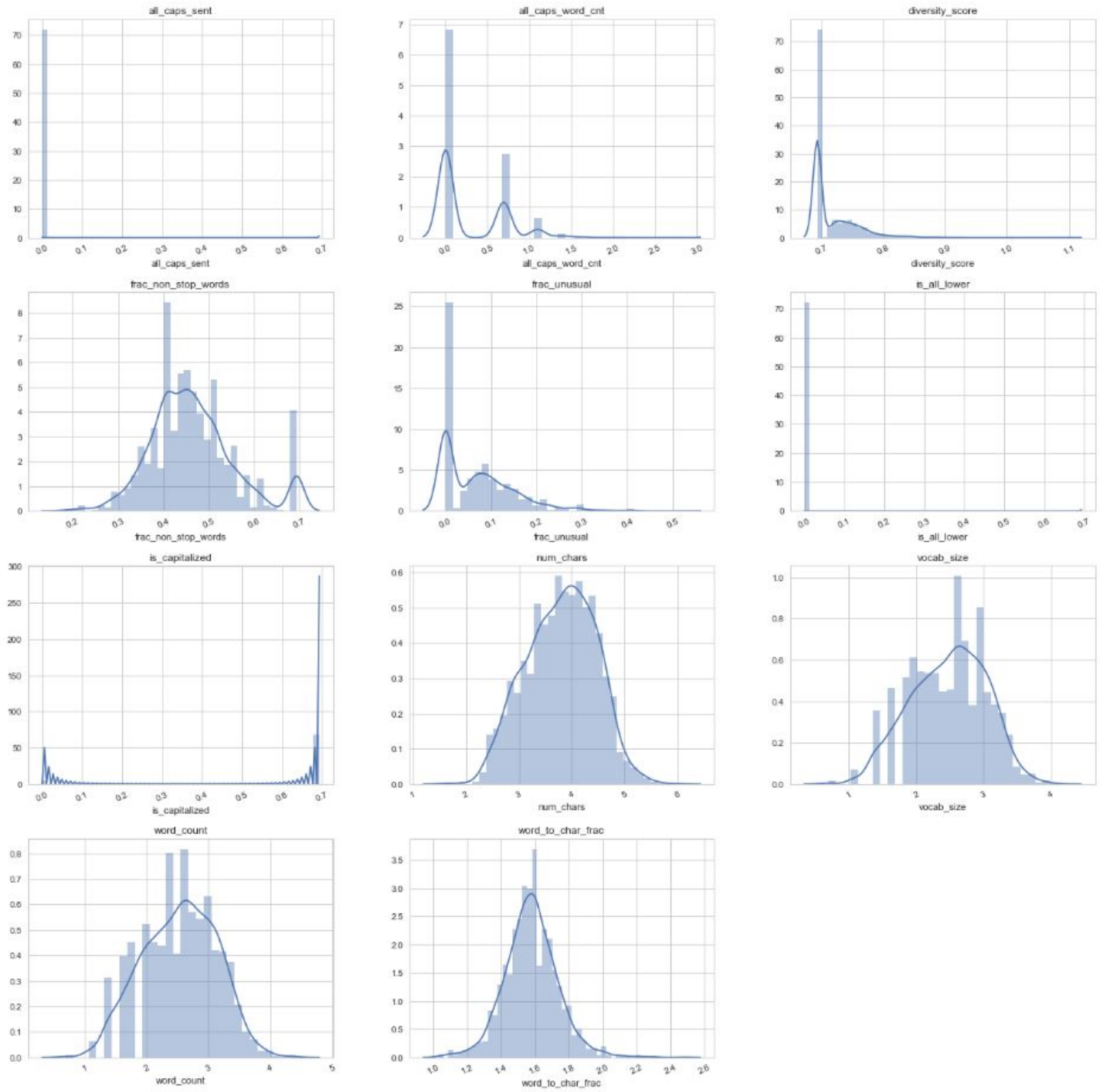
- more sentence level features: if there are ellipsis, hyphenated words, abbreviations, if money is in the sentence ex. 'All that for $15.00!!!' (emotions are typically tied to money), and
- word level-features (misspelled words - perhaps negative sentiment affects spelling?), etcetera.

Below are some Sentence-Level Features highlights and distributions:

- The vocab_size (unique words per document) is 11 words
- The average document has 7.6% frac_unusual (unusual, misspelled, or using words from the Word Corpus from the /user/share/dict/words Unix file used by spellcheckers)
- The frac_non_stop_words was interesting, as even the 25th quartile was at 50% and the 75th at 66%.
- It can be seen that most of the features were fairly normal before any transformations were applied

	count	mean	std	min	25%	50%	75%	max
all_caps_word_cnt	3000.0	0.474333	0.875365	0.0	0.0	0.000000	1.000000	15.000000
diversity_score	3000.0	1.051270	0.084383	1.0	1.0	1.000000	1.090909	2.000000
frac_non_stop_words	3000.0	0.596731	0.152479	0.2	0.5	0.571429	0.666667	1.000000
frac_unusual	3000.0	0.076202	0.087251	0.0	0.0	0.062500	0.125000	0.666667
num_chars	3000.0	53.570333	36.295530	4.0	27.0	46.000000	72.000000	407.000000
vocab_size	3000.0	12.803667	7.514116	1.0	7.0	11.000000	17.000000	59.000000
word_count	3000.0	13.886333	8.960181	1.0	7.0	12.000000	19.000000	81.000000
word_to_char_frac	3000.0	3.901527	0.801059	1.8	3.4	3.833333	4.316388	11.000000

(Above: Sentence-Level Features Summary Statistics)



(Above: Distributions of Hand-Crafted, Sentence-Level Features)

Implementation and Refinement

There were some coding challenges in this project. Specifically, implementing word2vec into the voting models. A second issue was recycling code from class that was written for Python 2. This created graphical issues that took over a week of debugging before I broke off that section to a separate notebook. Finally, trying to implement some of Scikit-Learn's custom Class creation functionality was not effective.

The model exploration and refinement were conducted as follows:

- Baseline Models - Naïve Model - guessing majority class, baseline of Default Settings Logistic Regression on Bag-of-Words (only uni-grams)

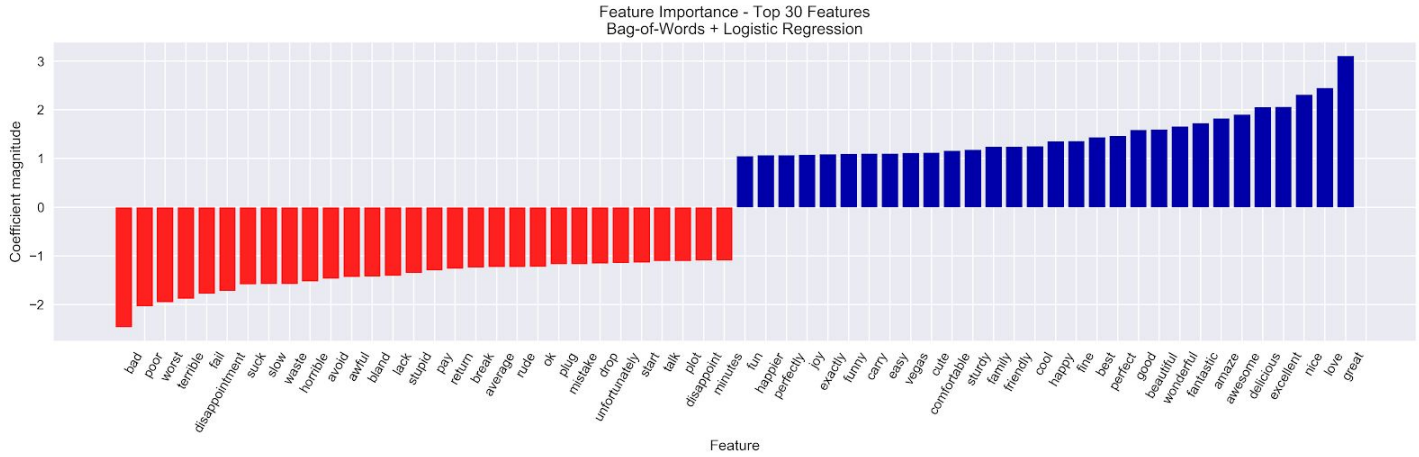
- Default Settings Logistic Regression Using TFIDF (weighting) scale down impact of very frequent tokens versus those occurring in a smaller fraction of documents Ensemble models
- Feature engineering
    - N-grams
        - Logistic Regression with Bag of Words adding n-grams to capture some context
        - Logistic Regression with TF-IDF adding n-grams
        - Logistic Regression with Bag of Characters
    - Grid search combination of models - attempt to recapture lost information from text pre-processing using presence and type of Characters, Punctuation, Digits, Unusual Words/Slang, Diversity of Text, Fraction of Stopwords, Average Word Length and Sentence Length
- word2vec – out of box (would train on corpus if had time)

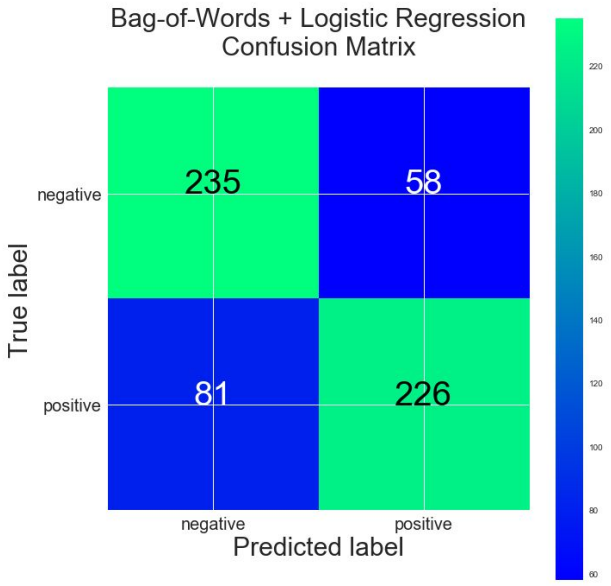
Benchmark Naive Predictor:

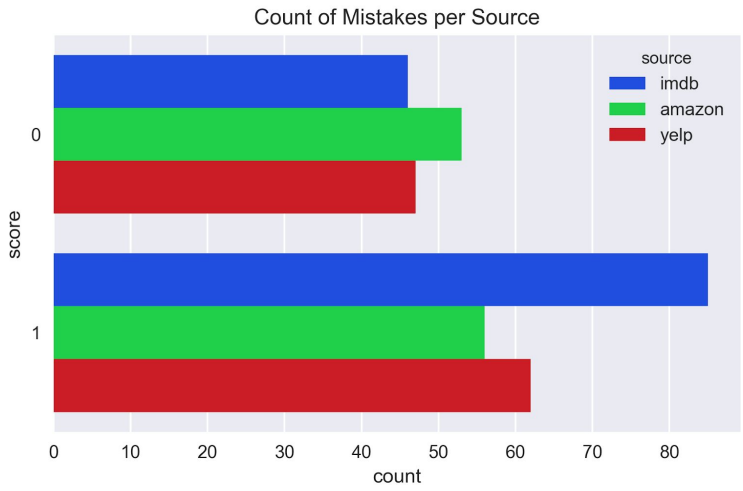
- accuracy = 0.500, precision = 0.500, recall = 1.000, f1 = 0.556
- simple baseline is just guessing the majority class (which in this case the dataset has been carefully selected to be balanced)
- the guess should be 0.50 and the same for the f1 score (and this is what we see in the results below)

Baseline Model - Bag-of-Words + Logistic Regression:

- Mean cross-validation accuracy: 0.79
- accuracy = 0.768, precision = 0.770, recall = 0.768, f1 = 0.768
- sklearn's simple count vectorization for BOWs
- using the some of the default **sklearn** params for **CountVectorizer**
    - tokenizing - default regexp select tokens of 2 or more alphanumeric characters (punctuation is completely ignored and always treated as a token separator)
    - default is lowercase=True
    - vocabulary building
    - sparse matrix encoding
    - using **sklearn ENGLISH_STOP_WORDS** (note: sklearn has more stopwords than **NLTL** out of the box)
- using some specified params for consistency between comparing models
    - setting max features to 10k
    - ignore terms w/ doc freq higher than threshold above 0.70
        - using 1 n-gram
- using the default **sklearn** params for **LogisticRegression**
- using 5 fold cross validation

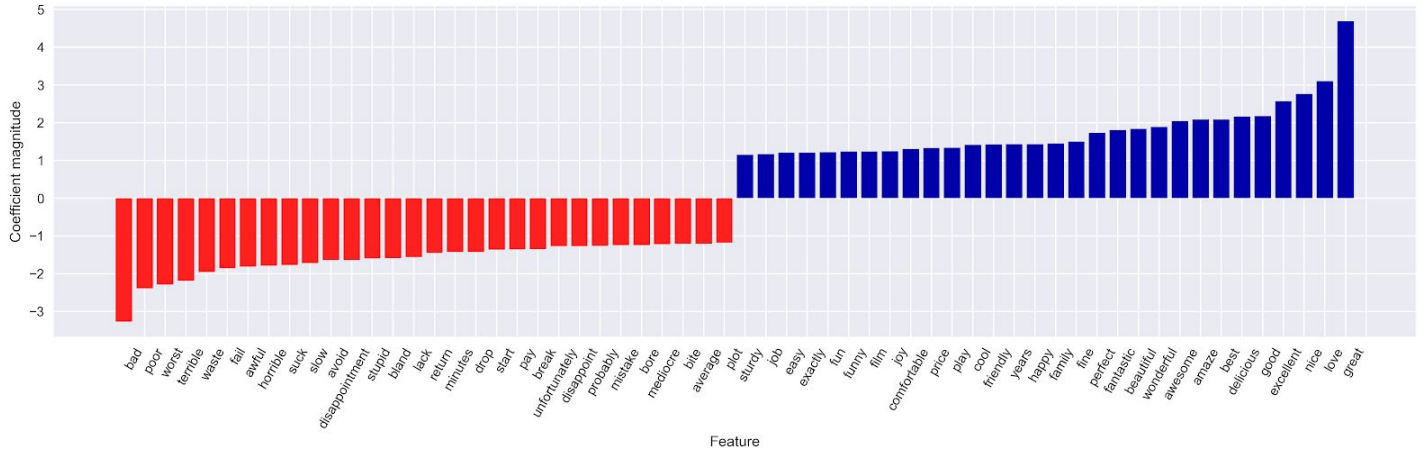


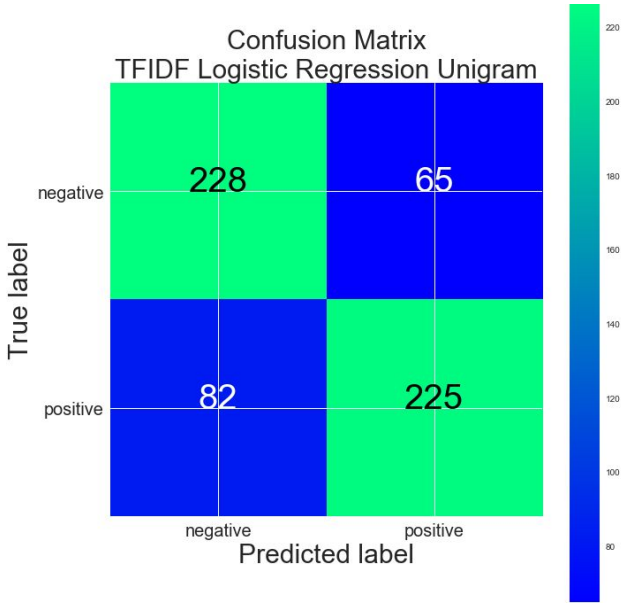


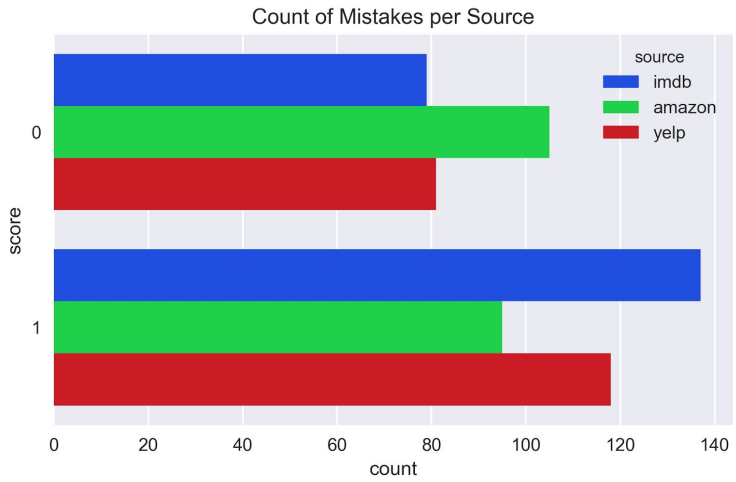


Baseline Model - TF-IDF + Logistic Regression:

- Mean cross-validation accuracy: 0.78
- accuracy = 0.755, precision = 0.756, recall = 0.755, f1 = 0.755







High Confidence vs. Low Confidence Accuracy

This will help to build intuition as to why a model is behaving the way it does. The review of accurate predictions while contrasting high versus low can sometimes yield additional insight. The high confidence predictions tended to be longer while the shorter ones likely suffered due to the preprocessing removal of words.

High confidence predictions seem to be working well. It is also worth noting that the model was able to distinguish a word like 'waste' used in negative and positive context ("waste of my time" vs. "money wasted the right way").

Below is a sample of high confidence accurate prediction of negative sentiment:

	document	score	y_pred_prob_def_bow_0
0	A very, very, very slow-moving, aimless movie about a distressed, drifting young man.	0	0.848500
1	Not sure who was more lost - the flat characters or the audience, nearly half of whom walked out.	0	0.881240
2	Attempting artiness with black & white and clever camera angles, the movie disappointed - became even more ridiculous - as the acting was poor and the plot and lines almost non-existent.	0	0.925928
15	It had some average acting from the main person, and it was a low budget as you clearly can see.	0	0.943852
34	Today the graphics are crap.	0	0.879759

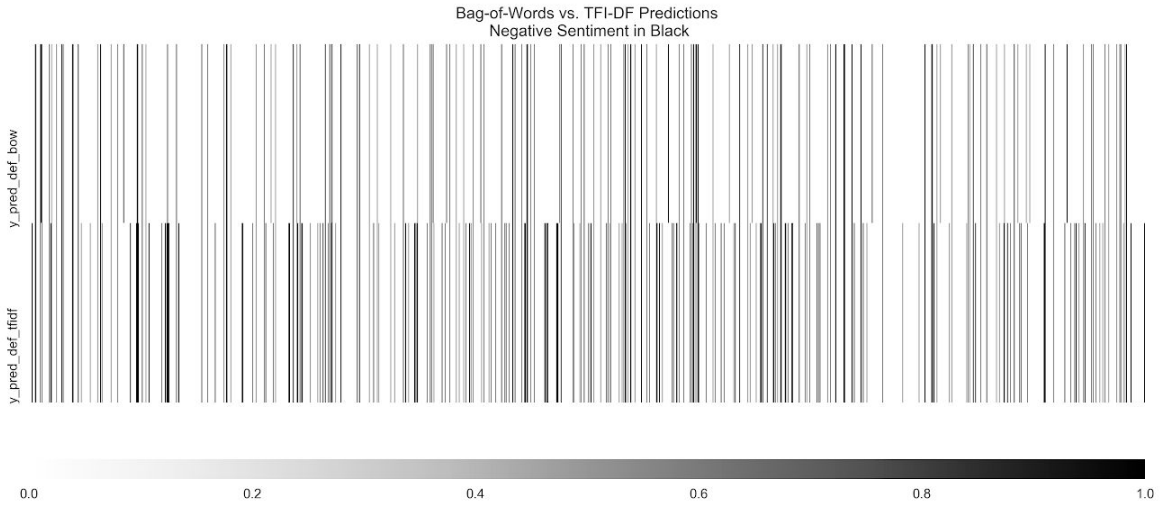
And, for contrast, here is a sample of low confidence accurate prediction of negative sentiment:

	document	score	y_pred_prob_def_bow_0
5	The rest of the movie lacks art, charm, meaning... If it's about emptiness, it works I guess because it's empty.	0	0.551179
8	A bit predictable.	0	0.596552
41	I wasn't the least bit interested.	0	0.599650
65	One character is totally annoying with a voice that gives me the feeling of fingernails on a chalkboard.	0	0.546862
105	Very disappointing.	0	0.599650

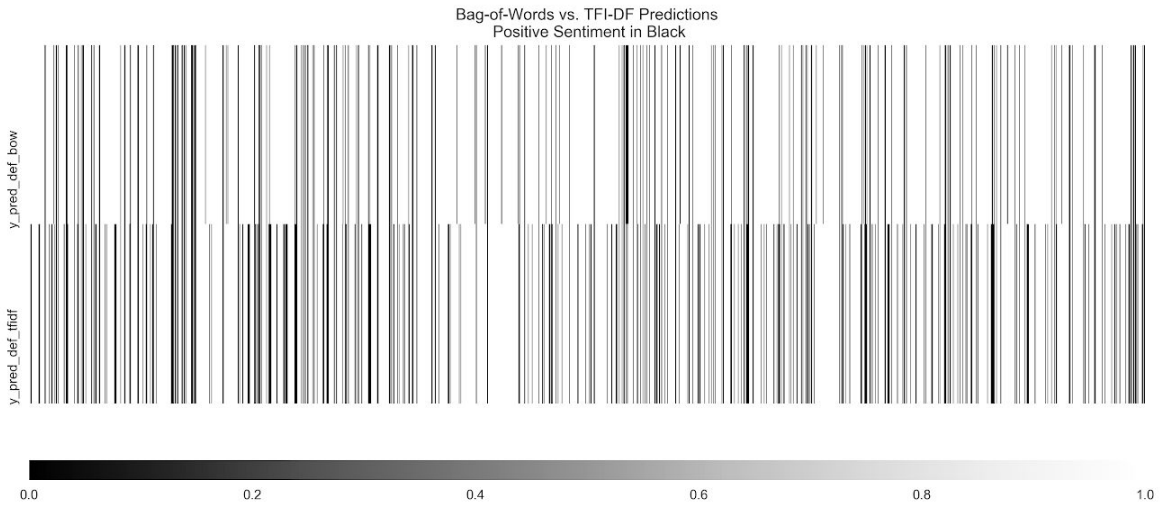
Visualize mistakes between models

With these two models established, we examined if there is overlap in the mistakes to the instance level. This is important since the model metrics are roughly equivalent and can give insight into whether or not it will be beneficial to combine the models.

Correct predictions are white and mistake in black. This quick visualization assists in checking if the mistakes are similar and the amount of overlap between mistaken predictions. The idea is that the models make mistakes in different ways, at different times, and for different texts, then this could be used to our advantage when modeling. We could simply use the highest class prediction from the two models for predicting a label of unseen text.



(Above: comparing incorrect negative sentiment prediction between Bag-of-Words and TF-IDF models)



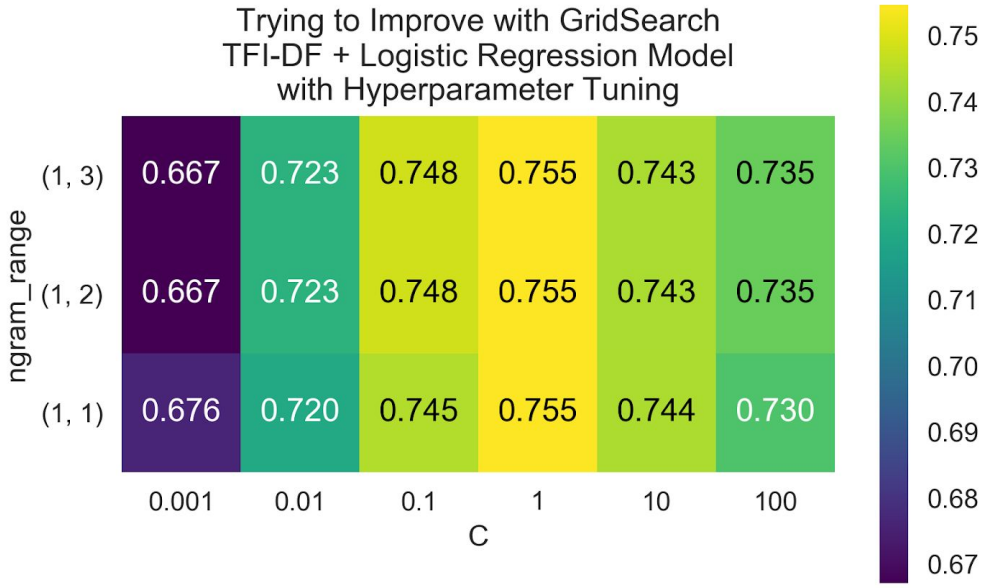
(Above: comparing incorrect positive sentiment prediction between Bag-of-Words and TF-IDF models)

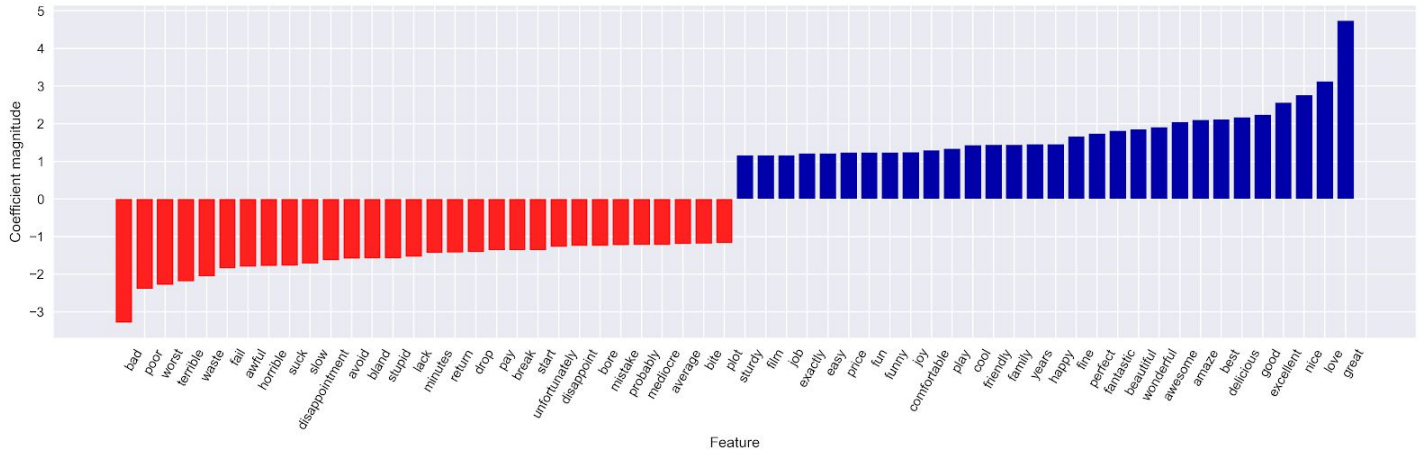
In summary above, if we combined both models and utilized the highest confidence prediction, we would be accurate more than 93% of the time. This number is corroborated through the actual implementation of the voting classifiers and demonstrate the improvements which can be achieved by combining two weak models if their strengths are complimentary (accurate in different areas and considering different aspects of the data).

Can we improve TfidfVectorizer + Logistic Regression? Not really...

Once the benchmark and baseline models were produced, refinements were applied via grid searching hyperparameters. Many of these overlapped between the Count-Based model and the TF-IDF Model.

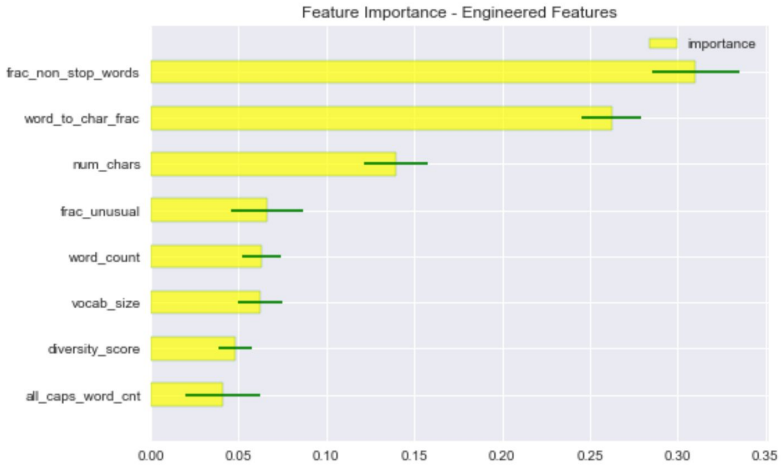
- Much of the processing in the baseline models were accomplished with **sklearn** preprocessing within the classes **CountVectorizer' & 'TfidfVectorizer**. If the performance were likely a result of the preprocessing such as the following, more time could be spent doing this in NLTK (which I did but didn't produce much better results with the manual preprocessing):
    - **Noise Reduction** - removing non-relevant text to the context of the data (stop words, URLs/links, social media entities, punctuations and industry specific words, etc.)
    - **Lexicon Normalization** - feature engineering step by converting versions of a word into a lemma, effectively reducing dimensionality
        - stemming - rule-based, removing suffixes
        - lemmatization - obtaining root form of word, using a vocabulary (standard lexical dictionary for importance of words) and morphological analysis (word structure and grammar relations)
- Next I will look at increasing performance by using more than 1 n-gram (2 and 3)
- In summary, the initial TF-IDF was the better performer.
- **Observation:** In general, the heatmap shows that there were slight gains for increasing model complexity. However, the best performing model was the one using only unigrams and the LinearRegression arg C of 1. The Best cross-validation score: 0.75 with the Best parameters: {'tfidfvectorizer__ngram_range': (1, 1), 'logisticregression__C': 1}



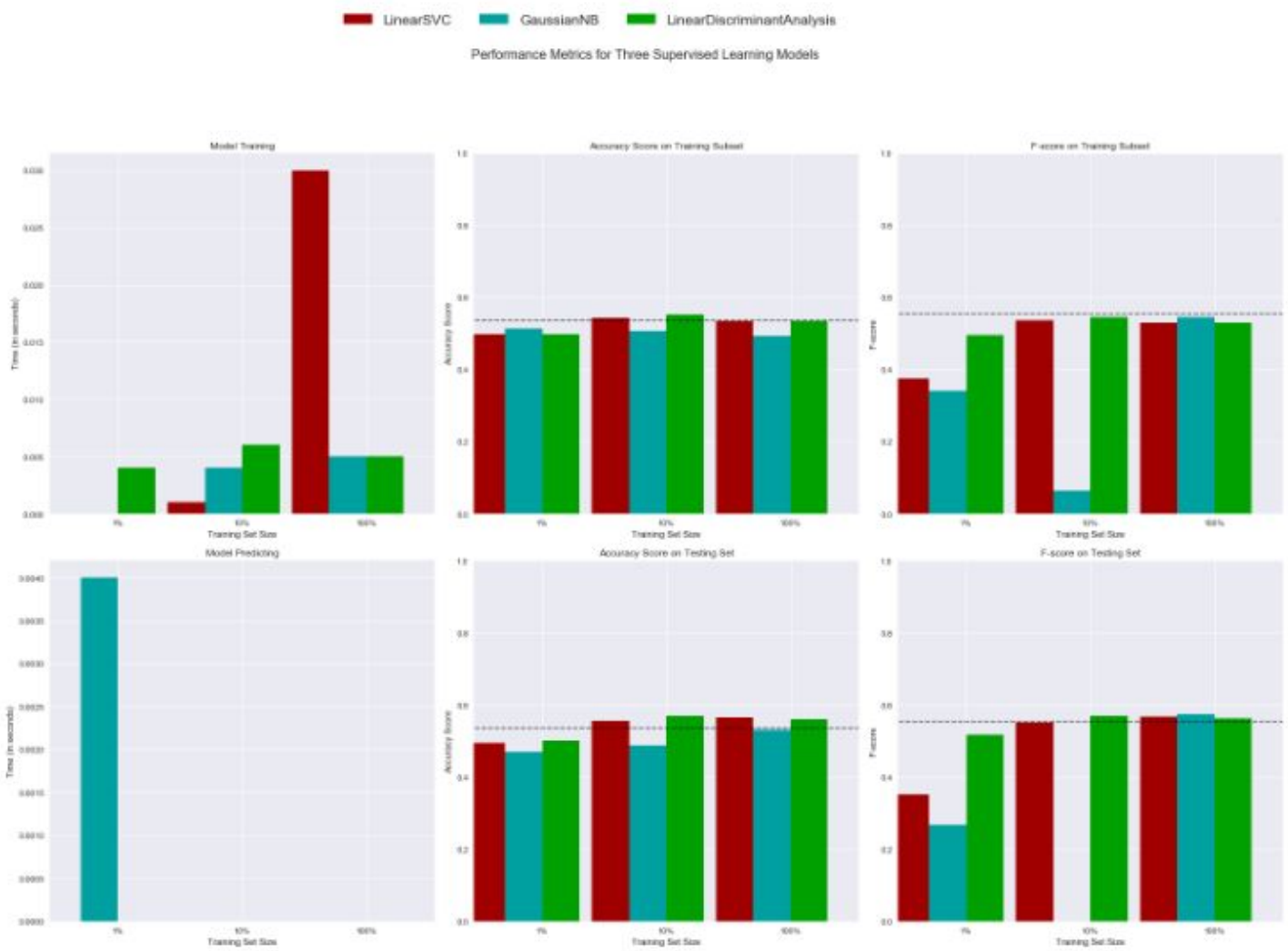


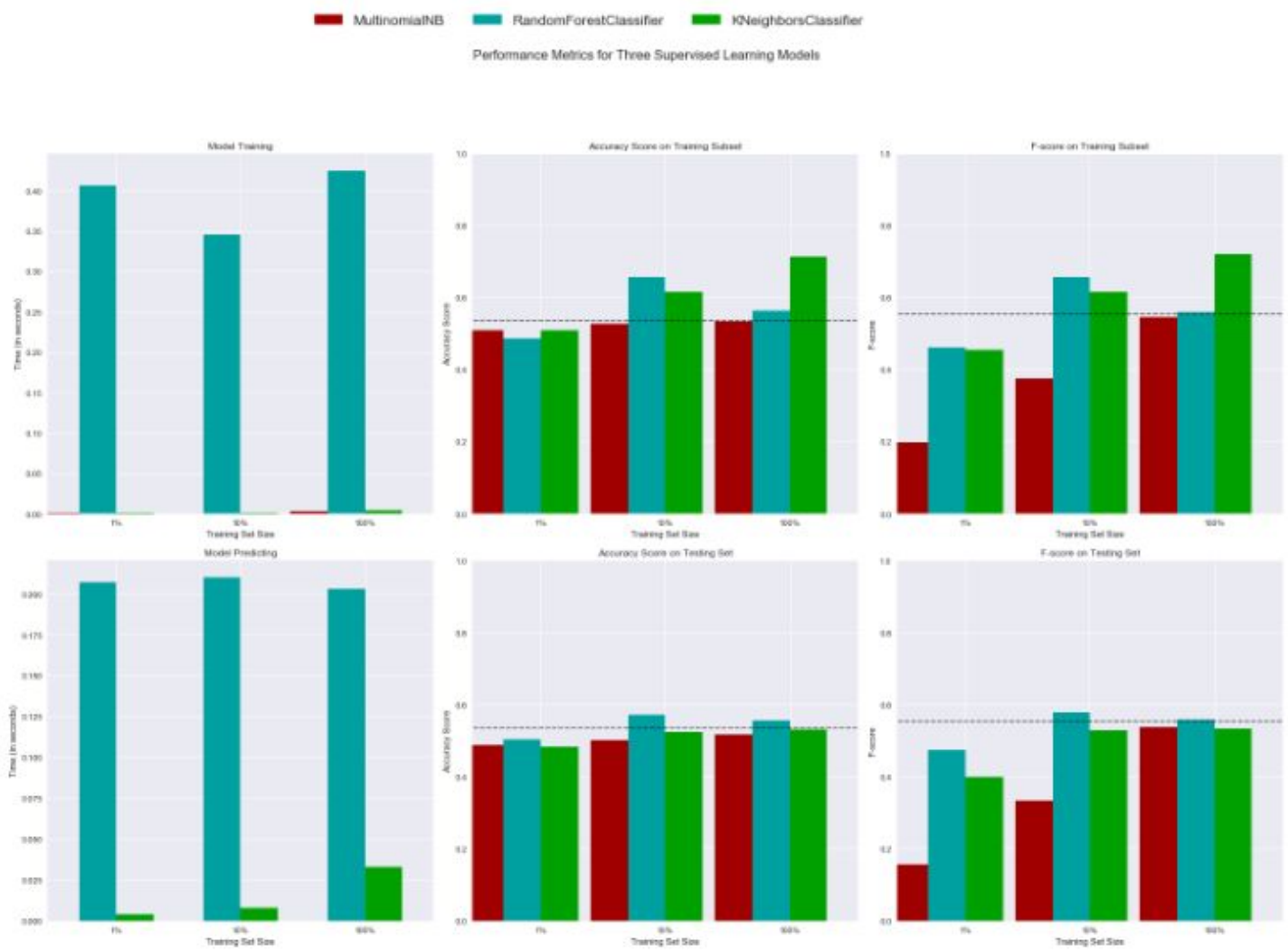
Various Models with Custom Engineered Features

This was an attempt to utized code learned in the class, to examine features designed to explore if anything could be gained from trying to capture some aspects of lost information due to preprocessing of text.



We examined accuracy, time to train and predict, results with varied data sampling, and using f1 score the training and testing predictions. Unfortunately, the results are slightly better than the Naive Model Predictor.





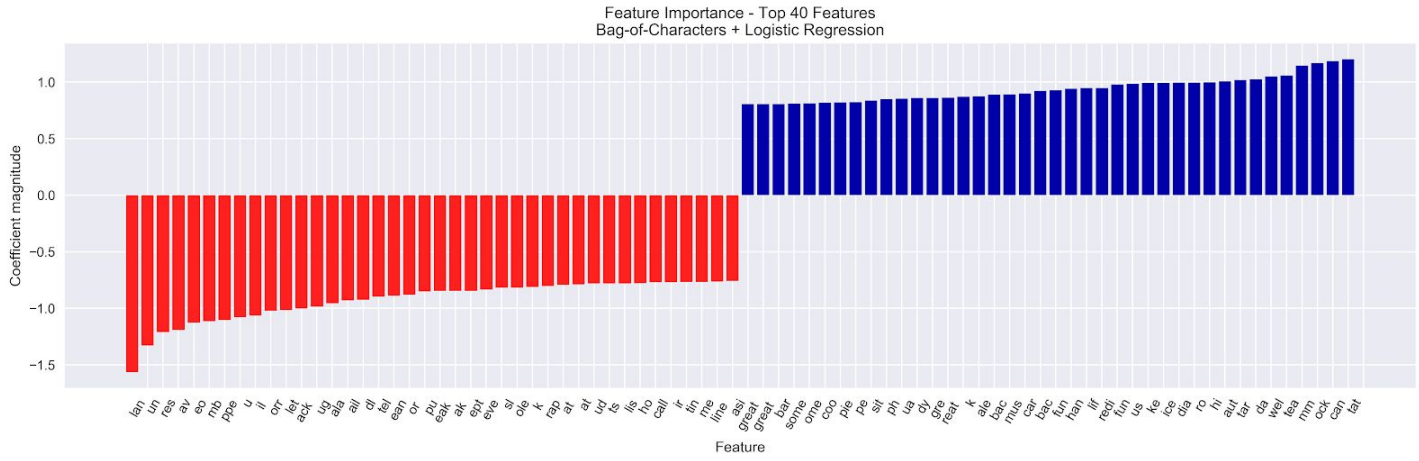
What about Bag-of-Characters + Logistic Regression? Really surprising!

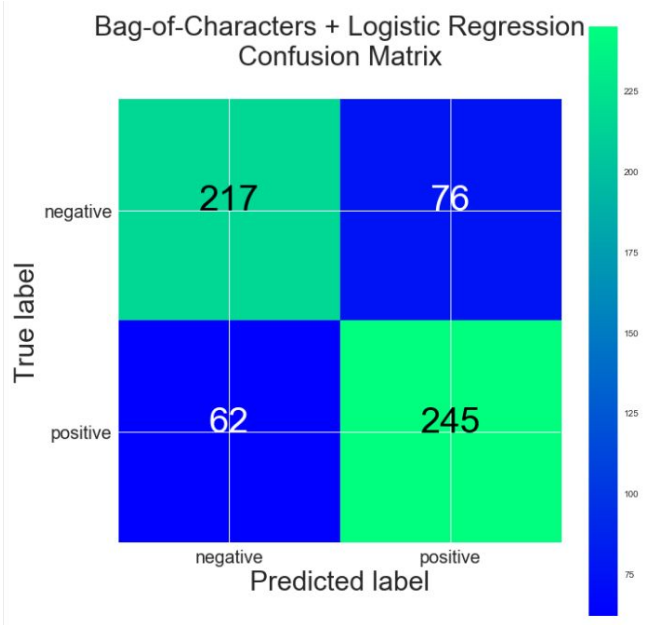
- Mean cross-validation accuracy: 0.77
- accuracy = 0.770, precision = 0.770, recall = 0.770, f1 = 0.770
- Bag-of-Characters was a surprise performer. The best performing model was with the ngram_range of (2, 10), a document frequency min/max of 0.01/0.80. Given the scores, this would be a contender for a generalized model offering.

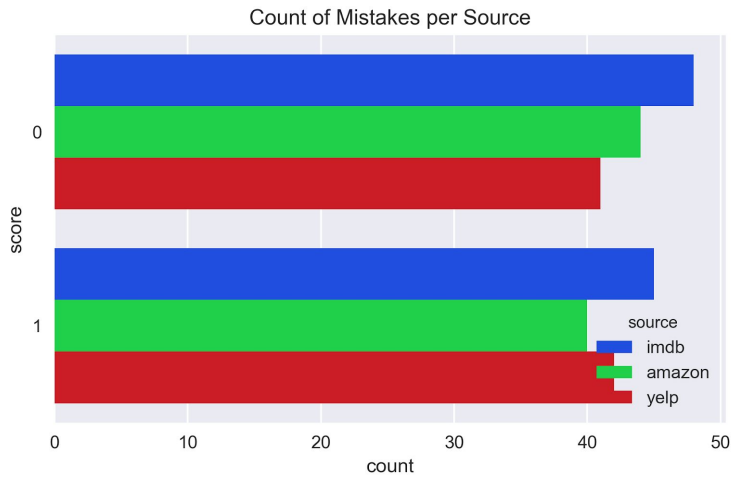
The Bag-of-Characters + Logistic Regression was a shock to me. The idea was sparked from seeing a paper on using Neural Nets trained on Shakespeare or stackoverflow.com and then generating text. I then thought about trying the CountVectorizer with character-level n-grams. Below are some example features, many are whole, albeit short, words:

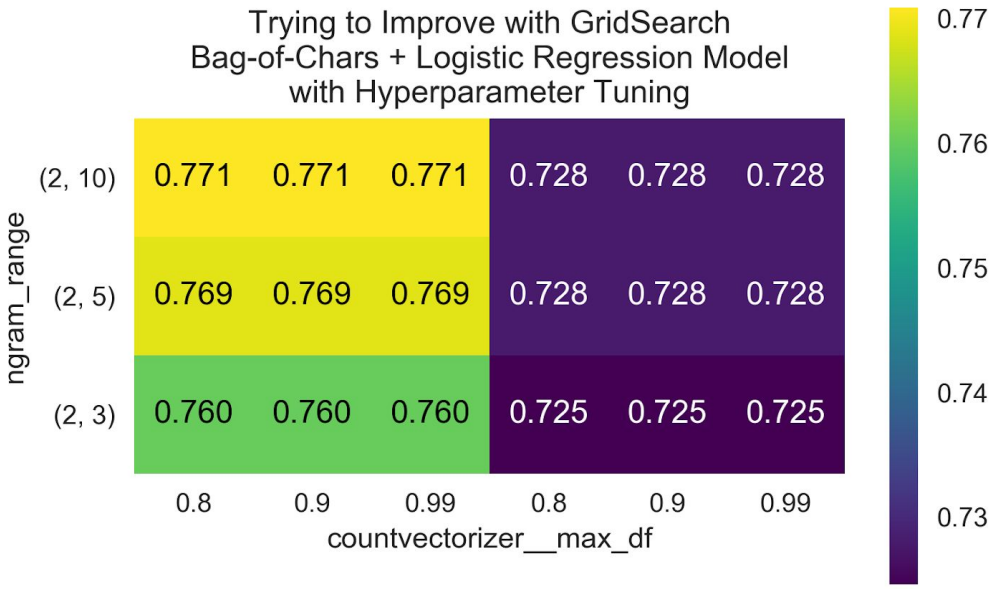
```
num features: 2523

features head: [' a', ' ab', ' ac', ' act', ' act ']

every 100th feature:
[' a', ' cont', ' foo', ' love', ' pretty ', ' str', ' worth', 'aracter ',
'break', 'day', 'eed', 'ervice', 'five ', 'hone ', 'ing ', 'lem', 'minute',
'nty', 'ort', 'play', 'recommend', 'ru', 'ste', 'tiv', 'usa', 'xperien']

features tail: ['ythin', 'ything', 'ything ', 'ze', 'ze ']
```

Model performance was consistent with/without preprocessing, and across n-gram ranges, min/max document frequencies, with/without stop words and more. Given these discoveries, a compact model is likely the better choice for multiple reasons.





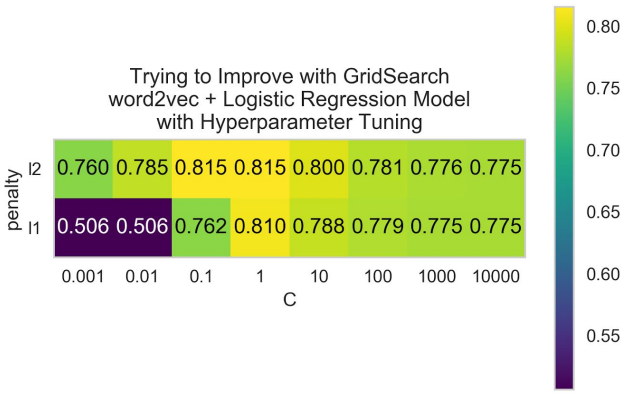




(Above: x-axis (green/yellow) is min_df = 0.01 & the other (purple) is min_df = 0.05)

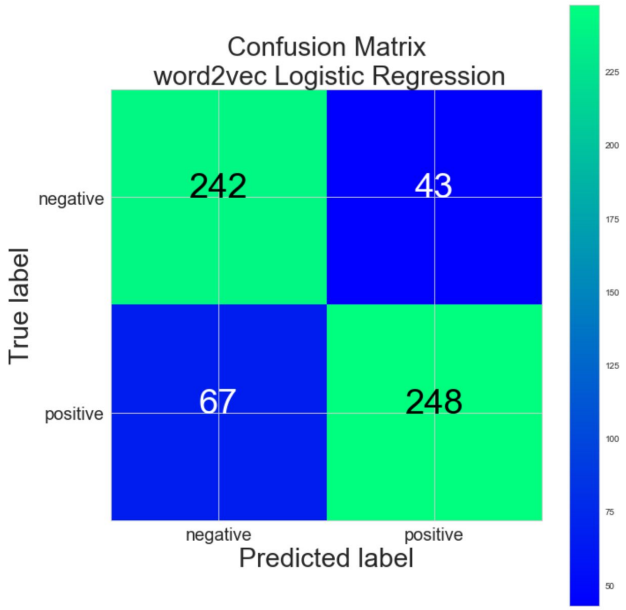
Gensim Implementation of Google's word2vec + Logistic Regression - Using learned embeddings

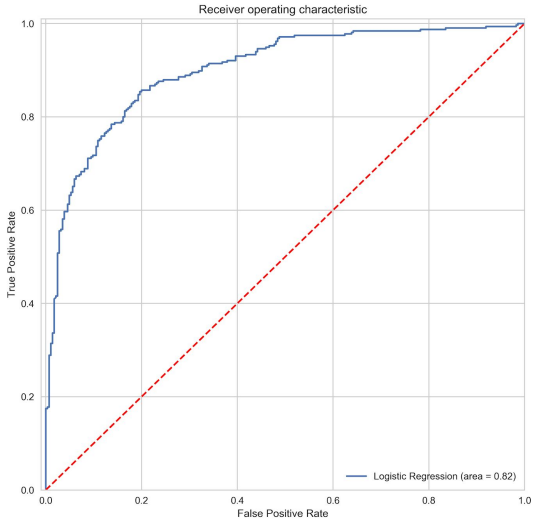
- accuracy = 0.770, precision = 0.770, recall = 0.770, f1 = 0.770
- Best cross-validation score: 0.80
- Best parameters: {'penalty': 'l2', 'C': 0.1}]
-
- the idea is that **CountVectorizer** & **TfdifVectorizer** capture the counts or a scaled vector of inverse document frequency which is essentially signalling on the presence of a word.
- without a large training dataset, a lot of semantic meaning can be missed
- give model data representation where words like ***greatest*** and ***best*** have similar vectors while words like ***best*** and ***worst*** have vectors that are further
- we will represent individual words and sentences as vectors, then adding the vectors together as well as taking the average
- word vectors can be visualized by reducing dimensionality with PCS, then projecting them onto 2D space (this loses information but the regular structure is still often observable)
- below will use **gensim** and **word2vec** to capture semantic meaning by way of distance metrics
- create the tokens from the custom tokenizer and normalization, gensim expects a list of tokens for a sentence/document



About Google's word2vec

- the word2vec model trained by Google on the Google News dataset,
- on about 100 billion words
- contains 3,000,000 unique phrases built with layer size of 300
- Note that the similarities were trained on a news dataset, and that Google did very little preprocessing there. So the phrases are case sensitive: watch out! Especially with proper nouns. There are typos and spelling errors as well.
- The ROC chart, bottom right, could be used to further refine the solution depending upon the need. For example, what is the price of being wrong on a prediction, or missing a prediction? The stakes would determine where to position the model along this curve.





Results

Model Evaluation and Validity

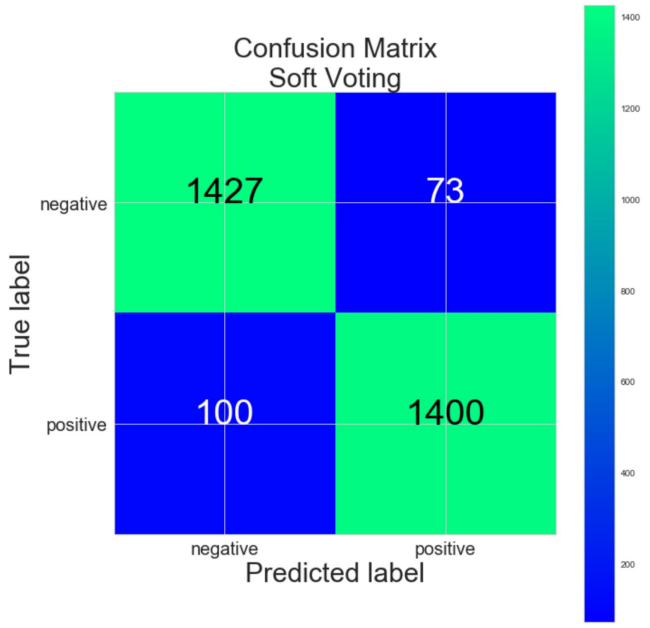
The Final Model(s)

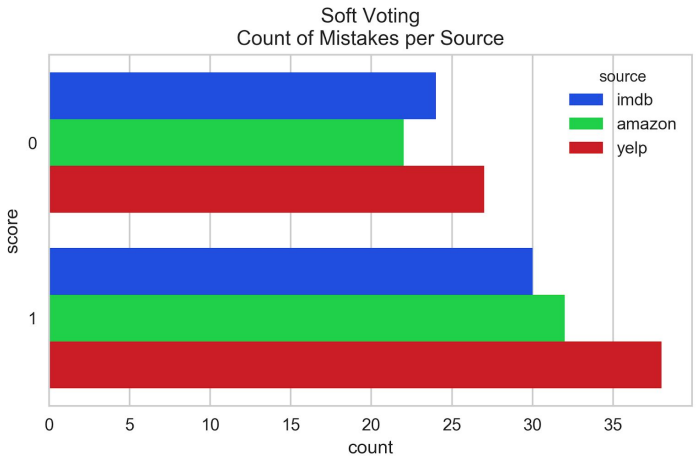
Both voting classifiers performed remarkably well. They were the combination of the best of class for three of the models explored (CountVectorizer (both word-level and character-level n-grams) and TfidfVectorizer. These were optimized using GridSearchCV.

- BoWs/CountVectorizer + LogisticRegression used original hyperparameters described earlier in project.
- TfidfVectorizer + LogisticRegression used original hyperparameters described earlier in project.
- BoCs/CountVectorizer + LogisticRegression described earlier in the project.
- Both hard and soft voting were tested with similar results. These results were spot on with the back-of-the-envelope estimate of implementing voting classifiers earlier in the paper (93% accuracy estimated).
- The variation in the mistakes is similar in number with the hard voting classifier having more of a balance between mistake counts between classes.

Soft Voting Classifier

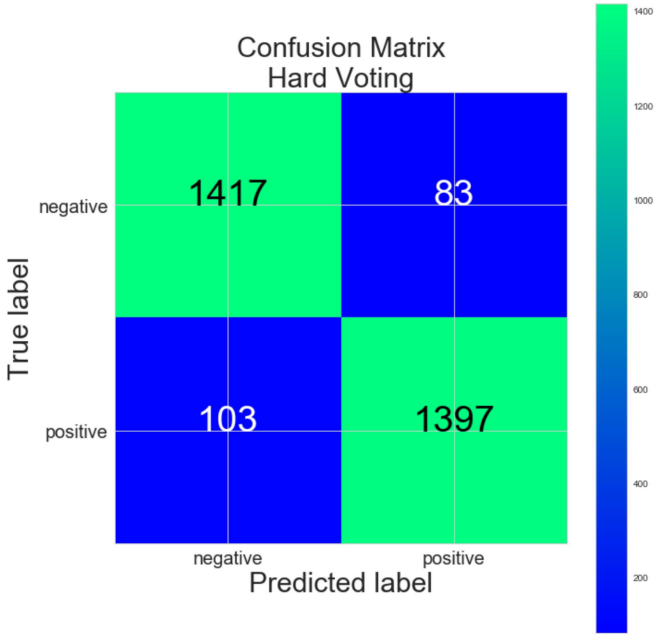
- accuracy = 0.942, precision = 0.942, recall = 0.942, f1 = 0.942

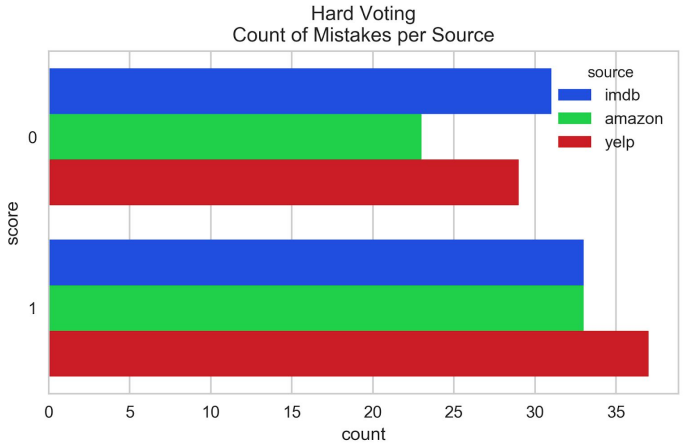




Hard Voting Classifier

- accuracy = 0.938, precision = 0.938, recall = 0.938, f1 = 0.938





Justification

The final results of both voting classifiers (93-94% accuracy) supersede the performance of the Naive models (~55% accuracy). Additionally, the voting classifier not only beats the benchmark default models, but it handily outperforms any single classifier (74-81% accuracy). Furthermore, no significant overhead in computation was needed to achieve the higher results.

Conclusion

Free-Form Visualization

Test Sentences for additional validation - from urbandictionary.com

I have collected a sample of sentiment-loaded sentences from urbandictionary.com. This is anecdotal but is a good spot check for out of sample performance. The red checks in the image below show the errors by the various best performing algos. It can be seen that the Bag-of-Characters has the least accuracy when applied to this sample. Also, interesting is the clustering of errors for indices 9 and 16. For index 9, the sentence could be construed as a little ambiguous. For index 16, this seems to be a clearly positive sentence, although it could be viewed as sarcasm, but it's challenge likely stems from a lack of context.

	document	score	bows_1	tfidf_1	w2v_1	bocs_pred	hard_vote_pred	soft_vote_pred
0	Expensive restaurants with horrible food that people still go to for the prime location or atmosphere.	0	0.054484	0.304571	0.317084	0	0	0
1	It has a beautiful view of the Eiffel Tower!	1	0.863899	0.851153	0.879514	1	1	1
2	Nah, that place is a total touristaurant, my food was awful last time I ate there.	0	0.067938	0.214175	0.230749	0	0	0
3	I had a great day, climbed a hill and didn't check facebook.	1	0.972061	0.920973	0.682593	1	1	1
4	That corgi-shaped macaron is expensive because it's got instagram tax.	1	0.450501	0.429538	0.383475	1	0	0
5	That 1962 Alfa Romeo is good for transportainment	1	0.748797	0.876521	0.752375	1	1	1
6	not so great for getting to work in the snow	0	0.954142	0.965835	0.724132	1	1	1
7	This is so sad Alexa play despacito	0	0.540381	0.577829	0.557572	1	1	1
8	This better not escalate to Dutch ovens any time soon...	0	0.402838	0.429672	0.301215	1	0	1
9	Oh. It has too many calories for me. Well, OK.	1	0.177230	0.383922	0.651511	0	0	0
10	Mmm, this pizza is some good food.	1	0.634286	0.620322	0.608796	1	1	1
11	Food's better for throwing at someone than eating.	0	0.415085	0.468051	0.211836	0	0	0
12	Are you finished with your restau-rant? I get it, we're outta here. Let's pay the check and leave.	0	0.054965	0.202283	0.305217	1	0	0
13	The Primrose Bar and Restaurant in Ballynahinch, Co. Down in the north of Ireland is one of the best in the country.	1	0.934805	0.813399	0.704198	0	1	1
14	This burger is crappy.	0	0.367896	0.393745	0.087171	0	0	0
15	DUDE! I got buffungled by Amazon.com with a 8-12 day shipping time!	0	0.573092	0.582170	0.495714	0	1	1
16	I am going to BUY that game RIGHT NOW.	1	0.458261	0.475028	0.442353	1	0	1
17	wow, that was great!	1	0.972877	0.941800	0.981993	1	1	1
18	That movie was great!	1	0.946260	0.960578	0.948775	1	1	1
19	The computer crashed again. I lost three thousand lines of code.	0	0.421927	0.407135	0.213776	0	0	0

Reflection

In this project, the NLP task of sentiment analysis was tackled. First, a balanced dataset was selected to facilitate ease of the solution. After the baselines were established with a Naive Model and Bag-of-Words + Linear Regression, additional models were explored like Bag-of-Character, Hand-Crafted Features with a ranged of models tested, TF-IDF + Logistic Regression and word2vec + Logistic Regression. The best of these was then refined through grid search and 5-fold cross-validation. Almost all of the non-Naive models were in the same ballpark regarding results. For the final model, a voting classifier was utilized, testing both hard and soft voting with results around 94% accuracy, which is sufficient to solve this problem.

Improvement

There are several areas to investigate for improved results. For starters, all of the models used were not state-of-the-art (SOTA). Different forms of Neural Nets could be considered like LSTMs and CNNs, as many forms of these have had good success recently and are well-enough documented to pursue experimentation. Additionally, these SOTA models could be combined with the traditional models explored in this project.

Additionally, Logistic Regression was chosen consistently to give a clear comparison between the count-based BoWs and BoCs, TF-IDF , and word-embedding models. It would be a natural next step to grid search over several models in addition to Logistic Regression and even testing a voting learner. Then, again applying the voting classifier for additional gains.

Even before building models, there are several other packages and techniques for preprocessing not explored here. Since my implementations were simple, there could be room for improvements by using Parts-of-Speech (POS) , Named Entity Recognition (NER), and topic modelling with techniques like LDA. Also, more advanced visualization could be used such as word clouds and t-SNE and PCA to build up intuition and insight. Finally more data or different data which has the intensity of the sentiment could provide further gains.

References

Practical/Books

- Müller, A., Guido, S. Introduction to Machine Learning with Python - A Guide for Data Scientists. O'Reilly Media, October 2016
- Géron, Aurélien. Hands-On Machine Learning with Scikit-Learn and TensorFlow - Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media, March 2017.
- Bird, S., Klein, E., Loper, E. Natural Language Processing with Python - Analyzing Text with Natural Language Toolkit. Web. 2018.

Research

- [pdf] Mikolov, T., Chen, K., Corrado, G., Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space.
- [pdf] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed Representations of Words and Phrases and their Compositionality. Accepted to NIPS 2013.

Sentiment

- [pdf] Mantyla, M., Graziotin, D., & Kuutila, M. (2016). The Evolution of Sentiment Analysis – A Review of Research Topics, Venues, and Top Cited Papers.

Interpretable Machine Learning

- [pdf] Finale, Doshi-Velez, & Been Kim (2017). Towards A Rigorous Science of Interpretable Machine Learning.

Introduction of the LSTM model:

- [pdf] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

Addition of the forget gate to the LSTM model:

- [pdf] Gers, F. A., Schmidhuber, J., & Cummins, F. (2000). Learning to forget: Continual prediction with LSTM. Neural computation, 12(10), 2451-2471.

More recent LSTM paper:

- [pdf] Graves, Alex. Supervised sequence labelling with recurrent neural networks. Vol. 385. Springer, 2012.

Benchmarking Sentiment Analysis

- [pdf] Abbasi, Ahmed & Hassan, Ammar & Dhar, Milan. (2014). Benchmarking Twitter Sentiment Analysis Tools. Language Resources and Evaluation Conference.
- Kurtulmus, Besir. "Benchmarking Sentiment Analysis Algorithms." Algorithmia, January 18, 2016, https://blog.algorithmia.com/benchmarking-sentiment-analysis-algorithms/.
- Mannes, John. "OpenAI sets benchmark for sentiment analysis using an efficient mLSTM." TechCrunch, April 7, 2017, https://techcrunch.com/2017/04/07/openai-sets-benchmark-for-sentiment-analysis-using-an-efficient-mlstm/.