I. Definition

Project Overview:

Sentiment analysis (also known as Opinion Mining or Emotion AI) is an interesting application of Natural Language Processing (NLP). I have chosen this subject for my capstone project as it is a powerful tool that can be employed to gauge customer opinion about pretty much everything: a new product, public policies, movies, etc. This can then be used to shape company strategy. Not only is this a very interesting technical challenge, its application is wide open.

Early works in sentiment analysis include Pang [1] and Turney [2]. Historically, topic-based text categorization is a precursor to the sentiment analysis [1]. Sentiment analysis can be imagined as a special case of topic-based categorization [1], with the two topics being 'positive sentiment' and 'negative sentiment'. However, special techniques need to be developed for sentiment analysis because of the linguistic nuances involved i.e. negations, sarcasm, etc.

Datasets for this project have been taken from UCI dataset repository:
https://archive.ics.uci.edu/ml/datasets/Sentiment+Labelled+Sentences#

This dataset was created for the Paper [3]. It contains reviews labelled with positive or negative sentiment.
Score is either 1 (for positive) or 0 (for negative).
The reviews come from three different websites/fields:

imdb.com
amazon.com
yelp.com

For each website, there exist 500 positive and 500 negative reviews. Those were selected randomly for larger datasets of reviews. The authors attempted to select reviews that have a clearly positive or negative connotation, the goal was for no neutral reviews to be selected.

Problem Statement:

Problem consists of classifying the sentiment polarity (positive or negative) of customer reviews. A successful algorithm will classify the customer reviews with high accuracy. Data will preprocessed by converting each review into a list of lower case words. Next preprocessing step will apply stemming on this data. Final preprocessing step will convert the lists of words into feature vectors using TF-IDF (Term Frequency-Inverse Document Frequency) [4]. This generates feature vectors of words that are proportional to their frequency in a given review and inversely proportional to their total frequency in the corpus. This step also removes stop words from reviews, as implemented in `TfidfVectorizer` function in sklearn. After partitioning the data into training set and test set, we will train multiple supervised machine learning models using the training set. We will then document and compare the performance of these algorithms (or models) on the test set and recommend the one with the best performance.

Evaluation Metrics:

As the dataset is balanced, accuracy will be used as evaluation metric for this project,

Accuracy = (true positives + true negatives) / dataset size

Where,

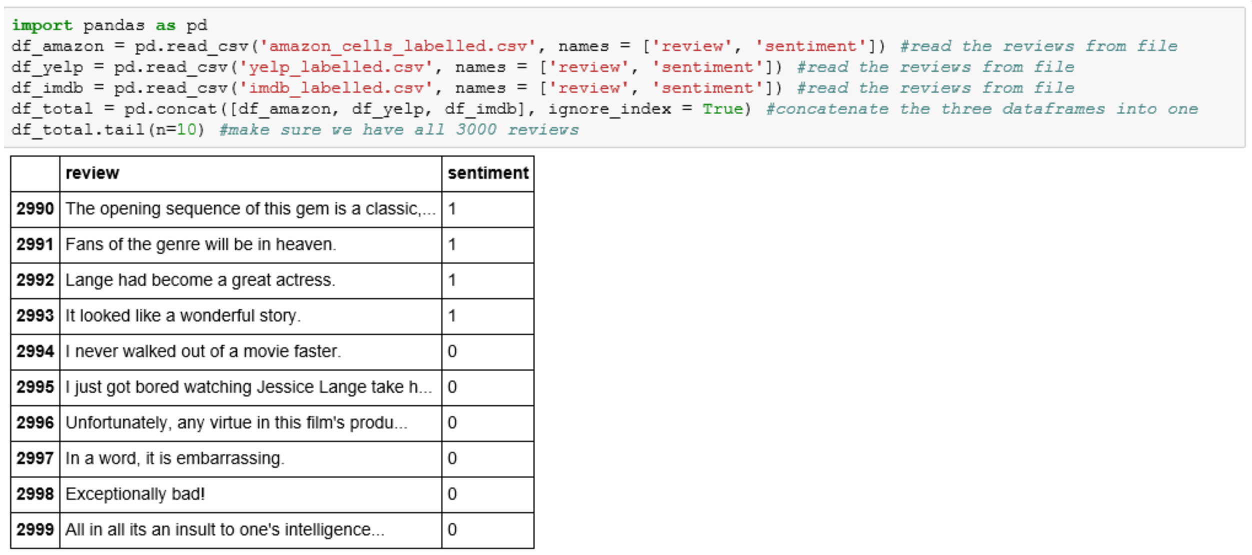
dataset size = true positives + true negatives + false positives + false negatives

False positives and false negatives incorrectly determine the polarity of the review's sentiment, and degrade accuracy.

II. Analysis

Data Exploration:

Input data came in three files: 'amazon_labelled.txt', 'yelp_labelled.txt' and 'imdb_labelled.txt'. On exploring these files, we noted that the review text (features) and sentiment (labels) were separated by "tab". Therefore, we tried to read the data using *pd.read_csv* and *sep* = *'\t'*. However, we failed to read all reviews correctly. We were seeing less entries than expected, as delimiter was unable to properly separate the reviews. We then read the files in Microsoft Excel, converted them to CSV format and then read them via *pd.read_csv* and default *sep*. This way, we are able to successfully read all the reviews (3000 total, 1000 per file) and their labels. Afterwards, we concatenated the three dataframes (one from each file) into one dataframe:



```
import pandas as pd
df_amazon = pd.read_csv('amazon_cells_labelled.csv', names = ['review', 'sentiment']) #read the reviews from file
df_yelp = pd.read_csv('yelp_labelled.csv', names = ['review', 'sentiment']) #read the reviews from file
df_imdb = pd.read_csv('imdb_labelled.csv', names = ['review', 'sentiment']) #read the reviews from file
df_total = pd.concat([df_amazon, df_yelp, df_imdb], ignore_index = True) #concatenate the three dataframes into one
df_total.tail(n=10) #make sure we have all 3000 reviews
```

	review	sentiment
2990	The opening sequence of this gem is a classic,...	1
2991	Fans of the genre will be in heaven.	1
2992	Lange had become a great actress.	1
2993	It looked like a wonderful story.	1
2994	I never walked out of a movie faster.	0
2995	I just got bored watching Jessice Lange take h...	0
2996	Unfortunately, any virtue in this film's produ...	0
2997	In a word, it is embarrassing.	0
2998	Exceptionally bad!	0
2999	All in all its an insult to one's intelligence...	0

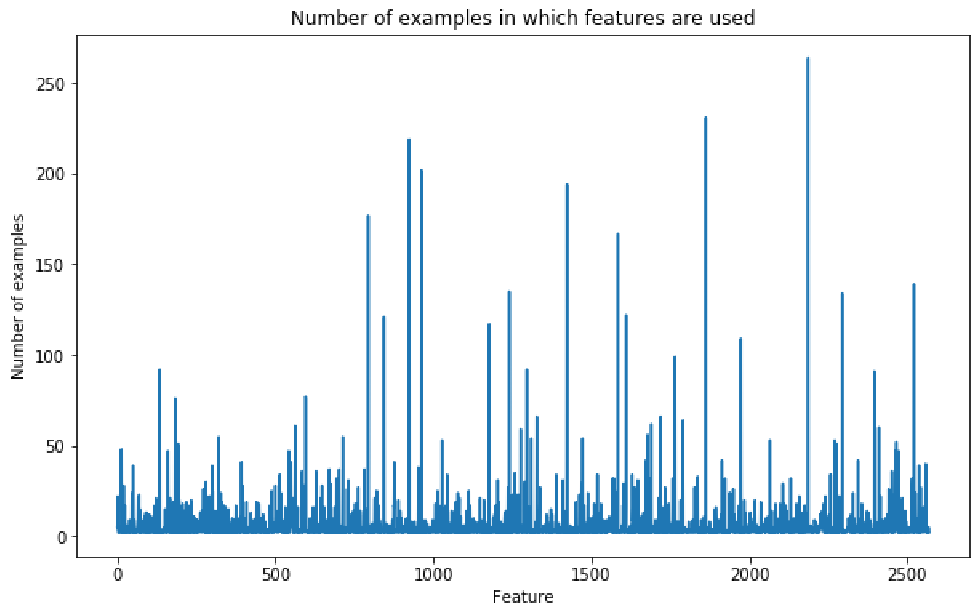
The features are extracted from data after some preprocessing steps, including Term Frequency-Inverse Document Frequency (TF-IDF). These preprocessing steps are discussed under Methodology section. We went on to explore the preprocessed data. Each feature represents a stemmed word or combination of two stemmed words. We retained only those features which are used in at least two examples. The dimension of the preprocessed data comes out to be 3000 x 2570, i.e. 3000 examples and 2570 features. Here are some interesting statistics about the data:

```
Shape of the TF-IDF transformed data is: (3000, 2570)
Number of nonzero elements in the TF-IDF matrix is: 17327
This represents only 0.224734111543% of entries in matrix
Among the non zero entries, Min: 0.105609921309, Max: 1.0, Mean: 0.386622708681, Std: 0.15022154459
An example review in transformed space:
  (0, 2065)	0.303743518298
  (0, 234)	0.360915754887
  (0, 2410)	0.296365014582
  (0, 1027)	0.427419380975
  (0, 193)	0.306028099206
  (0, 2506)	0.447787461729
  (0, 2066)	0.461295280141
In this example, only 7 features out of 2570 have nonzero values
```

As expected, the data represents a sparse matrix, i.e. the matrix whose majority of elements are zero. This is because any word or two word combination that is present in a review is not likely to appear in many other reviews. Only 0.2% of entries are nonzero. The situation would change if we increase the number of reviews. Which will subsequently help alleviate overfitting and improve testing accuracy.

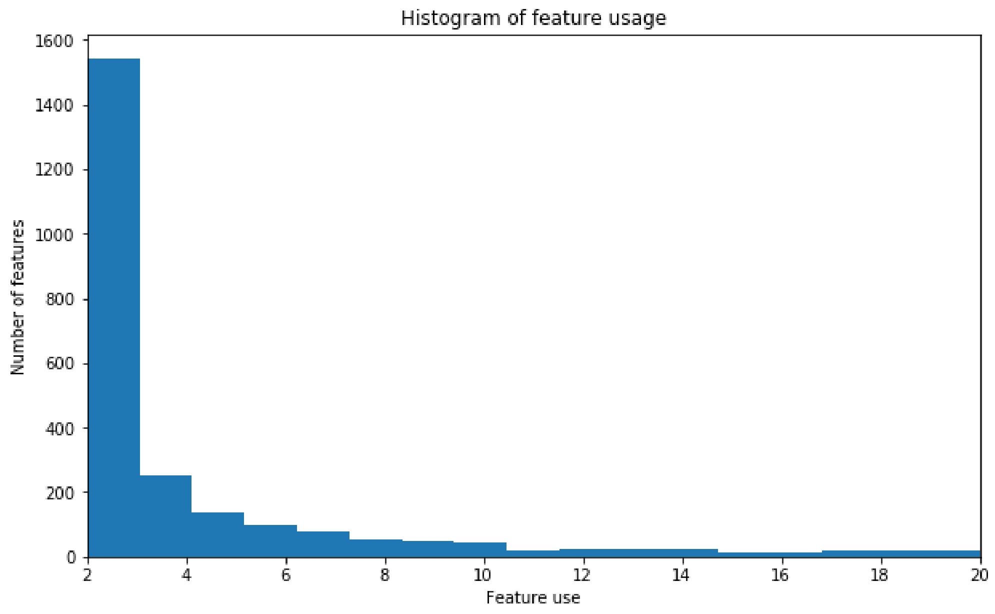
Exploratory Visualization:

Keeping in view the nature of data, i.e. sparse matrix, we went ahead and computed the number of examples in which each feature is used. The example in which a feature is used, will have a nonzero value for that feature. We produced two visualizations to help understand the data better. The first one shows the feature number on X-axis and the number of examples it is used in, on Y-axis:



This figure shows that the majority of the features are used in less than 50 examples.

To look at it in another way, we plotted a histogram which shows the number of examples features are used in, on the X-axis and the number of features which fall into that category on Y-axis:



This plot shows that a vast majority of features (>58%) are used in only 2 examples. And the number of features drops exponentially as we increase the occurrence requirement.

Algorithms and Techniques:

We will apply four supervised machine learning algorithms, which are consistent with size of data (2400 training examples, not too large). The three algorithms are:

1.) Logistic Regression

Logistic Regression is a linear classification model, i.e. it draws a linear decision boundary between the classes. The difference between linear regression is that it uses logistic function (or sigmoid function) as its activation function.

2.) Naive Bayes, natural choice as we are dealing with NLP

Naïve Bayes is a fast and efficient classification algorithm which is used widely in Natural Language Processing. Even though it assumes independence between features, it has proven to work in many practical situations. For this problem, we chose a Multinomial Naïve Bayes (a Naïve Bayes classifier with the assumption that the conditional probability distribution of its features is Multinomial distribution)

3.) Random Forest

Random Forest is an ensemble learning algorithm, which is based on the concept of bagging i.e. bootstrap aggregating. Ensemble learners consist of several uncorrelated weak learners, which in the case of Random Forests are decision trees. The final decision is the majority vote of all the decision trees.

4.) Support Vector Machines

Support Vector Machines (SVM) is a class of supervised machine learning algorithms that is used on classification problems. It separates the different classes with a linear decision boundary that maximizes the margin from the classes. Even though it provides good results, it is a resource intensive algorithm and doesn’t scale well with large datasets.

To fine tune the models, we will use multi-fold cross validation (GridSearchCV), i.e. taking an average of accuracy, while holding out a fold as validation set every time. Finally, the fine-tuned models will be used to find testing accuracy. Final model selection will be done based on test accuracy.

Benchmark Model:

The benchmark model that we are going to use for this project is the one that always returns a 1, meaning the review is positive. This benchmark model achieves an accuracy of 50% as only 50% of reviews are positive. However, in literature other baseline models have been used. [1] For instance, uses a model that manually captures the features (words, punctuations) from reviews and uses these to classify the reviews. This model achieves an accuracy of 69%.

III. Methodology

Data Preprocessing:

Data preprocessing is done in following four steps:

1. Substitute any character other than English alphabets (small or capital) with the space character (‘space’)

2. Substitute all upper case letters with lower case ones

3. Convert each review into a list of words

4. Apply snowball stemming algorithm on all words, with the exception of stop words. This requires downloading NLTK corpus (http://www.nltk.org/data.html) as we need a list of stop words

5. Append the stemmed list of words back into review form, joined by space characters. Make a list of preprocessed reviews. The dimension of this list will be *3000 x 1*

6. Apply TF-IDF (Term Frequency-Inverse Document Frequency) transformation on the reviews (list of 3000 reviews, dimension: *3000 x 1*), while removing the stop words. TF-IDF assigns a weight to each word in the review, that is proportional to its frequency (occurrence) in that review (Term Frequency) and inversely proportional to its frequency in the entire corpus. This transformation converts the list of reviews into a matrix of real numbers, whose dimension will be *3000 x number of features*. Number of features depends on the parameters we pass to the TF-IDF function. In our case, by using only up to 2 n-grams, and using *min_idf = 2*, i.e. retaining features who are used in at least two reviews, we get 2570 features.

7. Finally, we split the data (both reviews or features and sentiments or labels) into training (80%) and testing (20%) sets, while preserving the ratio of 1s and 0s in both sets.

Implementation:

We applied four supervised machine learning algorithms on the preprocessed data:

1. **Logistic Regression**

   We trained a Logistic Regression model using training data, and used it to predict class labels on testing data. After declaring the model with chosen parameters, we fit the model with training data. Training time is 0.063sec.

   We got an accuracy of 92.8% on training data, whereas we achieved an accuracy of 82% on testing data. This represents a variance or overfitting problem, which can be addressed by more data. However, we get much better performance than the benchmark model (50%).

2. **Naïve Bayes**

   We trained a Multinomial Naïve Bayes classifier on training data and measured its performance on testing data. Training time was 0.003sec, which is an order of magnitude better than the Logistic Regression classifier. Training set accuracy is 92.5% and testing set accuracy is 82.5%. We also notice the overfitting problem here.

3. **Random Forest**

   A random forest model was trained using the training data, using the default settings and a random state. Then, we noted the training time, training accuracy and testing accuracy. Training time is 0.96sec, which is roughly 3 orders of magnitude higher than the Naïve Bayes classifier. Training accuracy is 96.4%, whereas testing accuracy is 79.8%. This is not surprising as Random Forests are known to have overfitting problem.

4. **SVM**

   We trained an SVM model using training data, and then predicted the output labels with the help of this model. We saw a training time of 0.6sec, the training accuracy of 62.6% and testing accuracy of 62%. We note that the default model has poor training and testing accuracy than the previous models. In the next steps, we will refine the model.

Refinement:

1. **Logistic Regression**
   We refined '*C*', the inverse regularization parameter, for logistic regression. We tried values of *C: [1, 10, 100, 1000]* that cover a wide range. GridsearchCV function was used for this purpose with 20 cross validation sets. It was found that the default value: *C = 1* still gives the best result. Hence, refinement did not improve the performance over the original model.

2. **Random Forest**

   After trying several parameters, we found that '*n_estimators*' i.e. number of decision trees was the most important parameter to tune. The testing accuracy increased from 79.8% to 80.3% by using *n_estimators = 200*. However, we still recommend using default value of 10 as the slight improvement (0.5%) in accuracy does not justify the increased training time.

3. **SVM**

   We tuned '*gamma*' parameter of SVM to get the optimum performance. As a result, we noticed the testing accuracy increase from 62% to 81.8%.

IV. Results

Model Evaluation and Validation:

We chose Naïve Bayes classifier as the final model for our problem. It gets the best accuracy on testing data (82.5%) with the least time required for training (3ms for 2400 training examples).

We have noticed that we cannot push past the 82%-83% testing accuracy, while using an array of supervised machine learning algorithms. This shows that there is something fundamentally wrong, which cannot be overcome by the algorithms. For this model to generalize well, we see two roadblocks:

1. As we noticed in data exploration and visualization sections, majority of features (words or word combinations) are only used in couple of reviews. There is a nonzero probability that both of these reviews fall in testing data. In that scenario, the model has not learnt how to treat this feature and will be prone to error. To get rid of this issue, we need more data and then enforce the model to use only those reviews for which all the features are used in, let's say, at least 10% of reviews.
2. Second issue that might arise is negations. Since, we remove stop words, some negation words (for example: not) will be removed. This creates confusion for the algorithm, as it will see some examples where 'good' which was originally 'not good' will be treated as negative, whereas some examples where 'good' will be treated as positive.

One good generalization test would be to take reviews from an unrelated corpus (or website) and see how the model classifies them. There is a chance that different kind of vocabulary is used for different products/websites and the model trained on one cannot generalize on others.

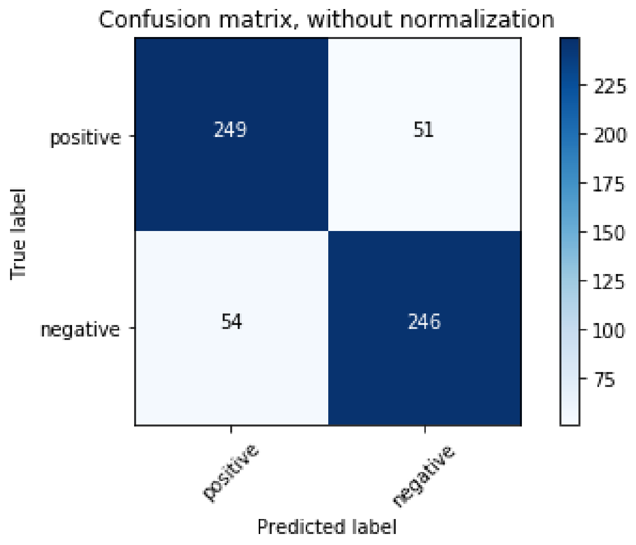
Justification:

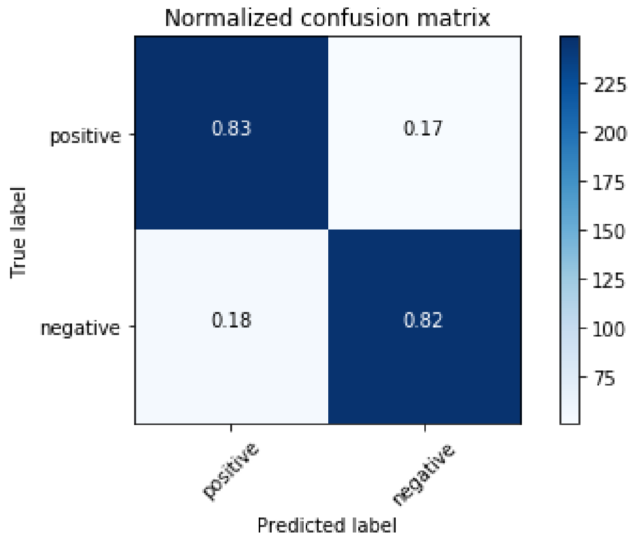
Our final model achieves 82.8% accuracy on testing data, whereas the benchmark model only gets 50% accuracy. This is an improvement of more than 65% over the benchmark.

V. Conclusion

Free-Form Visualization:

For free-form visualization, we decided to work on confusion matrix. The idea is to analyze if there is any one class which is being hit more by misclassification. We borrowed code from scikit-learn.org [5] to compute and plot confusion matrix. Here are the results:





It becomes clear that both the classes (positive and negative) are almost equally impacted by misclassification. Hence, none of the class is more vulnerable or sensitive to being wrongly classified, and no further analysis is necessary in this direction.

Reflection:

In this project, we started with a dataset that gave us customer reviews from three different popular websites and their labels i.e. the sentiment polarity, 0 if the review is negative and 1 if the review is positive.

First of all, we cleaned up the reviews, i.e. removed all characters except English alphabets. Then we applied stemming, and subsequently TF-IDF transformation. This was an interesting aspect, as this determined how many features we will get. The problem with transformation is that it represents words with real number and then it becomes hard to imagine the words behind the real numbers.

After splitting the data into training and testing sets, we train four supervised machine learning models on training data, noting their performance (accuracy) on training and testing data. Next, we refine the models by tuning the hyper parameters using GridsearchCV. Finally, the best model is chosen using the criteria of accuracy on testing data.

We get a testing accuracy of 82.8% by training only on 2400 examples. Not only we have small number of training data, the reviews mostly consist of one sentence. Having longer reviews will help in improving accuracy.

The most fun part in the project was feature engineering and the related visualizations. Much of feature engineering work was done during TF-IDF transformation. We had to play with number of n-grams to consider and finding the *idf_min* i.e. minimum number of examples a feature should be used in. We noticed that changing *min_idf* from 1 to 2 reduced the feature space from 16174 to 2570, i.e. reduction of 84% without compromising accuracy. The visualizations gave us valuable insight into how the features are being used in the examples.

The most challenging part was dealing with sparse matrix. A sparse matrix is one whose majority of data is zero. This is an interesting representation of data, it only stores and values and corresponding indices that are nonzero. We had to research and use special methods to deal with this kind of data representation.

Improvement:

1. Improvement can be achieved by performing an error analysis. This will give us an insight into what kind of reviews are not being classified correctly. Most probably, there are some missing words in the vocabulary generated by the training data. We can manually add more features (missing words) into the model to improve its accuracy.
2. We can apply deep learning to solve this problem. However, if the problem doesn't lie in learning, but missing vocabulary, then this is not going to help
3. Use Google's word2vector open source model

[1] Pang, Bo; Lee, Lillian; Vaithyanathan, Shivakumar (2002). "Thumbs up? Sentiment Classification using Machine Learning Techniques". *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. pp. 79–86.

[2] Turney, Peter (2002). "Thumbs Up or Thumbs Down? Semantic Orientation Applied to Unsupervised Classification of Reviews". *Proceedings of the Association for Computational Linguistics*. pp. 417–424. arXiv:cs.LG/0212032

[3] 'From Group to Individual Labels using Deep Features', Kotzias et. al,. KDD 2015

[4] https://jessesw.com/NLP-Movie-Reviews/

[5] http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html#sphx-glr-auto-examples-model-selection-plot-confusion-matrix-py