Machine Learning Engineer Nanodegree

Capstone Project on Sentiment Analysis

Simon Phua
August 27, 2017

DEFINITION: PROJECT OVERVIEW

The advent of mobile technologies and social media has given rise to new opportunities for companies to engage consumers. One challenge faced by companies is effectively managing the swathes of feedback they receive. Of particular concern is negative feedback, that if not addressed quickly, could go viral and damage a company's reputation.

Presently, social media managers sift through scores of tweets and posts one by one to identify negative sentiments. For popular brands, feedback could number in the hundreds and thousands on a daily basis.[1] Social media managers need help to identify negative sentiments more quickly and more effectively. Delays in addressing negative publicity could cost companies countless additional dollars in marketing and public relations spend.

DEFINITION: PROBLEM STATEMENT

This project explores the use of machine learning to train a sentiment analysis engine to make predictions on whether the sentiment of a small body of text is negative. This will enable social media managers to more quickly identify negative feedback and respond sooner.

This is a text-based classification problem. The proposed model will take as input a small body of text, such as a Facebook post or a tweet. Based on the words found in the body of text, the model will make a prediction on whether the sentiment is negative.[2]

Research materials on the field of text-based sentiment analysis are referred to in the footnotes. [3]

[1] On some platforms, users leave a rating or a score with their feedback, e.g. Yelp, AirBnB, and Amazon, so social media managers can quickly identify negative comments. On other platforms, e.g. Twitter, LinkedIn and Facebook, no such rating mechanism exists. This latter group poses a greater challenge for social media managers in identifying negative sentiments.

[2] The hypothesis underlying this project is that the collection of words found in a given body of text can be used to determine its sentiment.

[3] *'SentiBench – a benchmark comparison of state-of-the-practice sentiment analysis methods'*, by Filipe N Ribeiro, Matheus Araújo, Pollyanna Gonçalves, Marcos André Gonçalves and Fabrício Benevenuto (https://epjdatascience.springeropen.com/articles/10.1140/epjds/s13688-016-0085-1)
'Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank', by Richard Socher, Alex Perelygin, Jean Y. Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng and Christopher Potts (https://nlp.stanford.edu/~socherr/EMNLP2013_RNTN.pdf)
'NRC-Canada: Building the State-of-the-Art in Sentiment Analysis of Tweets', by Saif M. Mohammad, Svetlana Kiritchenko, and Xiaodan Zhu (http://aclweb.org/anthology//S/S13/S13-2053.pdf)
'Detection and Scoring of Internet Slands for Sentiment Analysis Using SentiWordNet', by Fazal Masud Kundi, Shakeel Ahmad, Aurangzeb Khan and Muhammad Zubair Asghar

This project will explore utilising a Naïve Bayes classifier, ensemble classifiers and a Neural Network to build the engine. These algorithms are well suited to handle non-linear classification problems with large feature sets.

This engine will be trained on the Amazon Fine Food Reviews dataset[4], comprising 568,454 reviews for food products on Amazon up until October 2012. This dataset was selected for the following reasons:

- Large number of unique reviews (including a large variety of unique reviewers, words used in the reviews and products reviewed) makes this dataset suitable for a variety of machine learning techniques, and suggests it could generalise well;
- Verbatim text of the reviews comprises a wide-ranging vocabulary, which could be the basis for building a sentiment analysis that generalises;
- Reviewers assign a score for each review, which provides an effective way to determine if reviews are positive or negative; and
- Reviews were submitted relatively recently which minimizes bias due to evolving language/lingo over time.

DEFINITION: METRICS

I propose the use of accuracy and recall as metrics to measure the performance of the model.[5]

Accuracy is defined as the proportion of correct predictions out of all predictions made. It is a good overarching measure of the model's performance. It is given by the following formula:

$$\frac{\text{(true positives + true negatives)}}{\text{(true positives + true negatives + false positives + false negatives)}}$$

Recall is defined as the proportion of correctly identified positives (i.e. negative reviews) over all positives (i.e. all negative reviews). It is a good complementary metric to accuracy because of the model's specific objective of identifying negative reviews. It is given by the following formula:

$$\frac{\text{true positives}}{\text{(true positives + false negatives)}}$$

In addition, the time taken to train and test the model will be assessed across the different algorithms. Given the speed of new information flow in social media, it is essential for the

(https://www.researchgate.net/publication/283318703_Detection_and_Scoring_of_Internet_Slangs_for_Sentiment_Analysis_Using_SentiWordNet)
'Context-Aware Spelling Corrector for Sentiment Analysis', by Fazal Masud Kundi, Aurangzeb Khan, Muhammad Zubair Asghar, Shakeel Ahmah (https://www.researchgate.net/publication/284344945_Context-Aware_Spelling_Corrector_for_Sentiment_Analysis)
'Lexicon-Based Sentiment Analysis in the Social Web', by Fazal Masud Kundi, Aurangzeb Khan, Muhammad Zubair Asghar, Shakeel Ahmah (https://www.researchgate.net/publication/283318830_Lexicon-Based_Sentiment_Analysis_in_the_Social_Web)

[4] Obtained from Kaggle at https://www.kaggle.com/snap/amazon-fine-food-reviews

[5] To avoid confusion, it is best to be clear that the term 'positive' as used in defining accuracy and recall, refers to negative reviews in the Amazon dataset.

model to be able to train on new data and make predictions quickly – in addition to being performing well.

ANALYSIS: DATA EXPLORATION

There are 568,454 reviews in the dataset. The reviews are all unique.

Each review within the dataset consists of the following labels:

- Score: an integer ranging from 1 to 5[6], indicating the reviewer's sentiment towards the product;
- Text: text of the review;
- Summary: title of the review;
- Id: integer stating the ordinal position of the review within the dataset
- UserId: unique identifier of the reviewer;
- ProductId: unique identifier of the product reviewed;
- ProfileName: username of the reviewer;
- HelpfulnessNumerator: number of mentions indicating the review was helpful;
- HelpfulnessDenominator: total number of mentions regarding the helpfulness of the review;
- Time: timestamp of the review.

Example of a review from the dataset:

> Score: 5
> Text: I have bought several of the Vitality canned dog food products and have found them all to be of good quality. The product looks more like a stew than a processed meat and it smells better. My Labrador is finicky and she appreciates this product better than most.
> Summary: Good Quality Dog Food
> Id: 1
> UserId: A3SGXH7AUHU8GW
> ProductId: B001E4KFG0
> ProfileName: delmartian
> HelpfulnessNumerator: 1
> HelpfulnessDenominator: 1
> Time: 1303862400

74,258 unique products were reviewed. The mean number of reviews for each product is 7.66, while the median number of reviews for each product is 2.

256,059 unique users contributed to the reviews. The mean number of reviews contributed by each user is 2.22, while the median number of reviews contributed by each user is 1.

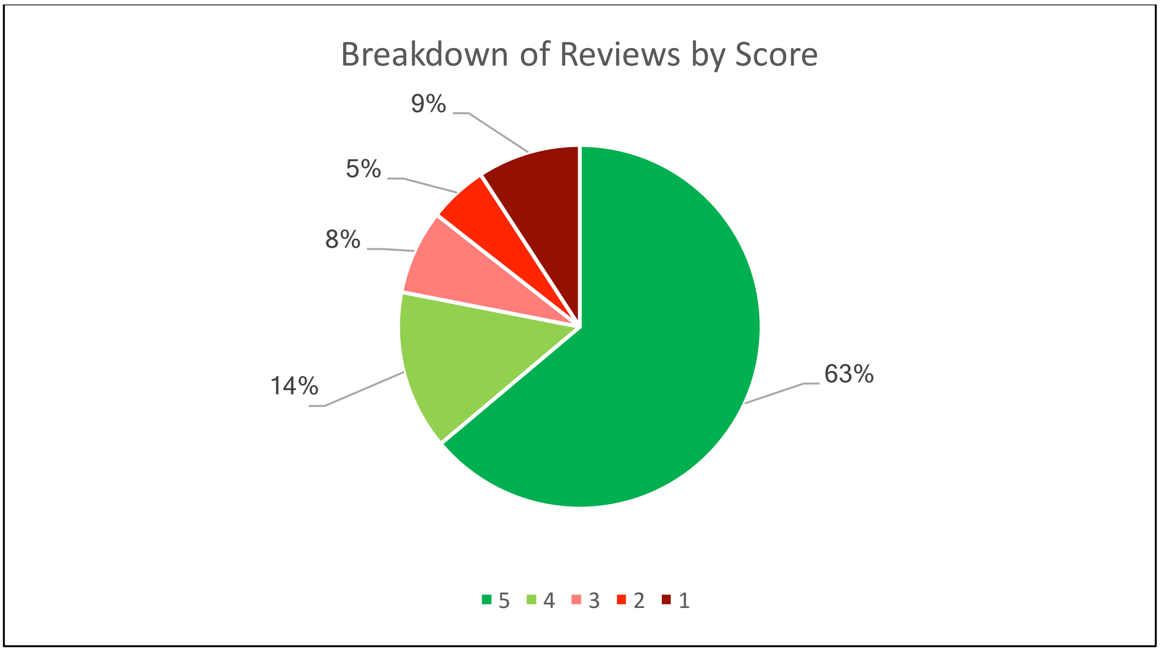
ANALYSIS: EXPLORATORY VISUALIZATION

There are 363,122 (63%) reviews with a Score of 5. There are 80,655 (14%) reviews with a Score of 4. There are 42,640 (8%) reviews with a Score of 3. There are 29,769 (5%) reviews

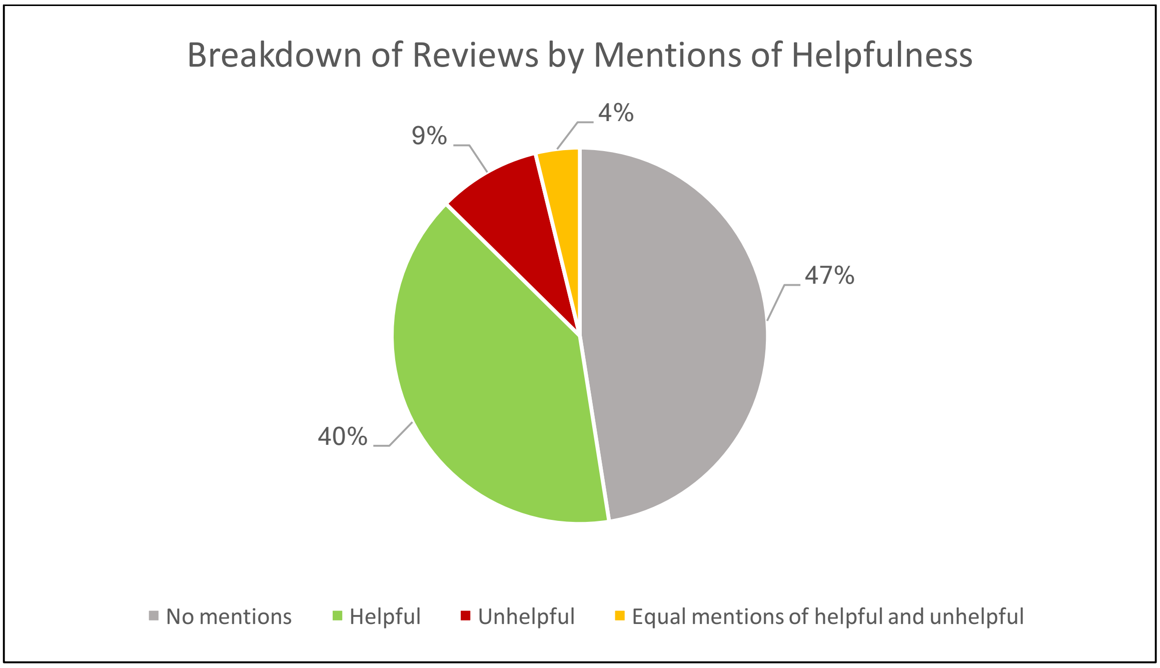
[6] A score of 5 being the best.

with a Score of 2. There are 52,268 (9%) reviews with a Score of 1. The mean Score is 4.18, while the median Score is 5.

Majority of reviews were awarded a Score of 5 and 4. Within the US cultural context, this is not surprising, as reviewers tend to be generous with ratings. A Score of 3 and below on Amazon is typically considered negative. 78% of all reviews received a score of 5 or 4, while 22% received a score of 3 and below and can be considered negative.



There are 298,402 (53% of all) reviews with at least one mention of helpfulness. There are 264,628 (46% of all) reviews that have at least one mention of being helpful. 226,666 (40%) of all reviews were indicated as being more helpful than unhelpful. 50,113 (9%) of all reviews were indicated as being more unhelpful than helpful. 21,623 (4% of all) reviews have equal mentions of helpfulness and unhelpfulness.



ANALYSIS: ALGORITHMS AND TECHNIQUES

Naïve Bayes Classifier: The Multinomial Naïve Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem with strong independence assumptions between features. It assumes features have a multinomial distribution in relation to the target variable. While simple, Naive Bayes classifiers can be extremely fast compared to more sophisticated methods and scale well for large features sets. They are known to work well in document classification problems.[7] This classifier was chosen because of the given dataset's large feature set, large sample size and text-based classification problem statement.

Ensemble techniques: Adaptive Boosting and Extreme Gradient Boosting are ensemble methods. Ensemble methods work by combining the predictions of several base estimators built with a given learning algorithm in order to help the model generalize well. Adaptive Boosting works by fitting a series of models that are only slightly better than random guessing. The predictions from all the models are then combined to produce a final prediction. [8] Extreme Gradient Boosting is premised on boosting tree algorithms. By employing multi-threads and imposing regularization, it is able to tap on more computational power and achieve greater accuracy. Extreme Gradient Boosting has been used in numerous winning Kaggle entries.[9] Both classifiers were chosen because of their ability to perform well on large non-linear feature/data sets.

Neural network: A neural network is made up of a collection of nodes arranged in layers. Each node typically processes information it receives before passing this on to another node. Learning takes place as information is passed through the layers, with the nodes adjusting how they treat the information after each series. Neural networks are especially useful for handling problems that are difficult to program with linear rules, and are effective when the dataset is large. The complexity of text-based sentiment analysis of the chosen problem statement and the large dataset suggest a neural network could be an appropriate model.

ANALYSIS: BENCHMARK

A social media manager combing through comments one at a time has a random chance of identifying a negative comment – the specific probability depending on the distribution of positive and negative sentiments. I propose using random chance as a benchmark in assessing the utility of the model.

In this dataset, 22% of the reviews are considered to be negative reviews. If I guessed every review was a positive review, I would score an accuracy of 78% and a recall of 0%. Conversely, if I guessed every review was a negative review, I would score an accuracy of 22% and a recall of 100%. The model should strive to achieve a good balance between accuracy and recall, while besting random chance and the strategy of guessing the same way every time.

[7] Reference https://en.wikipedia.org/wiki/Naive_Bayes_classifier and http://scikit-learn.org/stable/modules/naive_bayes.html

[8] Reference http://scikit-learn.org/stable/modules/ensemble.html#adaboost

[9] Reference http://xgboost.readthedocs.io/en/latest/model.html

METHODOLOGY: DATA PREPROCESSING

A new binary label, ‘Sentiment’, was created from “Score” by replacing values of ‘5’ and ‘4’ with ‘0’, while values, ‘1’, ‘2’ and ‘3’ were replaced with ‘1’. ‘Sentiment’ will serve as the target variable, where ‘1’ will indicate a review is negative. While it may be useful to ascertain the degree of negativity in a sentiment, I decided to transform the target label to a binary variable instead, in order to improve performance, and to benchmark against numerous existing sentiment analysis work which are also based on binary target variables.

```
#classifying scores into positive('0') and negative('1') sentiments
data['Sentiment'] = data.Score.map({5:0, 4:0, 3:1, 2:1, 1:1})
```

Next I split the dataset into separate pandas data frames – one as the training/testing set used to build and fine-tune the model, and the other as a holdout set used to test against the final model. This was done to prevent overfitting. The holdout set size was 0.3.

```
#extract hold-out set for testing final model
data1 = data.iloc[:397917]
data2 = data.iloc[397917:]
```

The samples from the training/testing data frame were split into training and testing sets, with a testing size of 0.3. This split was done with scikit-learn’s train_test_split function. Having a separate testing set helps to minimize overfitting.

```
#split into training and testing sets
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data1['Text'], data
1['Sentiment'], test_size=0.3, random_state=7)
```

The text in the ‘Text’ label was transformed through one-hot-encoding, which is a necessary step for feeding into a neural network. This was done through keras’ Tokenizer function and np_utils object. The number of unique words encoded was capped at 10000 to manage the memory requirements for implementing the neural network.

```
#create vectors for neural network
import keras
from keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer(nb_words=10000)

X_train_nn = tokenizer.sequences_to_matrix(X_train)
X_test_nn = tokenizer.sequences_to_matrix(X_test)

# One-hot encoding the output
from keras.utils import np_utils
num_classes = 2
y_train_nn = np_utils.to_categorical(y_train, num_classes)
y_test_nn = np_utils.to_categorical(y_test, num_classes)
```

The text in the ‘Text’ label was separately transformed to create a bag-of-words, with punctuations removed, in preparation for training the Naïve Bayes and ensemble classifiers. This was done with scikit-learn’s CountVectorizer function.

```
#build CountVectorizer object for NB, AdaBoost and XGBoost classifiers
```

```
from sklearn.feature_extraction.text import CountVectorizer
count_vector = CountVectorizer()

#transform and fit 'Text' training data and return frequency matrix
X_train = count_vector.fit_transform(X_train)

#transform 'Text' testing data and return frequency matrix
X_test = count_vector.transform(X_test)
```

METHODOLOGY: IMPLEMENTATION

Models for a Naïve Bayes, Adaptive Boosting and Extreme Gradient Boosting classifiers were build. Naïve Bayes and the ensemble models were built with scikit-learn, while Extreme Gradient Boosting was built with the xgboost library. Parameters for each model were left as defaults to test the models' performances out-of-the-box.

```
#build classifiers
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import AdaBoostClassifier
from xgboost import XGBClassifier

nb = MultinomialNB()
ada = AdaBoostClassifier()
xgb = XGBClassifier()
```

A looping function was created to train, predict and evaluate all three models. The times taken to train and predict were measured. Training and testing scores for accuracy, precision, recall and f1 were measured using scikit-learn's metrics object. Comparison of training and testing scores would help identify potential over- and under-fitting.

```
#create function to fit, predict and evaluate models
from sklearn.metrics import accuracy_score, precision_score, recall_sco
re, f1_score

def train_pred_eval(clf):
    #fit
    train_start = time()
    clf.fit(X_train, y_train)
    train_end = time()
    print('Trained {} model in {:.4f} seconds.'.format(clf, train_end –
train_start))
    #predict
    pred_start = time()
    pred_test = clf.predict(X_test)
    pred_end = time()
    pred_train = clf.predict(X_train)
    print('Made {} predictions in {:.4f} seconds.'.format(clf, pred_end
– pred_start))
    print('*******')
    #evaluate
    train_accuracy = accuracy_score(y_train, pred_train)
    train_precision = precision_score(y_train, pred_train)
    train_recall = recall_score(y_train, pred_train)
    train_f1 = f1_score(y_train, pred_train)
    test_accuracy = accuracy_score(y_test, pred_test)
    test_precision = precision_score(y_test, pred_test)
    test_recall = recall_score(y_test, pred_test)
    test_f1 = f1_score(y_test, pred_test)
```

The function was then run for each of the three models.

```
#fit, predict and evaluate models
for e in [ada, xgb, nb]:
    train_pred_eval(e)
```

A model architecture for the neural network was built with the keras library using a tensorflow backend. As a start, a simple model with fewer layers was chosen to quickly ascertain how the neural network would perform on the given dataset. A dropout layer was added to minimise overfitting, while a final softmax binary-class layer was chosen to reflect the model's attempt to predict the probability of a review being negative.

```
#build neural network model architecture
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation

model = Sequential()
model.add(Dense(32, activation='relu', input_dim=X_train_nn.shape[1]))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(2, activation='softmax'))
```

The model was compiled with a categorical_crossentropy loss function, which is suitable for the given classification problem, against the accuracy metric.

```
#compile neural network model
model.compile(loss='categorical_crossentropy', optimizer='adam', metric
s=['accuracy', 'precision', 'recall'])
```

A function was created to train and evaluate the neural network, with the times taken for training and making predictions measured.

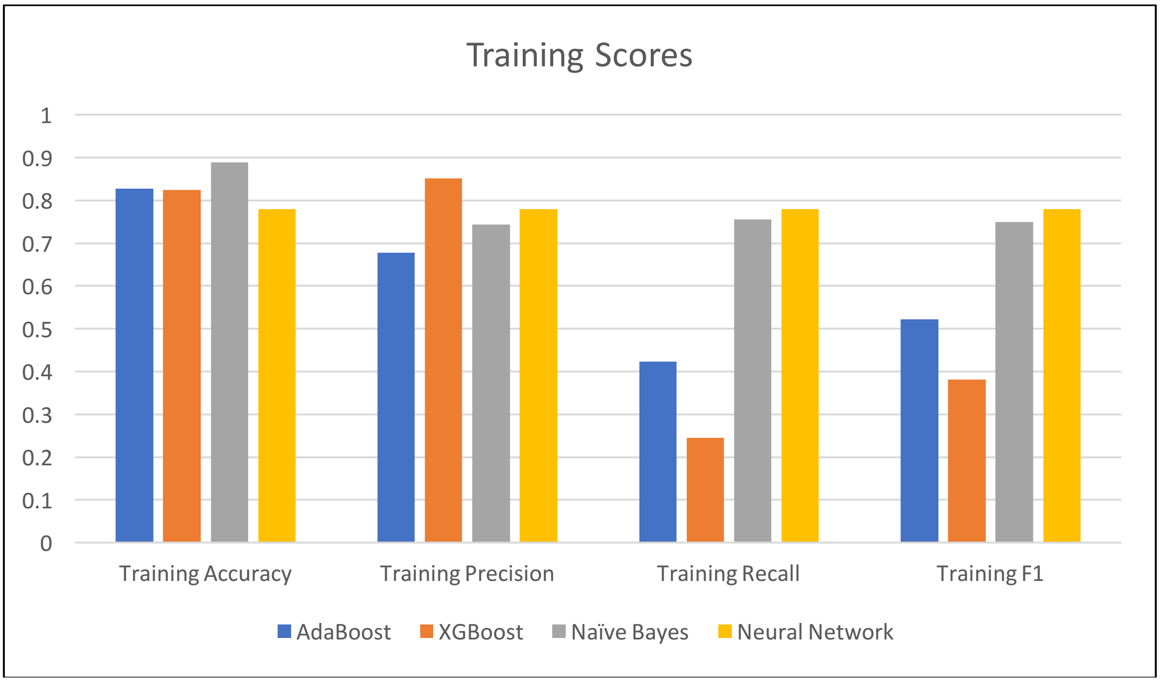
```
#create function to fit and evaluate neural network
def train_eval_nn(epoch,batch):
    #fit model
    train_start = time()
    model.fit(X_train_nn, y_train_nn, nb_epoch=epoch, batch_size=batch)
    train_end = time()
    print('Trained Neural Network in {:.4f} seconds.'.format(train_end
- train_start))
    #evaluate model
    pred_start = time()
    scores_nn = model.evaluate(X_test_nn, y_test_nn)
    pred_end = time()
    print('Made Neural Network predictions in {:.4f} seconds.'.format(p
red_end - pred_start))
```

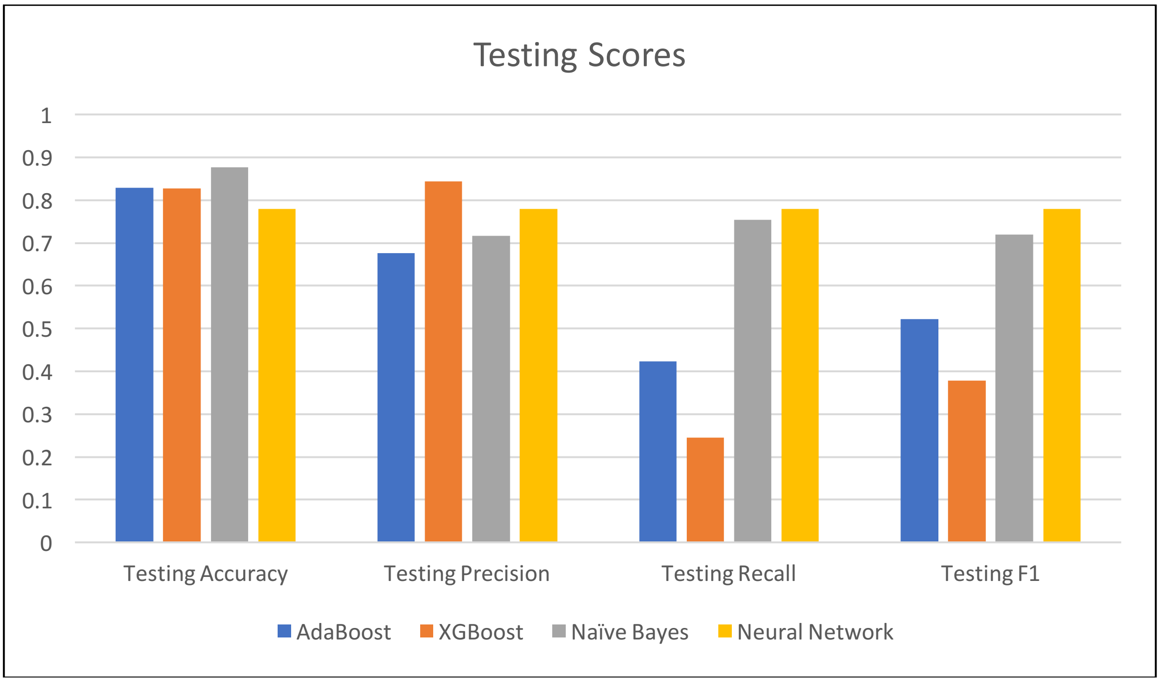
The function was run on the neural network model with an epoch of 3 and a batch size of 32. Both values were chosen to allow for quick computation.

```
#train and evaluate neural network
train_eval_nn(3,32)
```

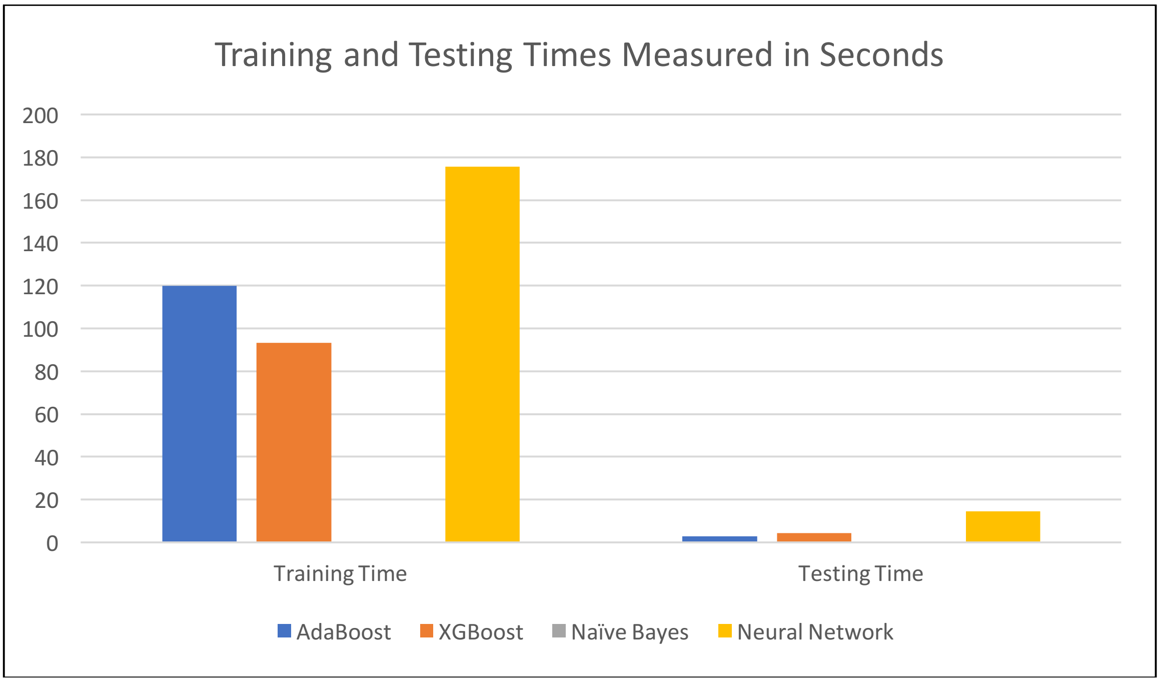
The performance of the models are captured in the column chart below. All four models scored well for testing accuracy. The poorest performer for recall was the Extreme Gradient Boosting classifier. Adaptive Boosting performed poorly for recall as well. Both the Naïve

Bayes classifier and the neural network performed the best, registering the best testing scores for accuracy and recall respectively. Both also exhibited little sign of overfitting when comparing training and testing scores.





The Naïve Bayes classifier took significantly less time to train and and make predictions than the other models. The neural network was the most computationally expensive model despite already being designed to be light-weight.



METHODOLOGY: REFINEMENT

Given the strong performance of the Naïve Bayes classifier and its lighter compute requirements, the Naïve Bayes model was chosen for subsequent refinement.

The first refinement was to increase the features, i.e. words, used to train the model. This was done by concatenating 'Summary' (the title of the review) with 'Text'. Increasing the size of the dataset used to train the model typically improves performance.

```
#create new column with combined text from 'Summary' and 'Text'
data1['Summary'] = data1['Summary'].astype(str)
data1['text_all'] = data1['Summary'] + " " + data1['Text']
```

The second refinement comprised of two parts. The first was to emphasize the importance of "Summary" over "Text". I did this by replicating "Summary" in the dataset used to train the model until the ratio of "Summary" to "Text" was 6:1. This was based on the hypothesis that the words in "Summary" had more predictive power than the words in "Text". It stands to reason that the words in the title of a review would be more meaningful than the actual review which is likely more verbose. I considered training the model initially with "Summary" instead of "Text", but found the performance to be poorer, likely because of the smaller vocabulary in "Summary". Through this refinement, I was able to incorporate the vocabulary of "Text", while leveraging the richness of the words in "Summary".

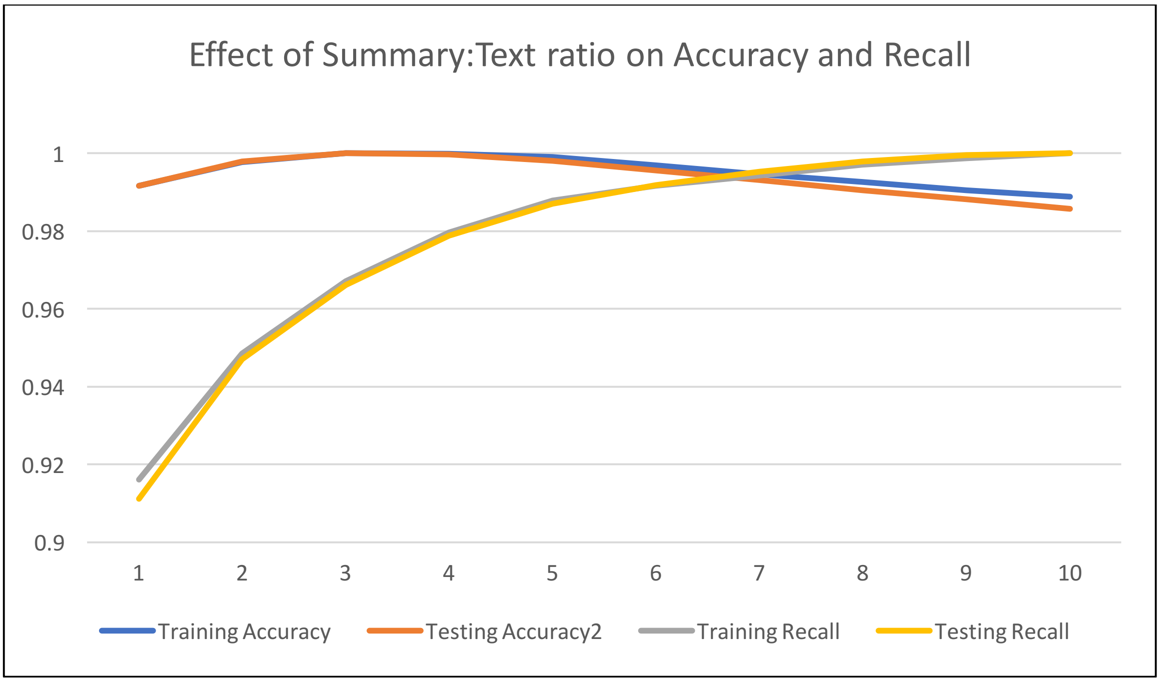
```
#accentuate words in 'Summary' by creating a new column with combined text from 'Summary' and 'Text' with the ratio of 6:1
data1['text_all'] = data1['Summary'] + " " + data1['Summary'] + " " + data1['Summary'] + " " + data1['Summary'] + " " + data1['Summary'] + " " + data1['Summary'] + " " + data1['Text']
```

The second refinement also consisted of a step to increase the emphasis of the most frequently occurring words in "Summary" and "Text". I did this by taking advantage of the characteristic of the bag-of-words technique, which counts the number of times a word appears in a review - and replicated the collection of words found in the combined

“Summary” and “Text” label. While this emphasis could also have been achieved through limiting the number of words used in training the Naïve Bayes classifier to the most frequently occurring words, this would also have had the effect of limiting the vocabulary, which would have a negative impact on the model’s performance.

```
#accentuate most frequently occuring words by replicating "text_all"
data1['text_all'] = data1['text_all'] + " " + data1['text_all'] + " " +
data1['text_all'] + " " + data1['text_all'] + " " + data1['text_all'] +
" " + data1['text_all'] + " " + data1['text_all']
```

The ratio of 6:1 and of replicating the collection of words 7 times were arrived at through a series of measured trials. I observed that increasing the emphasis of “Summary” over “Text” had the effect of improving accuracy only until a ratio of 3:1. Thereafter accuracy suffered. The table below captures this relationship. The x-axis represents the number of instances of “Summary”, while the y-axis is normalised to a scale of 0 to 1. Recall continued to improve with the ratio, but at a diminishing rate. Replicating the entire collection of words had the effect of reducing accuracy and improving recall.



The third refinement was to filter the dataset to include only reviews with at least one mention of it being helpful. This had the effect of improving both testing accuracy recall scores.

```
#remove reviews without a single HelpfulnessNumerator
data_helpful = data1[data1['HelpfulnessNumerator'] > 0]
```

In addition to the above discussed refinements, the following iterations were also attempted:

- Extracting a holdout set with size 0.2 and 0.1: Both resulted in overfitting and poorer performance for the holdout set;
- Splitting the training and testing set with a test set size of 0.2 and 0.1: Both resulted in overfitting to the training set and poorer performance for the testing set;

- Iterating over the number of words to be tokenized for the neural network: Increasing the number of words beyond 10000 resulted in minimal performance but significantly increased compute time;
- Using ‘english’ for CountVectorizer’s stop_words: This yielded a negative impact on performance;
- Setting a cap on CountVectorizer’s max_features: This had a negative impact on performance and was not required given Naïve Baye’s quick compute times;
- Setting a feature_count limit for Naïve Bayes model: This reduced the performance of the model;
- Using different activations, loss functions, batch sizes, epochs and optimizers to build and compile the neural network: Minimal impact to performance;
- Iterating the second refinement with different ratios of “Summary” to “Text” and different number of replications for the aggregated collection of words;
- Iterating the third refinement with different filters including omitting reviews that were mentioned as more unhelpful than helpful and only including reviews that were mentioned as more helpful than unhelpful

RESULTS: MODEL EVALUATION AND VALIDATION

The final model chosen is a Multinomial Naïve Bayes classifier. It was chosen because of its strong performance in accuracy, precision, recall and f1 scores, as well as its lighter compute requirements. It should be noted that a significant improvement in its performance was achieved by manipulating the data to take advantage of the bag-of-words technique. These gains would not have been achievable for the neural network model, which had performed well initially. The best testing accuracy I achieved with a neural network after refinements was still shy of 0.85.

The table below summarise the metrics of the final model and its improvement over the refinements.

	Initial	**After Refinement 1**	**After Refinement 2**	**After Refinement 3**
Train Accuracy	0.8881	0.9015	0.9035	0.9154
Train Precision	0.7433	0.7641	0.7295	0.7889
Train Recall	0.7546	0.8022	0.8957	0.9049
Train F1	0. 7489	0.7827	0.8042	0.8430
Test Accuracy	0.8761	0.8916	0.8909	0.8944
Test Precision	0.7164	0.7425	0.7034	0.7502
Test Recall	0.7546	0.8022	0.8959	0.9049
Test F1	0.7197	0.7591	0.7783	0.8031
Train Time (s)	0.1943	0.1900	0.1933	0.0989
Test Time (s)	0.0569	0.0592	0.0611	0.0311

The final model was tested against the holdout set to validate its robustness and whether it generalized well to unseen data. While the metrics dipped across the board, the scores were still reasonably strong and indicated the model’s value. Holdout set scores are captured in the following table.

	After Refinement 3	Holdout Set
Test Accuracy	0.8944	0.8567
Test Precision	0.7502	0.6384
Test Recall	0.9049	0.7750
Test F1	0.8031	0.7001
Test Time (s)	0.0311	0.0864

RESULTS: JUSTIFICATION

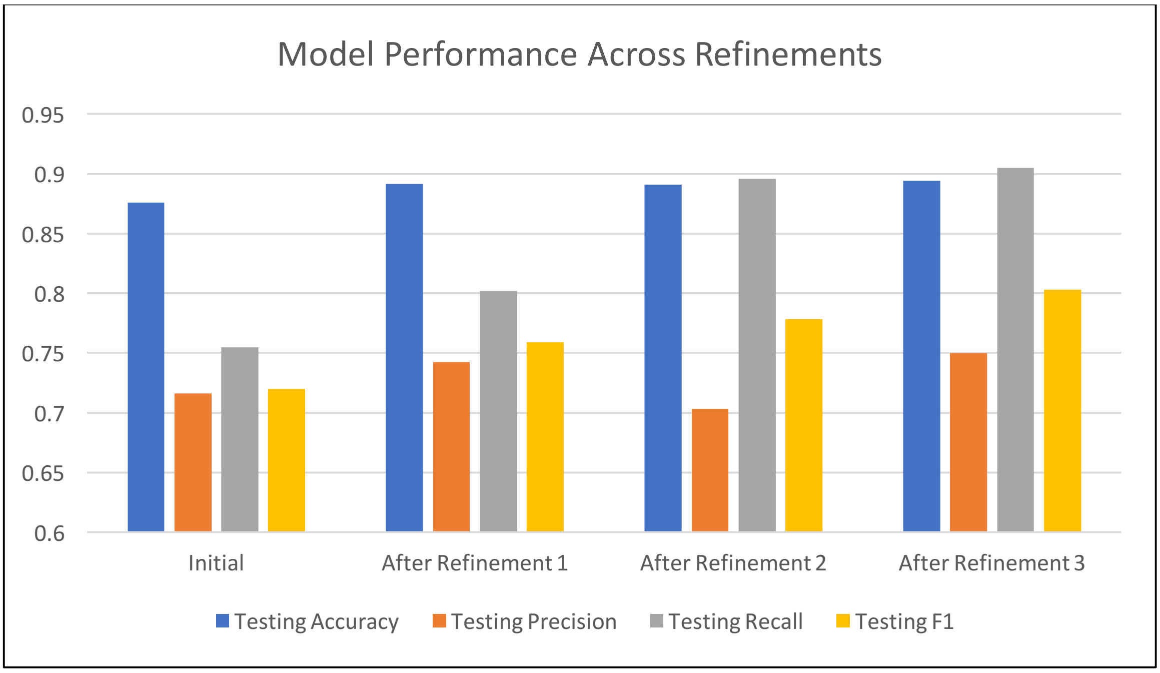
Based on the performance of the final model on the holdout set, I believe the model trained can address the stated problem statement, i.e. helping social media managers more effectively and quickly identify negative sentiments. The table below compares the model's metrics against the benchmark – random chance. The model significantly outperforms the benchmark across all metrics.

	Guessing that every review is negative	Guessing that every review is positive	Final model on holdout set
Accuracy	0.2200	0.78	0.8567
Precision	0.2200	0	0.6384
Recall	1.0000	0	0.7750
F1	0.3600	0	0.7001

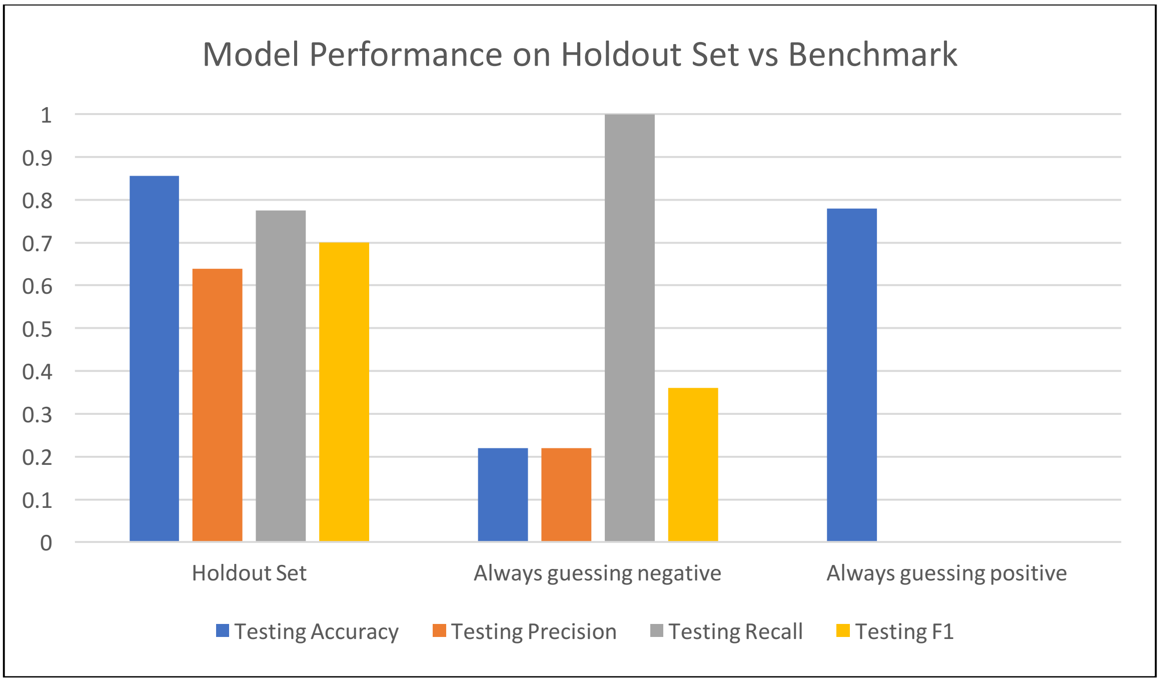
In addition, based on existing literature and work on sentiment analysis, an overall accuracy of 85% is considered good.

CONCLUSION: FREE-FORM VISUALIZATION

The column chart below captures the model's improvements across the refinements. It reflects the potential of leveraging machine learning to extract information and insights from data.



The column chart below captures the model's performance on the holdout set against the benchmarks. It indicates a significant outperformance and reflects the value of machine learning in assisting on everyday tasks.



Note: Interestingly, the neural network, without any parameter tuning, had an accuracy score of 0.7842 with the holdoutset, which is higher than the final model's recall score of 0.7750. This suggests there could be potential to build a more accurate model leveraging on a neural network architecture.

CONCLUSION: REFLECTION

When I started this project, I spent a considerable amount of time considering which problem statement to focus on. There are so many potential application areas for machine learning. An important consideration is being able to locate a dataset that canbe used to trained an appropriate model. On more than one occasion, I was excited by a potential application but could not locate an appropriate dataset, or conversely, found a rich dataset but could not match it with an application I was keen on.

Once I had determined a problem statement and located the relevant dataset, I set about researching different ways others have approached the same problem, paying particular attention to existing benchmarks for a good model. While sentiment analysis is a field that has seen much interest, there are still so many ways to build and fine-tune a model that have not been investigated in-depth.

The process of data exploration, prepossessing, implementation and refinements across different techniques was scientific. By that I mean there was a lot of trial and error, with each process meticulously measured and recorded. For me, machine learning, while a deeply logical field, feels like a concentration where practice has outpaced theory, and many decisions are based on intuition instead of an exact science. I absolutely love the confluence of creativity and logic arising from this irony. It was also a heart-pounding moment waiting for the results from testing against the holdout set.

The following are some of my other takeaways from this exercise:

- It was much easier to achieve good performance out-of-the-box with the simpler Naïve Bayes classifier as opposed to more sophisticated techniques;
- Significant improvements in performance can be achieved from data manipulation, potentially more so than parameter tuning (might be more true for simpler techniques);
- Difficult to determine an appropriate model architecture for a neural network;
- The neural network's loss plateaued after 2 epochs – I was under the impression that numerous epochs were usually required to train a neural network;
- I did not realise a neural network would produce identical scores for accuracy, precision and recall;
- Overfitting happens so easily and surreptitiously;
- Significant effort was required to set up the machine learning environment, especially when working across an AWS Ubuntu instance and a Macbook; and
- It is time to migrate to Python 3 with many popular and newer libraries already making the switch.

CONCLUSION: IMPROVEMENT

Some potential improvements to explore include:

- Using a recursive neural network model architecture;
- Incorporating image recognition training with a convolutional neural network, to better predict the sentiment of an online post;
- More parameter tuning for ensemble methods and neural network;
- Incorporate additional datasets to train the model, e.g. pair with IMDB movie dataset;
- Using a recursive neural network to draft responses for social media managers when a negative sentiment is identified; and
- Instead of setting a binary target variable, i.e. positive or negative review, consider incorporating degrees of negativity in the target variable.