Analysing Product Reviews

- Ganga Meghanath

***Abstract*—In this work, we develop sentiment analysis models to capture positive and negative sentiments associated with product reviews. We work with a subset of the Amazon Product Reviews dataset. We compare classification models built using count-based word vectors to ones trained on top of pre-trained sentence embeddings. We also evaluate the performance of pre-trained transformer based sentiment analysis models on the given dataset. Finally, we try to derive results on attributing credits to product features for the success or failure of the product using the reviews of customers on the product.**

I. Introduction:

A. Sentiment Analysis

Sentiment analysis studies the subjective information in an expression, that is, the opinions, appraisals, emotions, or attitudes towards a topic, person or entity. Expressions can be classified as positive, negative, or neutral[1]. Sentiment analysis is the contextual mining of text which identifies and extracts subjective information in source material, and helping a business to understand the social sentiment of their brand, product or service while monitoring online conversations[2].

In this project, we build models to classify sentences in the product reviews as positive or negative. This will help us go beyond product ratings and understand customers views about the product. For example, a user rating of 3/5 could be a result of a mix of positive and negative customer experience with the product. In the case of a tablet, the customer might like the screen feature for reading but at the same time dislike the user interface on the home screen.

B. Credit Assignment

Credit Assignment Problem concerns itself with determining how the success of a system's overall performance is due to the various contributions of the system's components.[3].

Instead of focusing on exhaustively solving this problem, we'll be focusing more on how this problem can be approached using sentiment analysis. The text content of product reviews may contain information about what the customer liked and disliked about the product. For this reason, instead of classifying the whole review, we'll focus on classifying sentences in the reviews as positive or negative. In order to retrieve feature related reviews, we'll mine the relevant sentences using vector representations of sentences that capture semantic information. Later, we'll define metrics to capture feature relevance to the overall product success.

II. Dataset

A. Consumer Reviews of Amazon Products

This is a list of over 34,000 consumer reviews for Amazon products like the Kindle, Fire TV Stick, and more provided by Datafiniti's Product Database[4]. The dataset includes basic product information, rating, review text, and more for each product[5](https://www.kaggle.com/datafiniti/consumer-reviews-of-amazon-products).

We are interested in the fields id, name, reviews.rating, reviews.title and reviews.text for our analysis. Product rating (1-5) indicates the consumers satisfaction with the overall product which we will threshold and use as labels for training our sentiment classifiers along with the review content as the text input.

```
Data columns (total 21 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   id                     34660   object
 1   name                   27900   object
 2   asins                  34658   object
 3   brand                  34660   object
 4   categories             34660   object
 5   keys                   34660   object
 6   manufacturer           34660   object
 7   reviews.date           34621   object
 8   reviews.dateAdded      24039   object
 9   reviews.dateSeen       34660   object
 10  reviews.didPurchase    1       object
 11  reviews.doRecommend    34066   object
 12  reviews.id             1       float64
 13  reviews.numHelpful     34131   float64
 14  reviews.rating         34627   float64
 15  reviews.sourceURLs     34660   object
 16  reviews.text           34659   object
 17  reviews.title          34655   object
 18  reviews.userCity       0       float64
 19  reviews.userProvince   0       float64
 20  reviews.username       34658   object
```

An example review :

- id : AVphgVaX1cnluZ0-DR74
- name : Fire Tablet, 7 Display, Wi-Fi, 8 GB - Includes Special Offers, Magenta
- reviews.rating : 4.0
- reviews.title : Good product.
- reviews.text : This tablet is a well made tablet. Be sure to get a sd card as the memory is not great. The Amazon underground with totally free games and apps is worth the purchase alone. You do not need Amazon Prime subscription service to enjoy this little tablet. Go on and purchase one for 50.00 you wont regret the purchase.

III. Data Pre-processing

For generating training data for the sentiment classification of reviews, we concatenate "reviews.title" and "reviews.text" into a new column "reviews.txt". We then create a hash map as a unique review identifier for the dataset. The new column "reviews.txt" is then split into individual sentences by splitting on sentence separator '.'. The rows with empty "reviews.txt" is filtered out.

Process	Data Count
Initial data	34660
Remove empty reviews.text and reviews.title	34654
Split each review into multiple sentences	156915
Remove empty reviews.txt	124985

For classifying reviews as positive and negative, we need to transform the sentences in "reviews.txt" into vectors.

A. Word Count vectors

We first remove punctuations (!"#$%&'()*+,-./ :;<=>?@[ ]^_`{|}) from the input string. We won't be using nltk library to remove stopwords from the reviews since words such as "wouldn't", "shouldn't", etc are present among stopwords and these indicate sentiments and hence we want to keep them in our corpus.

For converting to vector, we use "CountVectorizer()" function from "sklearn.feature_extraction.text". It convert a collection of text documents into a matrix of token counts[6] (scipy.sparse.csr_matrix). Here, we use the entire corpus (all review sentences) to get the word token frequency matrix. We don't explicitly convert it into train and test set and account for missing words since we won't be training this input vector layer to change this representation and will use the same feature space for training and testing our models.

B. Pretrained sentence embedding

Here, we directly make use a pre-trained sentence transformer model to encode the sentences into vector representations. The motivation to use a pre-trained embedding stems from the notion that it captures semantic information that can help us build better sentiment classifiers.

We use "all-mpnet-base-v2" model from HuggingFace for embedding our review sentences. It is a sentence-transformer model that maps sentences & paragraphs to a 768 dimensional dense vector space and can be used for tasks like clustering or semantic search. (They used the pretrained microsoft/mpnet-base model and fine-tuned it on a 1 Billion sentence pairs dataset using a contrastive learning objective[7]).

We run inference on the sentence-transformer using our dataset of review sentences to get their corresponding sentence embeddings which will be used as input to our classifiers.

IV. Sentiment Analysis Models

We'll be describing the classification models used in the project in the following subsections. Details regarding model performances will be discussed in later sections.

A. Naive Bayes Classifier

We use the Naive Bayes classifier for multinomial models (MultinomialNB) from "sklearn.naive_bayes" library. The multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). Hence, we directly train the model on Word Count vectors. It's said that in practice, fractional counts such as tf-idf may also work with MultinomialNB. Hence we also try training it on Pretrained sentence embedding after offsetting it to create positive vector values.

B. Support Vector Machine : rbf kernel

We use the C-Support Vector Classification function from "sklearn.svm" library with rbf kernel to train our SVM for classifying sentiments. Two different models are trained using Word Count vectors and pretrained sentence embedding respectively as the sentence vector inputs to the model. For Word Count vector input, we standardise and scale the vectors using StandardScaler() function from "sklearn.preprocessing" library before feeding it into the classifier.

C. K-Nearest Neighbor Classifier

We use the "KNeighborsClassifier" function from "sklearn.neighbors" for training a K-nearest neighbour classifier on the two sentence representations : Word Count vectors and Pretrained sentence embedding. Since this is based on distance metric, we can feed in data directly to the models without scaling. We use n_neighbors$= 5$ in our classifiers.

D. Random Forest

We use "RandomForestClassifier" from "sklearn.ensemble" to build our classification models on the two different sentence representations. Since this is based on decision tree as our base estimator and splitting based on features at nodes, scaling the features shouldn't affect our classifier performance. We use 100 decision trees (base estimators) in our ensemble with splitting criterion set as gini.

E. AdaBoost Classifier

We use "AdaBoostClassifier" from "sklearn.ensemble" with decision tree as base classifier and number of estimators as 100 to build our classification model on the two different input sentence feature representations. The max depth of the base decision tree is set to 1 and uses the real boosting algorithm SAMME.R to train the classifiers.

F. Neural Network Classifier

We use the Multi-layer Perceptron classifier from the "sklearn.neural_network" library to build our classification models on the two different sentence feature embeddings. We use a 3 hidden layer neural network (256, 128, 64) with relu activation and train it for 200 epochs using adam optimizer with early stopping enabled and validation_fraction$= 0.1$.

G. Distilled Bert

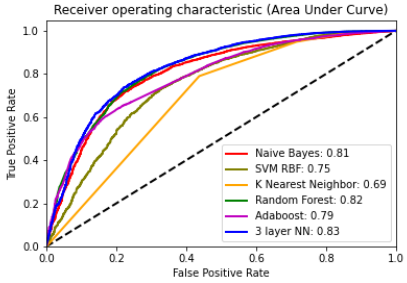
We use sentiment-analysis pipeline from transformers library. We load the "distilbert-base-uncased-finetuned-sst-2-english" model and tokenizer from HuggingFace[8] and use it to infer the sentiment class and score on our entire reviews dataset.

H. MPNet

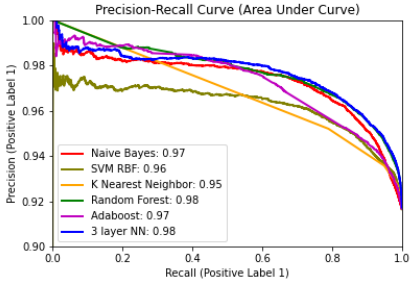
We use sentiment-analysis pipeline from transformers library. We load the pretrained weights from "microsoft/mpnet-base" to MPNetTokenizer and MPNetForSequenceClassification from HuggingFace transformers[9] and use it to infer the sentiment class and score on our entire reviews dataset.

V. Performance Analysis

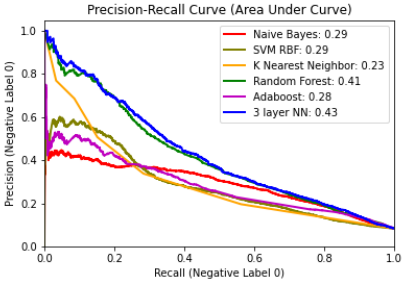
This section comprises of performance analysis of the sentiment classification models based on conclusions drawn from the experimental results, mainly from the Figures 1, 2 and 3.



(a) ROC plot for the different classifiers. The area under the ROC curve for each classifier is given against it in the legend.

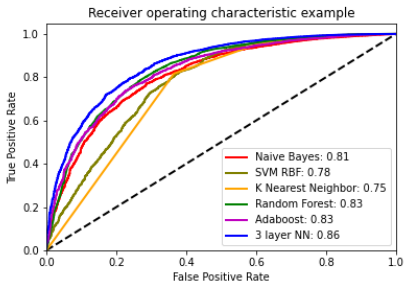


(b) PR curve for class 1 for the different classifiers. The area under curve for each classifier is given against it.

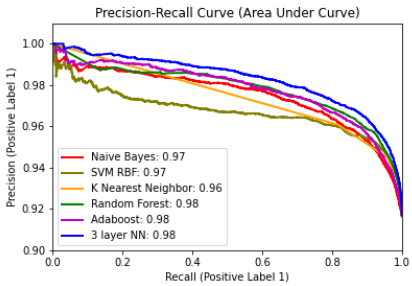


(c) PR curve for class 0 for the different classifiers. The area under curve for each classifier is given against it.

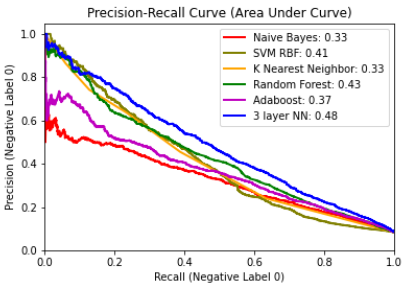
Fig. 1: Input sentence embedding : Word Count vectors : Performance analysis of different models on sentiment classification task using word-token-count based sentence embedding. Sentiment of a review is classified into either positive or negative. The train-test stratified split ratio is $80:20$. The performance is evaluated on the 20% test set. The true class labels for training and performance evaluation are obtained from "reviews.rating". We threshold "reviews.rating" column at value 3. If "reviews.rating" > 3 then the true class = positive = 1. If "reviews.rating" $<= 3$ then the true class = negative = 0.[**Colab Notebook link**]



(a) ROC plot for the different classifiers. The area under the ROC curve for each classifier is given against it in the legend.



(b) PR curve for class 1 for the different classifiers. The area under curve for each classifier is given against it.



(c) PR curve for class 0 for the different classifiers. The area under curve for each classifier is given against it.

Fig. 2: Input sentence embedding : Pretrained sentence embedding : Performance analysis of different models on sentiment classification task using sentence-transformer based review embedding. Sentiment of a review is classified into either positive or negative. The train-test stratified split ratio is $80:20$. The performance is evaluated on the 20% test set. The true class labels for training and performance evaluation are obtained from "reviews.rating". We threshold "reviews.rating" column at value 3. If "reviews.rating" > 3 then the true class = positive = 1. If "reviews.rating" $<= 3$ then the true class = negative = 0.[**Notebook link**]

A. Input sentence embedding : Word Count vectors

From Fig. 1, we can see that, when the models are trained using Word Count vectors as the input sentence embeddings, the best performing model (on the test set) (based on the area under the ROC and PR curves) is the 3-layered Neural Network, followed by Random Forest, followed by Naive Bayes.

B. Input sentence embedding : Pretrained sentence embedding

From Fig. 2, we can see that, when the models are trained using the pretrained sentence embeddings (obtained by running inference on the sentence-transformer) as the input sentence embeddings, the best performing model (on the test set) (based on the area under the ROC and PR curves) is the 3-layered Neural Network, followed by Random Forest, followed by Adaboost.

C. Pre-trained Sentiment Analysis Classifier

From Fig. 3, on comparing the two pre-trained sentiment analysis transformer models performance on the given dataset, we see that (from the area under ROC and PR curves), Distilled bert model performs slightly better than the MPNet model on our dataseet at both thresholds (for true class) 0.3 and 0.4.

D. Overall performance analysis

Comparing Fig 1, 2 and 3, we can see that on the given dataset, the model with the best performance (from the area under ROC and PR curves) is the 3-layered Neural Network trained on the pre-trained sentence-transformer embeddings.

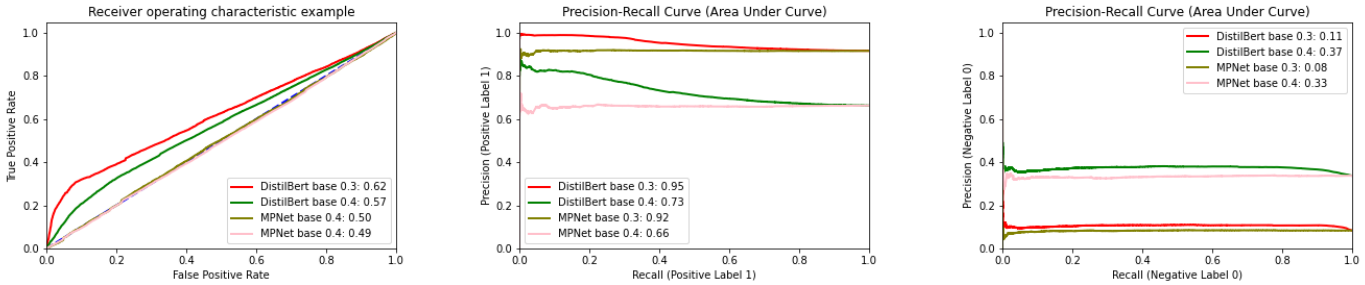
But we must keep in mind that these models were trained using hard labels extracted from review ratings and does not capture the notion of positive and negative sentiments well since our labelled data for each review sentence is biased towards the overall product rating given by that reviewer. Hence, going forward, when we need to utilize a sentiment classifier, we'll be using the pre-trained sentiment classifier transformer model since it was trained on a much bigger dataset and finetuned for the general sentiment analysis task.

VI. Credit Assignment

We can use data mining and sentiment analysis to assign credits to different features for the success of a product. We can also try to find an approximate estimate of the overall average rating for each feature (we're interested in) for a product with different features. To demonstrate the idea, lets look at an example. Consider the following product [**Google Colab Notebook link**]:

- id = "AVphgVaX1cnluZ0-DR74"
- name = "Fire Tablet, 7 Display, Wi-Fi, 8 GB - Includes Special Offers, Magenta"

Data pre-processing steps for the reviews corresponding to the above product:



(a) ROC plot comparison between the two pre-trained sentiment analysis classifiers. The area under the ROC curve for each classifier is given against it in the legend.

(b) PR curve comparison for class 1 b/w the two pre-trained sentiment analysis classifiers. The area under curve for each classifier is given against it.

(c) PR curve comparison for class 0 b/w the two pre-trained sentiment analysis classifiers. The area under curve for each classifier is given against it.

Fig. 3: Pre-trained Sentiment Analysis Classifier : Pretrained sentence embedding : Performance analysis of two different pre-trained sentiment analysis classification models on the product review dataset. The models used are Distilled Bert and MPNet. The models were not finetuned on the dataset and are only used for inference. The true class labels for performance evaluation are obtained from "reviews.rating". We threshold "reviews.rating" column at values 3 and 4 (Example : "reviews.rating" > 3, true class = positive=1 and "reviews.rating" $<= 3$, true class = negative=0.[**Google Colab Notebook link**]

Process	Data Count
Initial data	10751
Remove empty reviews.text and reviews.title	10751
Split each review into multiple sentences	46853
Remove empty reviews.txt	37325

For the given example, let's use the pre-trained sentence embedding from section III B to get vector representations of the review sentences.

Now let's decide on a couple of features we might be interested in for a tablet. Consider the following sets of features and query terms :

Feature	Query Terms
OS	["OS", "operating system"]
audio	["listen to music", "audio"]
camera	["camera"]
games	["games"]
kindle	["kindle"]
price	["price", "value"]
read	["read books", "read"]
speed	["speed", "fast", "slow", "lag"]
touch	["touch"]
watch	["watch videos", "play movies"]

Get the sentence embedding for the query terms and mine data from the reviews corpus for each feature using

$$\text{average(cosine\_similarity(query, review))} > 0.5$$

.

For each feature, this gives us reviews that are similar to it. For the mined subset of reviews, get the sentiment score. For now, let's use the pretrained sentiment-analysis model mentioned in Section IV H (MPNet) to get the sentiment score for these reviews. Before deciding the sentiment class as positive or negative, let's adjust the confidence score based on the product rating. Calculate the new sentiment score as,

$$\text{NewScore} = 0.75 \times \text{MPNet Score} + 0.25 \times \text{review.rating}/5$$

Now define a threshold for positive and negative reviews,

$$\text{Positive Review} \implies \text{NewScore} > 0.5$$
$$\text{Negative Review} \implies \text{NewScore} \leq 0.5$$

This gives us the following counts per feature :

Feature	+ve review	-ve review
OS	7	0
audio	3	0
camera	76	6
games	49	0
kindle	4352	69
price	243	3
read	51	1
speed	29	9
touch	9	0
watch	11	1
excess	31136	1277

Now, we can consider positive review contribution weight of each feature i as,

1.) Within feature weights :

$$w_i^+{}_{small} = \frac{n_{\text{+ve review i}}}{n_{\text{review i}}}$$
$$w_i^-{}_{small} = \frac{n_{\text{-ve review i}}}{n_{\text{review i}}}$$

2.) Across feature weights :

$$w_{+sum} = \sum_{j=1}^{m} n_{\text{+ve review j}}$$
$$w_{-sum} = \sum_{j=1}^{m} n_{\text{-ve review j}}$$
$$n_{i-pos} = \frac{n_{\text{+ve review i}}}{w_{+sum}}$$
$$n_{i-neg} = \frac{n_{\text{-ve review i}}}{w_{-sum}}$$

$$w_i^+{}_{big} = \frac{n_{i-pos}}{n_{i-pos} + n_{i-neg}}$$
$$w_i^-{}_{big} = \frac{n_{i-neg}}{n_{i-pos} + n_{i-neg}}$$

3.) Combining both the information gives us,

$$w_i^+ = \frac{w_i^+{}_{big} \times w_i^+{}_{small}}{\sum_{j=1}^{m} w_j^+{}_{big} \times w_j^+{}_{small}}$$
$$w_i^- = \frac{w_i^-{}_{big} \times w_i^-{}_{small}}{\sum_{j=1}^{m} w_j^-{}_{big} \times w_j^-{}_{small}}$$

$$\sum_{j=1}^{m} w_j^+ = 1$$
$$\sum_{j=1}^{m} w_j^- = 1$$

where m includes the features we considered and the additional "excess" feature accounts for the reviews we didn't consider in our feature analysis.

$$n_{\text{review excess}} = count(set\{\text{all reviews}\} - \cup_{j=1}^{m} set\{\text{reviews for j}\})$$

To compute approximate overall rating for feature i based on the rating for the product, we first compute the average product rating using the reviews that cover information about the feature :

$$\text{avg.reviews.rating}_i = \frac{\sum_{j=1}^{n_{\text{review i}}} \text{reviews.rating}_j}{n_{\text{review i}}}$$

Hence, overall rating for feature i is given by,

$$\text{rating}_i = (1 + w_i^+ - w_i^-) \times \text{avg.reviews.rating}_i$$
$$\text{rating-corrected}_i = max(1, min(5, \text{rating}_i))$$

For the above example features we considered,

Feature	w+	w−
OS	0.1379	0.0
audio	0.1379	0.0
camera	0.0477	0.1122
games	0.1379	0.0
kindle	0.0961	0.0130
price	0.1027	0.0087
read	0.0891	0.0191
speed	0.0114	0.6155
touch	0.1379	0.0
watch	0.0372	0.1715
excess	0.0636	0.0597

Feature	$\text{avg.reviews.rating}_i$	$\text{rating-corrected}_i$
games	4.734	5
kindle	4.563	4.942
price	4.495	4.918
speed	3.315	1.312

VII. Conclusion

In this project, we trained and evaluated different models for identifying sentiments in reviews and classifying sentences in reviews as positive or negative. We also evaluated the quality of the initial sentence embeddings through model performance evaluations. We identified that pre-trained sentence transformer embedding that captures context and semantic information from text gave better classification results as compared to models trained on word-token-count based sentence embedding. We further looked at credit assignment to features of the product using consumer reviews. We talked about how sentiment analysis classifiers can aid in allocating feature weights to positive and negative feature related reviews mined from the corpus for a specific product. We finally concluded with equations to compute feature ratings from product ratings.

Appendix

The classification report for each of the trained models are given below :

1) Naive Bayes
   - Word-Count-vectors

	precision	recall	f1-score	support
0	0.37	0.24	0.29	2087
1	0.93	0.96	0.95	22910
accuracy			0.90	24997
macro avg	0.65	0.60	0.62	24997
weighted avg	0.89	0.90	0.89	24997

   - Pretrained-sentence-embedding

	precision	recall	f1-score	support
0	0.00	0.00	0.00	2087
1	0.92	1.00	0.96	22910
accuracy			0.92	24997
macro avg	0.46	0.50	0.48	24997
weighted avg	0.84	0.92	0.88	24997

2) SVM RBF kernel
   - Word-Count-vectors

	precision	recall	f1-score	support
0	0.00	0.00	0.00	2087
1	0.93	1.00	0.96	22910
accuracy			0.92	24997
macro avg	0.46	0.50	0.48	24997
weighted avg	0.84	0.92	0.88	24997

   - Pretrained-sentence-embedding

	precision	recall	f1-score	support
0	0.00	0.00	0.00	2087
1	0.92	1.00	0.96	22910
accuracy			0.92	24997
macro avg	0.46	0.50	0.48	24997
weighted avg	0.84	0.92	0.88	24997

3) K-Nearest Neighbors
   - Word-Count-vectors

	precision	recall	f1-score	support
0	0.51	0.15	0.23	2087
1	0.93	0.99	0.96	22910
accuracy			0.92	24997
macro avg	0.72	0.57	0.59	24997
weighted avg	0.89	0.92	0.90	24997

   - Pretrained-sentence-embedding

	precision	recall	f1-score	support
0	0.64	0.24	0.34	2087
1	0.93	0.99	0.96	22910
accuracy			0.92	24997
macro avg	0.79	0.61	0.65	24997
weighted avg	0.91	0.92	0.91	24997

4) Random Forest
   - Word-Count-vectors

	precision	recall	f1-score	support
0	0.70	0.20	0.31	2087
1	0.93	0.99	0.96	22910
accuracy			0.93	24997
macro avg	0.81	0.59	0.63	24997
weighted avg	0.91	0.93	0.91	24997

- Pretrained-sentence-embedding

	precision	recall	f1-score	support
0	0.80	0.12	0.20	2087
1	0.93	1.00	0.96	22910
accuracy			0.92	24997
macro avg	0.86	0.56	0.58	24997
weighted avg	0.92	0.92	0.90	24997

5) Adaboost

- Word-Count-vectors

	precision	recall	f1-score	support
0	0.50	0.10	0.17	2087
1	0.92	0.99	0.96	22910
accuracy			0.92	24997
macro avg	0.71	0.55	0.56	24997
weighted avg	0.89	0.92	0.89	24997

- Pretrained-sentence-embedding

	precision	recall	f1-score	support
0	0.53	0.19	0.28	2087
1	0.93	0.98	0.96	22910
accuracy			0.92	24997
macro avg	0.73	0.59	0.62	24997
weighted avg	0.90	0.92	0.90	24997

6) Neural Network

- Word-Count-vectors

	precision	recall	f1-score	support
0	0.67	0.22	0.33	2087
1	0.93	0.99	0.96	22910
accuracy			0.93	24997
macro avg	0.80	0.60	0.64	24997
weighted avg	0.91	0.93	0.91	24997

- Pretrained-sentence-embedding

	precision	recall	f1-score	support
0	0.66	0.26	0.38	2087
1	0.94	0.99	0.96	22910
accuracy			0.93	24997
macro avg	0.80	0.63	0.67	24997
weighted avg	0.91	0.93	0.91	24997

7) Distilled Bert

- Threshold 0.3

	precision	recall	f1-score	support
0	0.2	0.62	0.32	10433
1	0.96	0.79	0.87	114552
accuracy			0.78	124985
macro avg	0.59	0.71	0.59	124985
weighted avg	0.90	0.78	0.82	124985

- Threshold 0.4

	precision	recall	f1-score	support
0	0.51	0.36	0.43	42189
1	0.72	0.82	0.77	82796
accuracy			0.67	124985
macro avg	0.62	0.59	0.60	124985
weighted avg	0.65	0.67	0.65	124985

8) MP Net

- Threshold 0.3

	precision	recall	f1-score	support
0	0.09	0.44	0.15	10433
1	0.92	0.61	0.74	114552
accuracy			0.60	124985
macro avg	0.51	0.52	0.44	124985
weighted avg	0.85	0.60	0.69	124985

- Threshold 0.4

	precision	recall	f1-score	support
0	00.34	0.39	0.36	42189
1	0.66	0.61	0.63	82796
accuracy			0.53	124985
macro avg	0.50	0.50	0.50	124985
weighted avg	0.55	0.53	0.54	124985

NOTES

[1]https://monkeylearn.com/blog/sentiment-analysis-examples/
[2]https://towardsdatascience.com/sentiment-analysis-concept-analysis-and-applications-
[3]https://click360.ai/2019/07/06/the-credit-assignment-problem-in-marketing-attribution
[4]https://developer.datafiniti.co/docs/product-data-schema
[5]https://www.kaggle.com/datafiniti/consumer-reviews-of-amazon-products
[6]https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
[7]https://huggingface.co/sentence-transformers/all-mpnet-base-v2
[8]https://huggingface.co/distilbert-base-uncased-finetuned-sst-2-english
[9]https://huggingface.co/docs/transformers/model_doc/mpnet