SENTIMENT ANALYSIS

AMAZON REVIEWS DATASET

Group Members: Akanksha Arora, Suranjana Chowdhury, Xiaohong Liu, Sai Meghana Reddy Devarapalli

PART 1

1. Problem Statement:

We need to perform Sentiment Analysis on a regular basis across various industries. Sentiment Analysis not only helps businesses in gauging the common sentiment of the customers but also help them identify issues and act upon them in case the sentiment is not positive.

Deep learning provides us with **Recurrent Neural Network (RNN**) for handling **sequential data** (like **reviews dataset**) and with that we can do a wide range of task in machine learning and artificial intelligence.

Thus, our objective is *to train and build a neural network model (RNN) which can accurately predict sentiments of customer reviews.*

2. Dataset Description:

Our dataset is the Amazon fine food reviews (Kaggle) dataset that consists of about 500k (568,454) review comments and 10 features. *(The dataset chosen is different from the traditional numerical data, as in we would be working with text data for this analysis).* The reviews are from time Oct 1999 – Oct 2012. The dataset consists of features such as Product ID, Reviews ID, Review Text, Product rating, etc. Below is the screenshot of the dataset considered.

We will use RNN model to analyze review comments and predict if the **sentiment is good or bad.**

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	Good Quality Dog Food	I have bought several of the Vitality canned d...
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	1346976000	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4	1219017600	"Delight" says it all	This is a confection that has been around a fe...

3. Overview of task:

The tasks involved in this project are (i) data preprocessing, (ii) data exploration, (iii) tokenization, (iv) train test split, (v) model building and (vi) result analysis. We will mainly use **keras** package in Python for data processing and model fitting.

4. Dataset Preprocessing:

After loading the dataset, following pre-processing steps were carried out -

- **Duplicates**: We checked for duplicates and luckily, no duplicates were found.
- **Feature selection**: The columns not necessary for our analysis like 'Id', 'ProductId', 'UserId', 'ProfileName', 'Helpfulness Numerator', 'Helpfulness Denominator', 'Time', etc. were dropped.
- **Missing Values**: 'Summary' field has 27 missing values, but since other two columns were non-empty; we did not remove missing value rows.
- **Text Column Treatment**: We converted 'Text' column to data type string and applied lambda expression to split text and summary columns.
- **Adding Sentiment (Class) Field**: We added a new column 'Class' using the Score column. Wherever score was greater than 3 we labelled that as positive the others were labelled as negative.

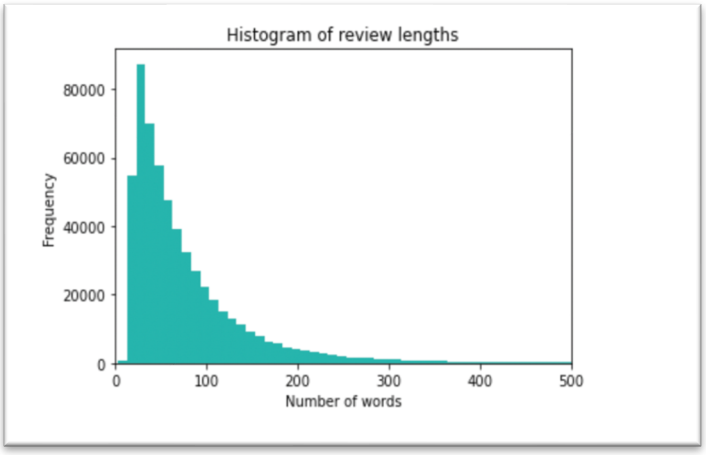
  We just kept two columns for analysis at the end – Text and Class Below is the screenshot of how the dataset looked like after following these steps.

	Text	Class
0	I have bought several of the Vitality canned d...	1
1	Product arrived labeled as Jumbo Salted Peanut...	0
2	This is a confection that has been around a fe...	1
3	If you are looking for the secret ingredient i...	0
4	Great taffy at a great price. There was a wid...	1

We also summarized the data for better understanding:

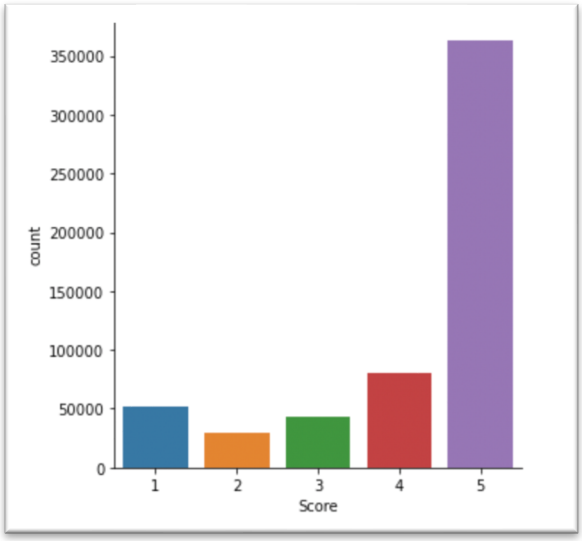
- Number of reviews: 568454
- Number of positive reviews: 443777
- Number of negative reviews: 124677
- Average review length: 80.26402312236347
- Average summary length: 4.113101499857509

5. Dataset Exploration:

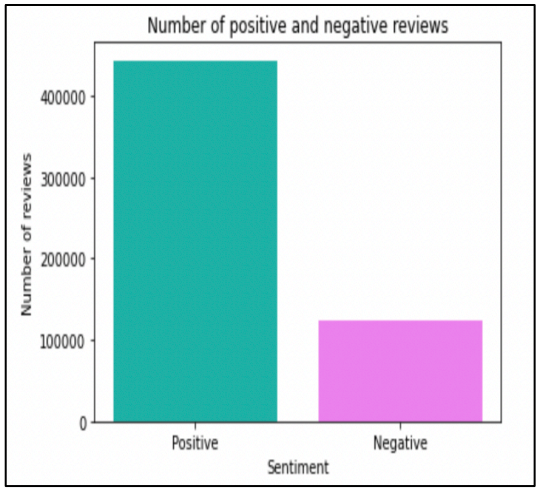


Most of the reviews are 20-40 words long & Average length: 80 words

We also checked things like score count, number of positive vs negative reviews:

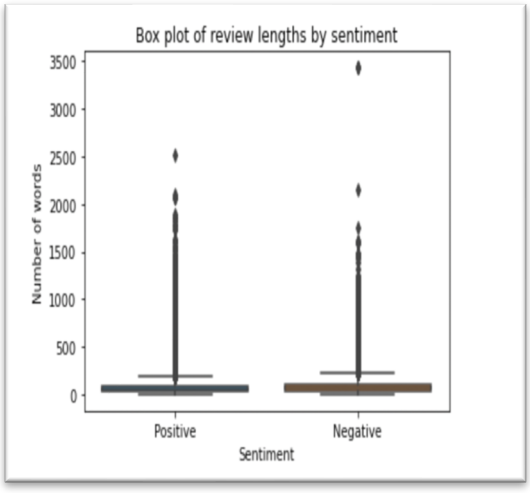


Majority of the data has >5 score; *Positive reviews > negative reviews*; *The model will be* ***Biased***



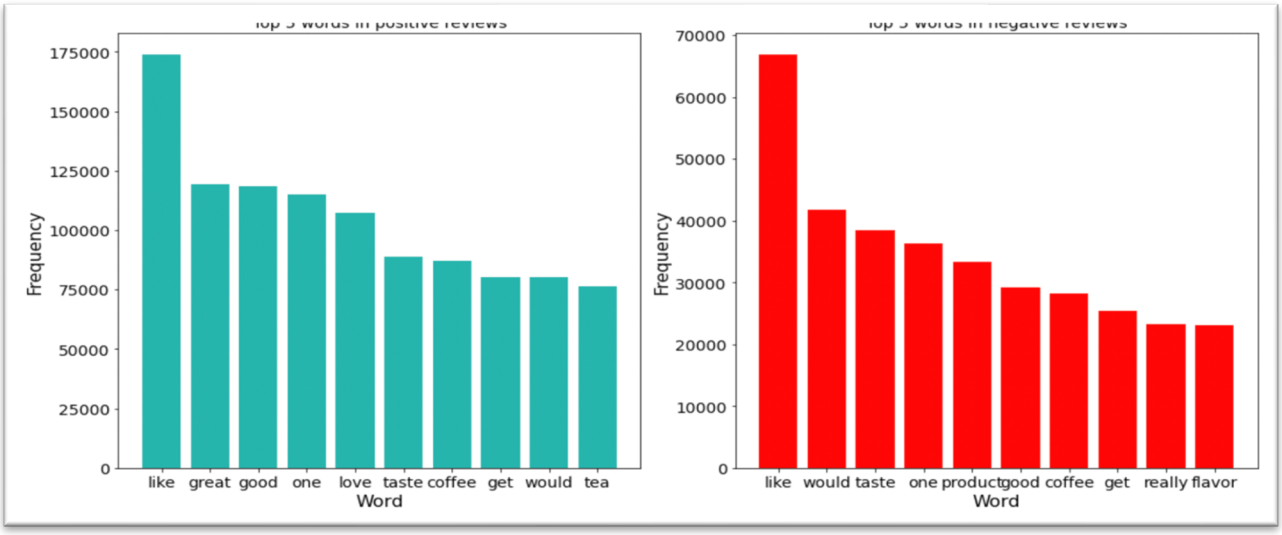
Score = 4, 5 Positive (~86%); *Score =1, 2, 3 Negative (~14%)*

We also wanted to see how the Box plots of review length by sentiment looks like:



Majority of reviews are less than 100 words

We also checked top 5 words by positive and negative reviews to understand the sentiment:



Positive reviews tend to include words like great, good, love etc.

Negative reviews also have similar top words as positive reviews. But do not have adjectives like good, great, etc.

6. Task/End Goal:

Our end goal is to build model to predict **sentiment** (ex: good, bad) using RNN. When fed with a review our model will be able to predict if our model has a good or bad sentiment.

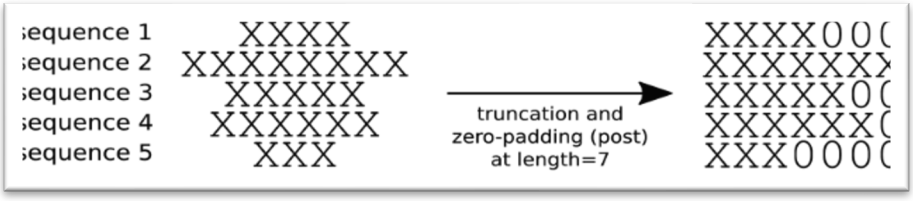
7. Why did we use RNN?

- RNNs are effective for processing sequential data which makes them well-suited for machine learning tasks that involve processing sequences, such as natural language processing, time-series analysis etc. In these tasks, the order of the data points matters, and RNNs can process this data in a way that preserves its sequential nature.
- RNNs retain a **memory** of previous inputs, which allows them to use this information.

Since we're working on sequential text data, RNNs are the most suitable choice for our analysis

8. Model Building:

- **Tokenization**
  Each word in the comment will be changed to integer (word index). Then each comment will be an integer sequence as follows:
  [13,345,277,123,3,4,94,22,120,5]
- **Padding**
  We use padding sequence in tokenizer of keras to let the sequences be the same length, because keras cannot deal with the unequal length sequences.



- **Test Train Split**
  Dataset is split into train (80%) and test (20%)

- **Model Fitting**
  We call keras package to define a sequential model and add layers one by one. For easy computing, the first layer will use the embedding method to decrease the input dimension, and the dropout probability between the RNN layer and the dense layer is 0.2. The sigmoid activation function is chosen in the dense layer to output a binary prediction. We also set the loss function to 'binary_crossentropy', optimizer to 'adam', and metrics to 'accuracy' We summarize the model:

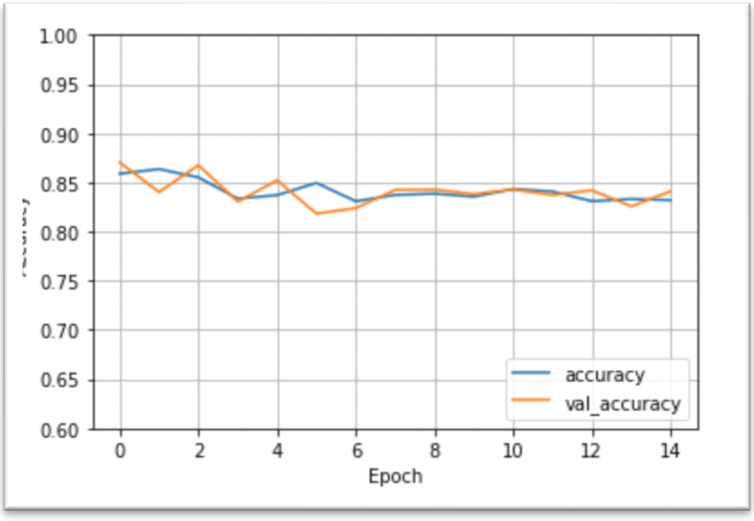
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 50, 30)            60000

 simple_rnn (SimpleRNN)      (None, 50)                4050

 dropout (Dropout)           (None, 50)                0

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 64,101
Trainable params: 64,101
Non-trainable params: 0
```

Because neural network models usually run too slowly, the epoch step number in our project is set to 15 so that we can get results in time. Letting the test set be validation data. The accuracy score and validation accuracy will be calculated in each step.

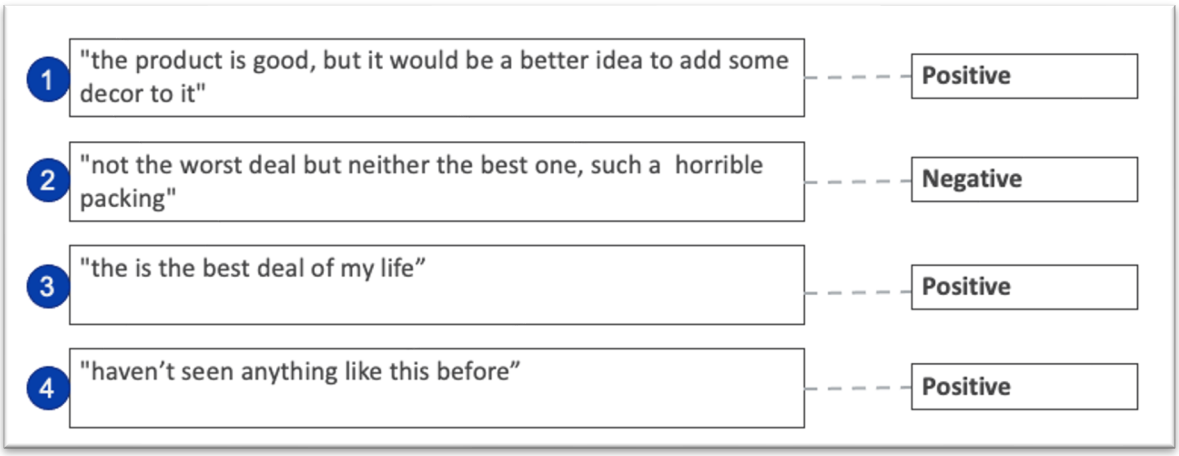
9. Model Fit and Accuracy

We plot both training accuracy and validation accuracy vs epoch as follows. From the chart, it is obvious that as the training accuracy changes , validation accuracy keeps from 0.82 to 0.87. Therefore, the RNN model performs well for this dataset.



10. Conclusion/Suggestion:

We stored the model and made it predict sample reviews. Model gave accurate predictions of – good, bad reviews after processing the input that was fed to it.



- *Reviews 2,3 and 4 are very clear if they are positive or negative and model accurately predicted them*
- *Review 1 looks like a balanced review, and hence model put it as positive*

Using the model, a string of reviews pertaining to a products can be parsed to judge the common sentiment
Negative reviews can be studied to figure out the issue and rectify them

PART 2

1. Description of the Game:

The Last Kingdom is a strategy game in which players must first build the defenses of their kingdom and then defend their castle from waves of enemy attacks. The game will be played on a grid-based map, and players must strategically place defense towers to stop the enemy units from reaching their castle.

The objective of the player would be to reach a level by constantly strengthening their defense by defeating enemies.

2. Actions, States, Rewards/Penalties:

At the beginning of the game, the player will start on level 1, will get 10 coins and 20 grid spaces to build the kingdom. One coin is equivalent to one tower. This means, player can setup 10 towers in level 1.

The player will climb up the levels by defeating enemies. The end objective will be to reach level 10. The player will be rewarded with points, coins, and grid spaces with every win. The coins can be used to build more and upgrade, and points will determine the level. Following is the level to points map.

Level	Points	Level	Points
1	0-4	6	21-24
2	5-8	7	25-28
3	9-12	8	29-32
4	13-16	9	33-36
5	17-20	10	37-40

The player wins the game by reaching level 10 or achieving 40 points. The player will not go back the level if the points drop.

Actions: The player can take multiple actions during the game. A few of the actions are listed below.

- Placing the towers in the allocated area of the kingdom to build defense.
- Upgrading the already deployed towers.
- Changing the position of the already deployed towers.

States: The state of the game will be defined by a function of:

- The number of the enemy waves already defeated.
- The positions and number of the towers placed.
- Number of coins collected.
- Number of points collected.
- Number of grid spaces.

Thus, the state would be different at each level, and would further change after every successful defend or defeat, change in number or position of the towers.

Rewards/Penalties:

I. The player receives a reward of +1 for successfully defending the kingdom against a wave, -1 for failing to defend against a wave, and 0 for all other actions.
II. The player will also receive 10 coins for successfully defending the caste. And will lose 5 coins as a penalty for failing to do so.
III. The player will receive 20 grid spaces with every increase in the level. (Only Reward, no penalties in this case)

The player will now be referred as the agent (Computer/software program/algorithm) which learns how to play a game by observing the current game state and selecting the best move to make based on that state. The agent will play lots of games, reward the agent for winning and penalize it for losing, and over time it will (hopefully) learn a good function representing the value of a state.

3. Hypothesis for 'Action Rule' and 'Reward and State Distribution':

The '**Action rule**' is the decision-making process that involves choosing the action with the highest expected reward. In RL, the agent learns to make decisions by interacting with an environment and receiving feedback in the form of rewards or penalties. The action rule in RL involves selecting an action that maximizes the expected future rewards, considering the current state and the available actions. *The agent will use a policy, which is a mapping from states to actions, to choose the best action.*

To do this in the game, the expected reward for each action is calculated (for instance, defeating an enemy is one action, installing a tower is another action). The agent will choose the action that has the highest expected reward. For instance, the agent will install a tower in grid space 1 instead of 2, if it results in defeat of the enemy (+1 point)

We can use a dueling Q-networks algorithm with rectified linear unit (ReLU) activation functions for our neural network. The input layer will take in the state of the game, and the output layer will represent the expected reward for each possible action.

The '**Reward and state distribution**' define the RL problem for the agent. The agent's goal is to learn a policy that maximizes the expected cumulative reward over time, given the state distribution of the environment.

The reward is scalar value that represents how good or bad the outcome of the action was for the agent. It is the primary means by which the agent learns to make decisions. In our game, the goal of the agent is to maximize the cumulative reward (coins, points, and grid spaces) it receives over time.

State distribution describes how often the agent is likely to encounter each possible state while it is trying to achieve its goal. The state distribution will be based on a sliding window approach that considers the previous few states to

capture temporal (time based) information. By considering this temporal (time based) information, the agent can make more accurate predictions.

4. **Final Reward:**

The "**final reward**" refers to the reward that the agent receives at the end of an episode (series of actions). The final reward is used in combination with the cumulative reward to evaluate the agent's overall performance over multiple actions.

The final reward will be cumulative sum of binary outputs of how many enemy waves were defeated. If the player is successful, the reward is +1, and if the player fails, the reward is -1. Thus, if 10 enemy waves were dealt with, out of which 8 were defeated, the final reward would be 6 (+8-2) points, level 2, 70 (+80-10) coins, 20 more grid spaces.

5. **Analysis Procedure:**

The analysis procedure for this game will involve implementing the dueling Q-networks algorithm with ReLU activation functions, and the sliding window approach for state distribution will be used. The analysis procedure will involve iteratively training, evaluating, and refining the agent until it achieves satisfactory performance in the game (that is reach level 10 as soon as possible).

Since the agent learns by interacting with the environment, receiving feedback in the form of rewards, and adjusting its behavior to maximize the expected cumulative reward over time; so, first we will train the agent to interact with the game for 100,000 turns and then test its performance against a set of predetermined scenarios. We will judge how well the agent performs by measuring its rewards (both intermediate and cumulative) at every step. For instance, one of the performance criteria can be that the agent should ideally have more than 50% success rate in defeating the enemy waves. If the agent performs well, we will consider it successful and move on to more testing and improvements. However, if it does not perform well, we will make modifications to the chosen neural network and re-train.

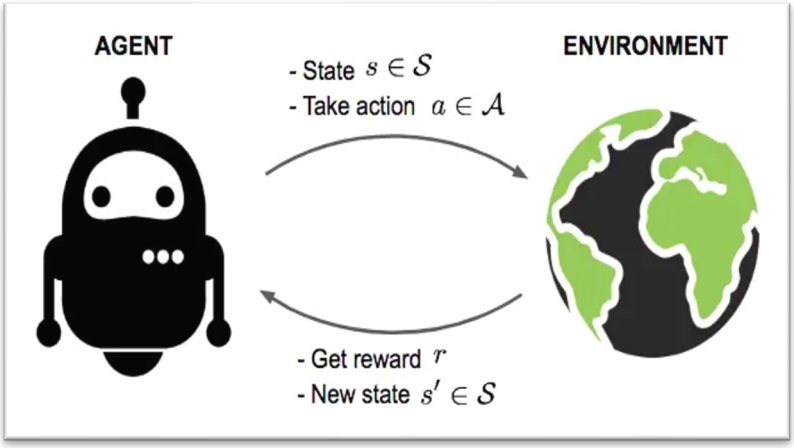


**Image for reference purposes*