Sentiment Analysis on Amazon Fine Food Reviews using Machine learning in Python

We will be doing some sentiment analysis on Amazon Fine Food Reviews in python using two different techniques:

1. VADER (Valence Aware Dictionary and sEntiment Reasoner) - Bag of words approach
2. Roberta Pretrained Model from 🤗
3. Huggingface Pipeline

Dataset link: https://www.kaggle.com/datasets/snap/amazon-fine-food-reviews

This dataset consists of reviews of fine foods from amazon. The data span a period of more than 10 years, including all ~500,000 reviews up to October 2012. Reviews include product and user information, ratings, and a plain text review. It also includes reviews from all other Amazon categories.

Step 1. Read in Data and import libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

plt.style.use('ggplot')

import nltk
```

```
# Read in data
df = pd.read_csv('/content/Reviews.csv')
print(df.shape)
df = df.head(50) #To show first 50 records
print(df.shape) #To check the number of Rows and Columns in The Dataset
```

```
(568454, 10)
(50, 10)
```

```
df.head(5)
```

	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...

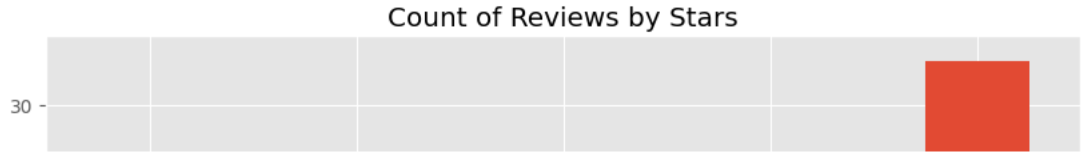
```
df['Text'].values[0]
```

```
'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.'
```

Exploratory Data Analysis

```
# Lets check the number of unique values present in Score column and the count it
df['Score'].value_counts().sort_index().plot(kind='bar', title='Count of Reviews by Stars', figsize=(10,5))
```

```
<Axes: title={'center': 'Count of Reviews by Stars'}>
```




```
#Modified code to plot the graph
ax = df['Score'].value_counts().sort_index() \
    .plot(kind='bar',
          title='Count of Reviews by Stars',
          figsize=(10, 5))
ax.set_xlabel('Review Stars')
plt.show()
```

What we get to Know: Most of the reviews are 5 Starts but it but then it kind of goes down and it has a little uptake in the number of 1 Star reviews we have.

NLTK Implementation

```
example = df['Text'][5]
print(example)
```

```
I got a wild hair for taffy and ordered this five pound bag. The taffy was all very enjoyable with many flavors: watermelon, root beer, melon, pep
```

```
# Tokenization-- Splitting the sentence into words by space
nltk.download("all") #Downloading all modules from NLTK
tokens = nltk.word_tokenize(example)
tokens[:10] #showing the first 10 tokens
```

```
[nltk_data]    |   Package sentence_polarity is already up-to-date!
[nltk_data]    | Downloading package sentiwordnet to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package sentiwordnet is already up-to-date!
[nltk_data]    | Downloading package shakespeare to /root/nltk_data...
[nltk_data]    |   Package shakespeare is already up-to-date!
[nltk_data]    | Downloading package sinica_treebank to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package sinica_treebank is already up-to-date!
[nltk_data]    | Downloading package smultron to /root/nltk_data...
[nltk_data]    |   Package smultron is already up-to-date!
[nltk_data]    | Downloading package snowball_data to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package snowball_data is already up-to-date!
[nltk_data]    | Downloading package spanish_grammars to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package spanish_grammars is already up-to-date!
[nltk_data]    | Downloading package state_union to /root/nltk_data...
[nltk_data]    |   Package state_union is already up-to-date!
[nltk_data]    | Downloading package stopwords to /root/nltk_data...
[nltk_data]    |   Package stopwords is already up-to-date!
[nltk_data]    | Downloading package subjectivity to
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Package subjectivity is already up-to-date!
[nltk_data]    | Downloading package swadesh to /root/nltk_data...
[nltk_data]    |   Package swadesh is already up-to-date!
[nltk_data]    | Downloading package switchboard to /root/nltk_data...
[nltk_data]    |   Package switchboard is already up-to-date!
[nltk_data]    | Downloading package tagsets to /root/nltk_data...
```

```
# Part-of-speech tagging: words related to its Part of Speech
tagged = nltk.pos_tag(tokens)
tagged[:10]
```

```
[('I', 'PRP'),
 ('got', 'VBD'),
 ('a', 'DT'),
 ('wild', 'JJ'),
 ('hair', 'NN'),
 ('for', 'IN'),
 ('taffy', 'NN'),
 ('and', 'CC'),
 ('ordered', 'VBD'),
 ('this', 'DT')]
```

```
# Named entity chunking
entities = nltk.chunk.ne_chunk(tagged)
entities.pprint()  # Print the named entities
```

```
  flavors/NNS
  :/:
  watermelon/NN
  ,/,
  root/NN
  beer/NN
  ,/,
  melon/NN
  ,/,
  peppermint/NN
  ,/,
  grape/NN
  ,/,
  etc/FW
  ./.
  My/PRP$
  only/JJ
  complaint/NN
  is/VBZ
  there/EX
  was/VBD
  a/DT
```

```
,/,
this/DT
lasted/VBN
only/RB
two/CD
weeks/NNS
!/.
I/PRP
would/MD
recommend/VB
this/DT
brand/NN
of/IN
```

Step 2. VADER Seniment Scoring

We will use NLTK's `SentimentIntensityAnalyzer` to get the neg(negative)/neu(neutral)/pos(positive) scores of the text.

- This uses a "bag of words" approach:
    1. Stop words are removed
    2. each word is scored and combined to a total score.

```
from nltk.sentiment import SentimentIntensityAnalyzer
from tqdm.notebook import tqdm

sia = SentimentIntensityAnalyzer() # Creating an Object of the class
```

The **SentimentIntensityAnalyzer** class from NLTK's nltk.sentiment module is a pre-trained sentiment analysis tool that can be used to analyze the sentiment (positive, negative, neutral, or compound) of textual data.

The tqdm module is a library for creating progress bars in Python. The notebook module variant is specifically designed for use in Jupyter Notebook or JupyterLab environments.

```
sia.polarity_scores('I am so happy!')
```

```
{'neg': 0.0, 'neu': 0.318, 'pos': 0.682, 'compound': 0.6468}
```

The polarity_scores method is a function provided by the SentimentIntensityAnalyzer class from NLTK's sentiment module. It is used to calculate sentiment polarity scores for a given text

```
sia.polarity_scores('This is the worst thing ever.')
```

```
{'neg': 0.451, 'neu': 0.549, 'pos': 0.0, 'compound': -0.6249}
```

```
sia.polarity_scores(example)
```

```
{'neg': 0.029, 'neu': 0.809, 'pos': 0.163, 'compound': 0.883}
```

```
# Run the polarity score on the entire dataset
res = {}
for i, row in tqdm(df.iterrows(), total=len(df)):
    text = row['Text']
    myid = row['Id']
    res[myid] = sia.polarity_scores(text)
```

100% 50/50 [00:00<00:00, 662.58it/s]

The code is iterating over each row in the DataFrame df and calculating the polarity scores using the SentimentIntensityAnalyzer on the 'Text' column. It stores the results in a dictionary res, where the 'Id' column values are used as keys.

```
res
```

```
{1: {'neg': 0.0, 'neu': 0.695, 'pos': 0.305, 'compound': 0.9441},
 2: {'neg': 0.138, 'neu': 0.862, 'pos': 0.0, 'compound': -0.5664},
 3: {'neg': 0.091, 'neu': 0.754, 'pos': 0.155, 'compound': 0.8265},
 4: {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0},
 5: {'neg': 0.0, 'neu': 0.552, 'pos': 0.448, 'compound': 0.9468},
 6: {'neg': 0.029, 'neu': 0.809, 'pos': 0.163, 'compound': 0.883},
 7: {'neg': 0.034, 'neu': 0.693, 'pos': 0.273, 'compound': 0.9346},
 8: {'neg': 0.0, 'neu': 0.52, 'pos': 0.48, 'compound': 0.9487},
 9: {'neg': 0.0, 'neu': 0.851, 'pos': 0.149, 'compound': 0.6369},
 10: {'neg': 0.0, 'neu': 0.705, 'pos': 0.295, 'compound': 0.8313},
 11: {'neg': 0.017, 'neu': 0.846, 'pos': 0.137, 'compound': 0.9746},
 12: {'neg': 0.113, 'neu': 0.887, 'pos': 0.0, 'compound': -0.7579},
 13: {'neg': 0.031, 'neu': 0.923, 'pos': 0.046, 'compound': 0.296},
 14: {'neg': 0.0, 'neu': 0.355, 'pos': 0.645, 'compound': 0.9466},
 15: {'neg': 0.104, 'neu': 0.632, 'pos': 0.264, 'compound': 0.6486},
 16: {'neg': 0.0, 'neu': 0.861, 'pos': 0.139, 'compound': 0.5719},
 17: {'neg': 0.097, 'neu': 0.694, 'pos': 0.209, 'compound': 0.7481},
 18: {'neg': 0.0, 'neu': 0.61, 'pos': 0.39, 'compound': 0.8883},
 19: {'neg': 0.012, 'neu': 0.885, 'pos': 0.103, 'compound': 0.8957},
```

```
    20: {'neg': 0.0, 'neu': 0.863, 'pos': 0.137, 'compound': 0.6077},
    21: {'neg': 0.0, 'neu': 0.865, 'pos': 0.135, 'compound': 0.6249},
    22: {'neg': 0.0, 'neu': 0.739, 'pos': 0.261, 'compound': 0.9153},
    23: {'neg': 0.0, 'neu': 0.768, 'pos': 0.232, 'compound': 0.7687},
    24: {'neg': 0.085, 'neu': 0.771, 'pos': 0.143, 'compound': 0.2617},
    25: {'neg': 0.038, 'neu': 0.895, 'pos': 0.068, 'compound': 0.3939},
    26: {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0},
    27: {'neg': 0.128, 'neu': 0.872, 'pos': 0.0, 'compound': -0.296},
    28: {'neg': 0.04, 'neu': 0.808, 'pos': 0.152, 'compound': 0.5956},
    29: {'neg': 0.022, 'neu': 0.669, 'pos': 0.309, 'compound': 0.9913},
    30: {'neg': 0.017, 'neu': 0.846, 'pos': 0.137, 'compound': 0.9746},
    31: {'neg': 0.041, 'neu': 0.692, 'pos': 0.267, 'compound': 0.9713},
    32: {'neg': 0.0, 'neu': 0.484, 'pos': 0.516, 'compound': 0.9153},
    33: {'neg': 0.069, 'neu': 0.839, 'pos': 0.092, 'compound': 0.7103},
    34: {'neg': 0.024, 'neu': 0.72, 'pos': 0.256, 'compound': 0.9779},
    35: {'neg': 0.0, 'neu': 0.874, 'pos': 0.126, 'compound': 0.9091},
    36: {'neg': 0.024, 'neu': 0.821, 'pos': 0.155, 'compound': 0.7622},
    37: {'neg': 0.0, 'neu': 0.754, 'pos': 0.246, 'compound': 0.9196},
    38: {'neg': 0.0, 'neu': 0.938, 'pos': 0.062, 'compound': 0.4457},
    39: {'neg': 0.05, 'neu': 0.846, 'pos': 0.104, 'compound': 0.7638},
    40: {'neg': 0.0, 'neu': 0.856, 'pos': 0.144, 'compound': 0.8114},
    41: {'neg': 0.033, 'neu': 0.82, 'pos': 0.147, 'compound': 0.9301},
    42: {'neg': 0.03, 'neu': 0.848, 'pos': 0.122, 'compound': 0.9435},
    43: {'neg': 0.0, 'neu': 0.588, 'pos': 0.412, 'compound': 0.9441},
    44: {'neg': 0.0, 'neu': 0.685, 'pos': 0.315, 'compound': 0.9161},
    45: {'neg': 0.031, 'neu': 0.778, 'pos': 0.191, 'compound': 0.8421},
    46: {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0},
    47: {'neg': 0.0, 'neu': 0.737, 'pos': 0.263, 'compound': 0.9169},
    48: {'neg': 0.0, 'neu': 0.868, 'pos': 0.132, 'compound': 0.4404},
    49: {'neg': 0.0, 'neu': 0.821, 'pos': 0.179, 'compound': 0.747},
    50: {'neg': 0.056, 'neu': 0.865, 'pos': 0.079, 'compound': 0.2363}}
```

```
pd.DataFrame(res).T
```

	neg	neu	pos	compound
1	0.000	0.695	0.305	0.9441
2	0.138	0.862	0.000	-0.5664
3	0.091	0.754	0.155	0.8265
4	0.000	1.000	0.000	0.0000
5	0.000	0.552	0.448	0.9468
6	0.029	0.809	0.163	0.8830
7	0.034	0.693	0.273	0.9346
8	0.000	0.520	0.480	0.9487
9	0.000	0.851	0.149	0.6369
10	0.000	0.705	0.295	0.8313
11	0.017	0.846	0.137	0.9746
12	0.113	0.887	0.000	-0.7579
13	0.031	0.923	0.046	0.2960
14	0.000	0.355	0.645	0.9466
15	0.104	0.632	0.264	0.6486
16	0.000	0.861	0.139	0.5719

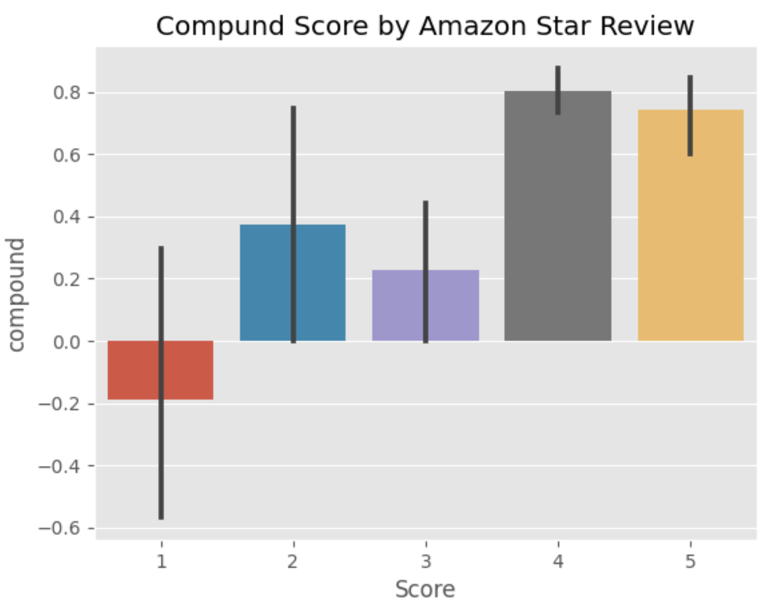
```
vaders = pd.DataFrame(res).T
vaders = vaders.reset_index().rename(columns={'index': 'Id'})
vaders = vaders.merge(df, how='left')
```

```
# Now we have sentiment score and metadata
vaders.head()
```

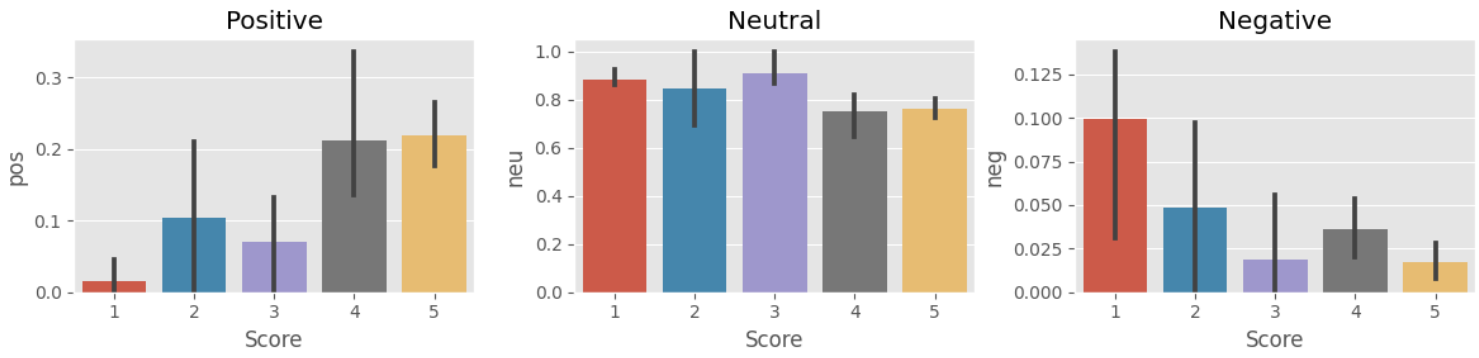
	Id	neg	neu	pos	compound	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time
0	1	0.000	0.695	0.305	0.9441	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400
1	2	0.138	0.862	0.000	-0.5664	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	1346976000

Plot VADER results

```
ax = sns.barplot(data=vaders, x='Score', y='compound')
ax.set_title('Compund Score by Amazon Star Review')
plt.show()
```



```
fig, axs = plt.subplots(1, 3, figsize=(12, 3))
sns.barplot(data=vaders, x='Score', y='pos', ax=axs[0])
sns.barplot(data=vaders, x='Score', y='neu', ax=axs[1])
sns.barplot(data=vaders, x='Score', y='neg', ax=axs[2])
axs[0].set_title('Positive')
axs[1].set_title('Neutral')
axs[2].set_title('Negative')
plt.tight_layout()
plt.show()
```



Draw Back of Vader Model: VADER's scoring is based on a pre-defined lexicon of words and their associated sentiment scores. This lexicon may not capture the full context or subjectivity of certain expressions or phrases, leading to potential inaccuracies in sentiment classification. The model may struggle with sarcasm, irony, or nuanced language that requires deeper understanding.

Step 3. Roberta Pretrained Model

- Use a model trained of a large corpus of data.
- Transformer model accounts for the words but also the context related to other words.

```
!pip install transformers
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-packages (4.29.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transformers) (3.12.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.14.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.14.1)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (1.22.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from transformers) (23.1)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (6.0)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (2022.10.31)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transformers) (2.27.1)
Requirement already satisfied: tokenizers!=0.11.3,<0.14,>=0.11.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.13.3)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from transformers) (4.65.0)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.14.1->transformers) (2023.4.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.14.1->transform
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (1.26.15)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2022.12.7)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2.0.12)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.4)
```

```
from transformers import AutoTokenizer
from transformers import AutoModelForSequenceClassification
from scipy.special import softmax
```

AutoTokenizer: This class from the Transformers library is used to automatically load the appropriate tokenizer for a specific pre-trained model. It simplifies the process of tokenizing input text to prepare it for model input.

AutoModelForSequenceClassification: This class is used to automatically load the appropriate pre-trained model for sequence classification. It provides a high-level interface to load and use pre-trained models for classifying sequences of text.

softmax: This is a function from the scipy.special module that is used to compute the softmax probabilities. The softmax function normalizes the output logits to produce a probability distribution over multiple classes.

```
MODEL = f"cardiffnlp/twitter-roberta-base-sentiment"
tokenizer = AutoTokenizer.from_pretrained(MODEL)
model = AutoModelForSequenceClassification.from_pretrained(MODEL)
```

MODEL: This variable stores the identifier or name of the pre-trained sentiment analysis model you want to use. In this case, it is set to "cardiffnlp/twitter-roberta-base-sentiment". This model is trained on Twitter data and is based on the Roberta architecture.

tokenizer: Using the AutoTokenizer.from_pretrained() method, the tokenizer corresponding to the specified model is loaded. The tokenizer is responsible for converting input text into tokens that the model can process.

model: The AutoModelForSequenceClassification.from_pretrained() method loads the pre-trained model for sequence classification. This model is specifically designed for sentiment analysis tasks, where the input is a sequence of text, and the model predicts the sentiment associated with it.

```
# VADER results on example
print(example)
sia.polarity_scores(example)
```

```
I got a wild hair for taffy and ordered this five pound bag. The taffy was all very enjoyable with many flavors: watermelon, root beer, melon, pep
{'neg': 0.029, 'neu': 0.809, 'pos': 0.163, 'compound': 0.883}
```

```
# Run for Roberta Model
encoded_text = tokenizer(example, return_tensors='pt')
output = model(**encoded_text)
scores = output[0][0].detach().numpy()
scores = softmax(scores)
scores_dict = {
    'roberta_neg' : scores[0],
    'roberta_neu' : scores[1],
    'roberta_pos' : scores[2]
}
print(scores_dict)
```

```
{'roberta_neg': 0.006129598, 'roberta_neu': 0.021795882, 'roberta_pos': 0.97207445}
```

```
def polarity_scores_roberta(example):
    encoded_text = tokenizer(example, return_tensors='pt')
    output = model(**encoded_text)
    scores = output[0][0].detach().numpy()
    scores = softmax(scores)
    scores_dict = {
        'roberta_neg' : scores[0],
        'roberta_neu' : scores[1],
        'roberta_pos' : scores[2]
    }
    return scores_dict
```

```
res = {}
for i, row in tqdm(df.iterrows(), total=len(df)):
    try:
        text = row['Text']
        myid = row['Id']
        vader_result = sia.polarity_scores(text)
        vader_result_rename = {}
        for key, value in vader_result.items():
            vader_result_rename[f"vader_{key}"] = value
        roberta_result = polarity_scores_roberta(text)
        both = {**vader_result_rename, **roberta_result}
        res[myid] = both
    except RuntimeError:
        print(f'Broke for id {myid}')
```

100% 50/50 [00:26<00:00, 2.65it/s]

```
results_df = pd.DataFrame(res).T
results_df = results_df.reset_index().rename(columns={'index': 'Id'})
results_df = results_df.merge(df, how='left')
```

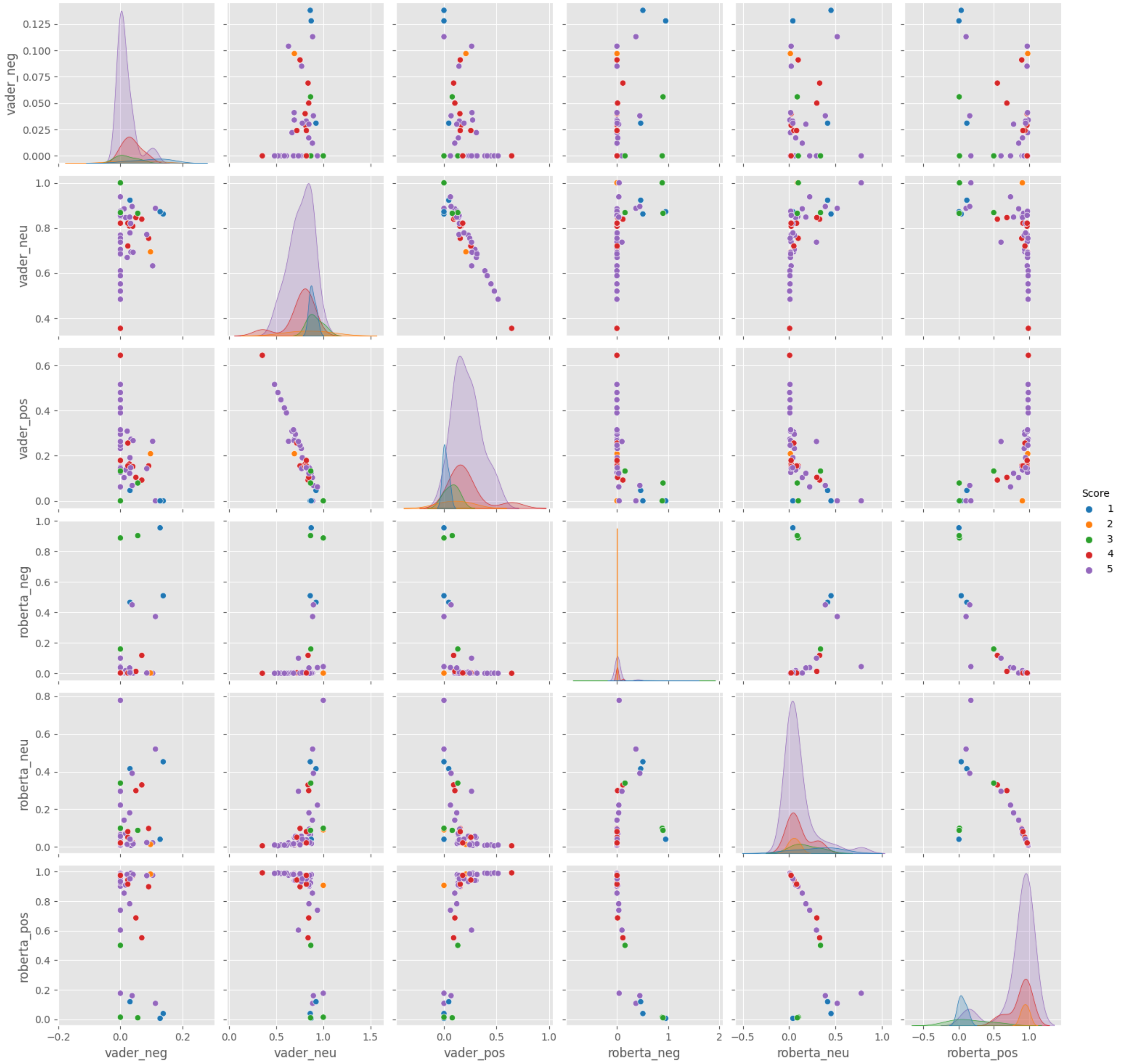
Compare Scores between models

```
results_df.columns
```

```
Index(['Id', 'vader_neg', 'vader_neu', 'vader_pos', 'vader_compound',
       'roberta_neg', 'roberta_neu', 'roberta_pos', 'ProductId', 'UserId',
       'ProfileName', 'HelpfulnessNumerator', 'HelpfulnessDenominator',
       'Score', 'Time', 'Summary', 'Text'],
      dtype='object')
```

Step 3. Combine and compare

```
sns.pairplot(data=results_df,
             vars=['vader_neg', 'vader_neu', 'vader_pos',
                  'roberta_neg', 'roberta_neu', 'roberta_pos'],
            hue='Score',
            palette='tab10')
plt.show()
```



Step 4: Review Examples:

- Positive 1-Star and Negative 5-Star Reviews

Lets look at some examples where the model scoring and review score differ the most.

```
results_df.query('Score == 1') \
    .sort_values('roberta_pos', ascending=False)['Text'].values[0]
```

    'My cats have been happily eating Felidae Platinum for more than two years. I just got a new b

```
results_df.query('Score == 1') \
    .sort_values('vader_pos', ascending=False)['Text'].values[0]
```

    'My cats have been happily eating Felidae Platinum for more than two years. I just got a new bag and the shape of the food is different. They tri
    ed the new food when I first put it in their bowls and now the bowls sit full and the kitties will not touch the food. I've noticed similar revie
    ws related to formula changes in the past. Unfortunately, I now need to find a new food that my cats will eat.'

```
# nevative sentiment 5-Star view
```

```
results_df.query('Score == 5') \
    .sort_values('roberta_neg', ascending=False)['Text'].values[0]
```

    'I have lived out of the US for over 7 yrs now, and I so miss my Twizzlers!! When I go back t
    o visit or someone visits me, I always stock up. All I can say is YUM!
Sell these in Mex
    ico and you will have a faithful buyer, more often than I'm able to buy them right now.'

```
results_df.query('Score == 5') \
    .sort_values('vader_neg', ascending=False)['Text'].values[0]
```

    'One of my boys needed to lose some weight and the other didn't. I put this food on the floor for the chubby guy, and the protein-rich, no by-pr
    oduct food up higher where only my skinny boy can jump. The higher food sits going stale. They both really go for this food. And my chubby boy
    has been losing about an ounce a week.'

Extra: The Transformers Pipeline

- Quick & easy way to run sentiment predictions

```
from transformers import pipeline

sent_pipeline = pipeline("sentiment-analysis")
```

    No model was supplied, defaulted to distilbert-base-uncased-finetuned-sst-2-english and revisi
    Using a pipeline without specifying a model name and revision in production is not recommended

Downloading (…)lve/main/config.json: 100% 629/629 [00:00<00:00, 41.9kB/s]

Downloading pytorch_model.bin: 100% 268M/268M [00:02<00:00, 72.9MB/s]

Downloading (…)okenizer_config.json: 48.0/48.0 [00:00<00:00,

```
sent_pipeline('I love sentiment analysis!')
```

    [{'label': 'POSITIVE', 'score': 0.9997853636741638}]

```
sent_pipeline('Make sure to like and subscribe!')
```

    [{'label': 'POSITIVE', 'score': 0.9991742968559265}]

```
sent_pipeline('booo')
```

    [{'label': 'NEGATIVE', 'score': 0.9936267137527466}]

The End