Lauren Kidman
COSC74: Machine Learning and Statistical Data Analysis
24F – 13 November 2024

Report for HW4: Amazon Review Binary Classification Task

Table of Contents:

High-Level Overview

In this homework, you will be given product reviews from Amazon. Your task is to perform binary classification with machine learning (ML) models and Large Language Models (LLMs), where the goal is to identify if the review text represents a high-star review or a low-star review. For this task, we define a high-star review as one with a score > 3 (i.e. a 4 or 5 star review), with low-star reviews being <= 3 (i.e. 1,2,3 star reviews).

Part 1: *Classification with Machine Learning Models*

1. <u>Description</u>

For part 1, your goal is to build binary classification models and report the scores of the models. As deliverables of this part, you will submit your best model prediction on the provided test set. Your best model performance on the test set would outperform the baseline score provided by the instructor. The baseline score for part 1 is 80.00.

The goal of this write-up is for you to a) explain what you learned about the data, b) describe what processing and modeling choices you made, and c) discuss how your model performed, given your choices.

2. Data

Describe the dataset you worked with, including explanations of interesting feature variables and the target variable related to this task. Highlight how you have done feature engineering for the binary classification task.

The dataset being analyzed contains various information about Amazon reviews for a variety of products; this information is grouped into "features" which indicate insight into user experiences, sentiments, verification, and product quality. Some of the more notable features are listed:

overall: This is the product's rating on Amazon.
reviewText: The textual content of the body of the review.
summary: A high-level summary of the review.
vote: How many people found the review helpful.

The target variable, **label**, is a binary classification for each review: 0 represents a low-star review, which is given to one of negative or neutral sentiment, while 1 represents a high-star review, given to one of positive sentiment. Given this goal, the above listed features were my target focus when analyzing the data, as they give the greatest insight into user satisfaction.

As discussed in Lecture 12, when applying machine learning models to text data, feature engineering and text processing are critical for removing and manipulating words that are not informative for the task at hand. Raw data often contains elements that do not contribute to or aid the model's predictability, such as punctuation, "stop words" (i.e. words like "and", "the", etc.), and inconsistently formatted text (like upper case vs. lower case)--in fact, these elements often introduce noise, therefore increasing the complexity of the model and reducing the overall accuracy of its predictions. Thus, to prepare the amazon data for modeling, I implemented two key feature engineering/text processing techniques:

1. *Combining features*

At first, I utilized only reviewText for my analysis, as I figured using both reviewText and summary would create redundancy and be overly complex. However, looking at the data, while a distinct sentiment can for the most part be reliably identified simply from reviewText, many rows contain unnecessary details and punctuation that muddle the data; meanwhile the summary column is more focused and more directly indicative of sentiment, in some cases outright saying the star rating of the review. Furthermore, there were also instances of discrepancies between the review text and the summary—for example, one review said "outstanding", but the summary indicated the reviewer gave the product one star. As a result, I chose to combine the data from both columns into a single text field, combinedData, which ultimately helped to improve my results (to be discussed later).

2. *Text cleaning*

I also wrote a function to address some of the raw data issues outlined above that standardizes the text being passed to the model. The text is converted entirely to lowercase using .lower(), punctuation and non-alphabetical characters are removed using .isalpha() and .isspace(), and extra spaces are stripped using .strip(). This makes it easier for the model to identify patterns within the data.

3. Methods

Describe the steps you took to build your model. Describe the hyperparameters you used in your best model. Describe how you did your hyperparameter search.

Before beginning, I first needed to choose the three classifier models I wanted to implement; in reviewing previous homeworks, I settled on Logistic Regression as it was the main focus for multiple homeworks, Support Vector Machine (SVM), and Naive Bayes, as I wanted to include a non-linear model to see how it handled the data in contrast to the other two models. **In the end, SVM produced the best F1 Macro score.**

The first step to begin building and training my models was converting the cleaned text into a format suitable for machine learning models–"Every machine learning paradigm requires that the data we deal with consists strictly of "numerical values." Thus, I used TfidfVectorizer to convert words into numerical features. I chose this approach, as opposed to something like the Bag-of-Words (BOW) approach, as Tfid weighs each word based on its frequency across the entire corpus–this ensures that common words (ex: "the", "and") are down-weighted, while more unique and informative words are given higher importance.

In using TfidfVectorizer, I implemented two feature engineering parameters to further help process the data:

1. *N-grams*

In looking at the TfidfVectorizer sci-kit learn documentation, I was intrigued to see that n_gram_range is an optional function parameter. Researching online, the definition of an n-gram is given as "a continuous sequence of n items (in this case, words) from a given text"–unigram (1,1) means a single word, bigram (2,2) means pairs of consecutive words, and so on. The default for the TfidfVectorizer is a range of (1,1), i.e only considering single words. I decided to change the range to include two- and three- word phrases so the model is able to detect better relationships between words, as oftentimes word combinations convey more meaning than single words alone. A classic example of why this subtle change makes a difference is recognizing that the model will read "not" and "good" separately, which can potentially be interpreted as positive, than reading "not good" together, which is clearly negative. I did not initially include n-grams in my vectorizer, and it significantly improved my results once I did, the best results coming with a range of (1,3).

2. *Max features*

The max_features parameter sets a limit on the number of features (in this case unique words or phrases) that the vectorizer will consider–it chooses these words based on their importance, indicated by their TF-IDF score–and thus ensures only the most informative words are used. Not only does it improve model performance but it also keeps the amount of data being processed (the feature space) to a minimum, quickening performance. Setting max_features to a smaller number such as 10, I found it did not handle data well as unless the specific 10 words were in a review (and reviews are highly variable) it could not give an output. On the other hand, I found setting max_features to anything over 1000 makes the code take multiple minutes to process, which is a waste of time and space.

I initially had stop_words='english', which would remove stop words from the data automatically before testing. I thought this was a very straightforward thing to do–however, contrary to my thoughts, I found that including stop_words actually improved the scores for every classifier, by as much as 0.45.

After establishing the vectorizer, it was important to decide the hyperparameters for each classifier I was going to use. The search process for finding these hyperparameters was the same for all of the classifiers I chose–I dug through the sci-kit learn documentation. Focusing on my winning model, SVM, I had to decide if I was going to use LinearSVC or the standard SVC class–The main difference between the two is that LinearSVC is targeted towards data that is known to be linearly separable, meanwhile SVC is more flexible, utilizing both non-linear and linear "kernels" to capture complex relationships within data. In addition, because it is specialized for linear problems, LinearSVC is generally faster and more memory-efficient compared to SVC when dealing with large datasets. I began testing first with standard SVC, but found it was extremely slow–taking upwards of 30 minutes to complete. Playing around with the kernel function did not change this. Knowing that LinearSVC worked better on larger datasets by being faster to train, and inferencing that the amazon data was somewhat linear (I had the added context that the best solver for my Logistic Regression model was liblinear), I decided to switch to LinearSVC, which ran and finished in less than 5 seconds and produced good results.

Once I had established the LinearSVC package, hyperparameter tuning was a straightforward yet crucial step to optimize the model's performance. To find the best combination of hyperparameters, I used GridSearchCV with 5-fold cross-validation, as specified within the assignment instructions. The GridSearch approach systematically searches through a predefined set of hyperparameter values and evaluates each combination based on the chosen scoring metric, which in this case was F1 Macro (in order to exceed the 0.80 baseline). The parameters I chose to optimize in my parameter grid:

1. *Penalty*

As said in Lecture 6, penalty, also known as regularization, is a process of introducing additional constraints or information to the model in order to prevent overfitting. For this classifier, there are two types of penalty, L1 (absolute value constraint) and L2 (squared value constraint), both of which are tested.

2. *C → regularization strength*

C controls the trade off between achieving a low training error and a low testing error–the goal being to fit as close as possible to the training data without overfitting. I gave multiples of 10 (0.01, 0.1, 1, 10) in order to give a broad range of values to explore this trade off.

3. *Loss Function*

As discussed in Lecture 2, the loss function measures how well the model's predictions align with the actual target values–it quantifies the difference between the predicted and true labels, which is what we are trying to minimize as a whole. To determine which loss function worked best, I used the options provided in the scikit-learn documentation and tested both hinge and squared_hinge.

4. Results

Show the precision, recall, and macro F1 score for the best set of hyperparameters using 5-fold cross-validation for each of your three models. What is a suitable metric for this task? Does your best model provide the best performance across all metrics? If so, why? Which class is more challenging to predict? Can you reason why they are harder to predict?

My table of results can be seen below:
blue indicates highest value amongst the 3 classifiers

Classifier	Precision (P)	Recall (R)	Macro F1 (F1)
Logistic Regression	*0.831549*	*0.761438*	*0.838193*
SVM	*0.844382*	*0.782159*	*0.850263*
Naive Bayes	*0.914697*	*0.564363*	*0.778377*

Table of Results

For one, for this binary text classification task, it makes sense that the most suitable metric is the macro F1 score. The macro F1 score provides a balanced measure of the model's precision and recall across both classes (high-star or low-star), without favoring the class that occurs more frequently.

Breaking down the results above:

1. **Precision**: Naive Bayes achieved the highest precision, but this came at the expense of low recall. This high precision means that when Naive Bayes predicts a high-star review, it is very likely to be correct; however, the low recall reveals it is very selective on what it deems to be a high star review, missing many true positive cases
2. **Recall**: SVM had the highest recall, indicating it was better at correctly identifying high-star reviews
3. **Macro F1 Score**: SVM also achieved the highest macro F1 score, making it the best overall model. The macro F1 score considers both precision and recall, and SVM's high F1 means it finds a proper balance between those two parameters

Based on the results table above, no, SVM does not provide the highest value for all metrics. While SVM had the highest macro F1 score and recall, Naive Bayes had the highest precision. However this does not mean something is wrong; rather, this indicates a trade-off: Naive Bayes was very conservative, prioritizing precision but sacrificing recall, while SVM offered a more balanced performance.

Which class is harder to predict–this is a difficult question. At first, I thought high-star reviews may be harder to predict, as we considered "neutral sentiment" to be considered a low-star review. Thus, in cases where the model is uncertain about whether a review is positive or negative, it might default to labeling it as low-star. However, after thinking it through, it makes sense that by categorizing neutral reviews as low-star, we increase the complexity of the low-star class, therefore doing more harm than good. This means that low-star reviews encompass a wider range of sentiments, from outright negative to more neutral or mildly disappointed, which I feel would make it harder for the model to distinguish low-star reviews from genuinely positive high-star reviews. Furthermore, especially for more neutral sentiments, people often express dissatisfaction in indirect or mixed ways, and even use negations of traditionally positive words–for example “not good” or “I don’t love it”. This ambiguity can confuse the model, especially if it struggles to understand context and/or word pairings.

Part 2: *Classification with Large Language Models*

---

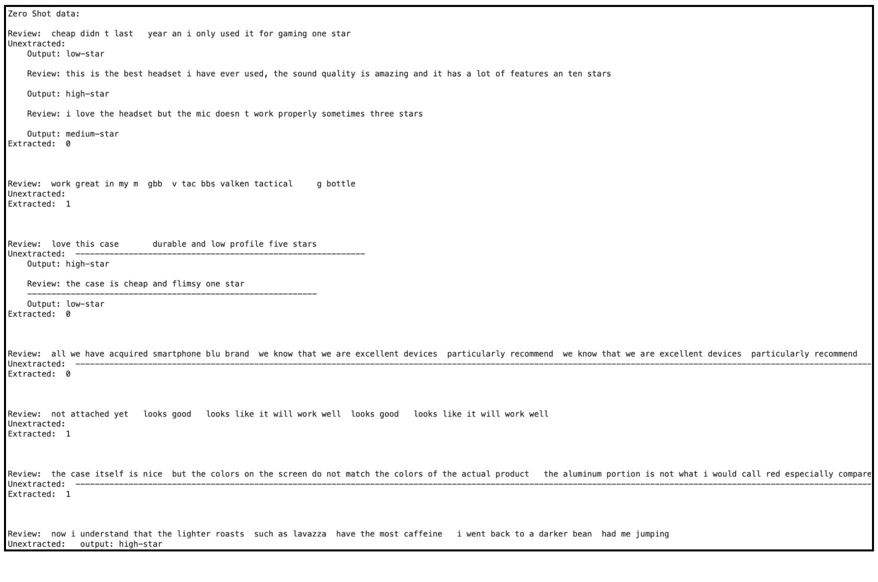
1. Description

For this part, your goal is to measure the performance of an open-source large language model of the review classification task and report their performance. For this task, you do not have to train your model. You will apply different in-context-learning/ prompting approaches on the provided test set and report the LLMs' precision, recall, and macro F1-score. Refer to this guide (https://www.promptingguide.ai/techniques) and X-hour materials provided in the class for an overview of different prompting techniques.

2. Results

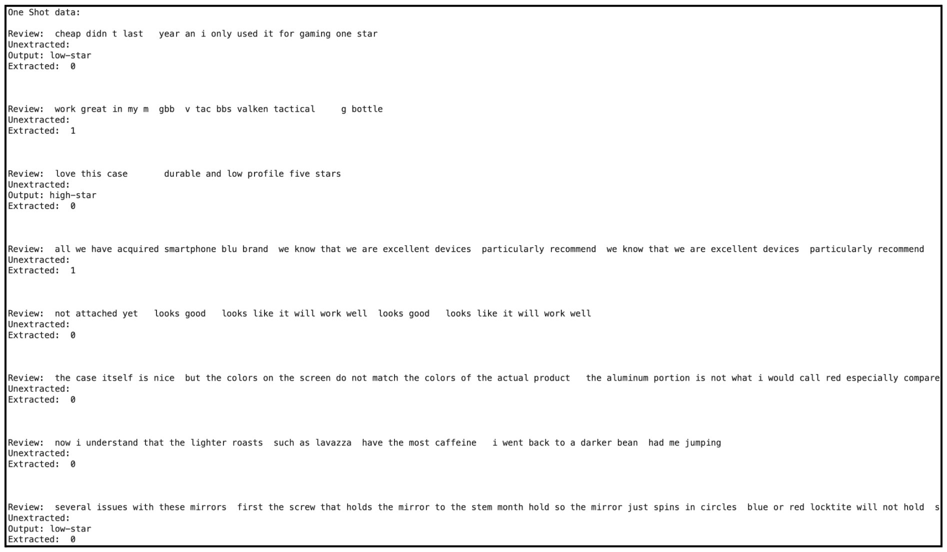
Before diving into the results, I would first like to acknowledge the difficulties I encountered while working with the HuggingFace model. The model frequently failed to produce outputs, despite numerous attempts to modify the task instructions. Even when using simple commands or specifying the desired output format, the model often ignored instructions such as “only output ‘high-star’ or ‘low-star.’” Additionally, the model’s behavior was inconsistent: it would sometimes work, but after a Colab disconnect, the same prompts would no longer yield results. Multiple consultations with TAs did not resolve the issue. A classmate suggested using the API from the HuggingFace page instead of the LangChain implementation, which was reportedly faulty, but this provided only marginal improvement. Overall, I spent hours putting in the work into this assignment and feel my code accurately reflects the expectations of the assignment, but the model did not work with me properly, which neither myself or TAs could diagnose–underscoring the unreliability of using the Huggingface model. Here are screenshots from instances when the model did work, as proof:

Zero shot:



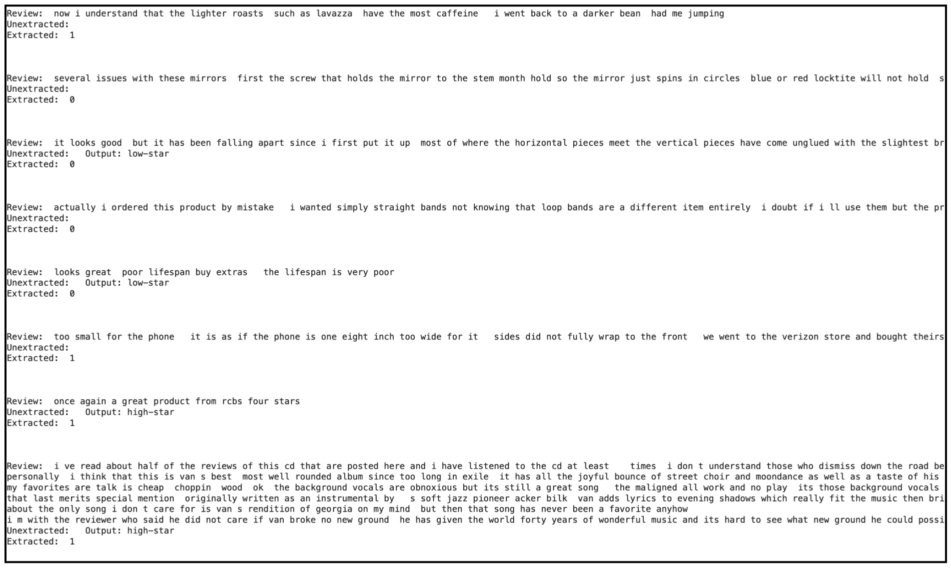
```
Zero Shot data:

Review:  cheap didn t last   year an i only used it for gaming one star
Unextracted:
    Output: low-star

    Review: this is the best headset i have ever used, the sound quality is amazing and it has a lot of features an ten stars

    Output: high-star

    Review: i love the headset but the mic doesn t work properly sometimes three stars

    Output: medium-star
Extracted:  0



Review:  work great in my m  gbb  v tac bbs valken tactical     g bottle
Unextracted:
Extracted:  1



Review:  love this case       durable and low profile five stars
Unextracted:  ------------------------------------------------------------
    Output: high-star

    Review: the case is cheap and flimsy one star
    ------------------------------------------------------------
    Output: low-star
Extracted:  0



Review:  all we have acquired smartphone blu brand  we know that we are excellent devices  particularly recommend  we know that we are excellent devices  particularly recommend
Unextracted:  ------------------------------------------------------------------------------------------------------------------------------------------------------------------
Extracted:  0



Review:  not attached yet   looks good   looks like it will work well  looks good   looks like it will work well
Unextracted:
Extracted:  1



Review:  the case itself is nice  but the colors on the screen do not match the colors of the actual product   the aluminum portion is not what i would call red especially compare
Unextracted:  ------------------------------------------------------------------------------------------------------------------------------------------------------------------
Extracted:  1



Review:  now i understand that the lighter roasts  such as lavazza  have the most caffeine   i went back to a darker bean  had me jumping
Unextracted:   output: high-star
```

One shot:



```
One Shot data:

Review:  cheap didn t last   year an i only used it for gaming one star
Unextracted:
Output: low-star
Extracted:  0



Review:  work great in my m  gbb  v tac bbs valken tactical     g bottle
Unextracted:
Extracted:  1



Review:  love this case       durable and low profile five stars
Unextracted:
Output: high-star
Extracted:  0



Review:  all we have acquired smartphone blu brand  we know that we are excellent devices  particularly recommend  we know that we are excellent devices  particularly recommend
Unextracted:
Extracted:  1



Review:  not attached yet   looks good   looks like it will work well  looks good   looks like it will work well
Unextracted:
Extracted:  0



Review:  the case itself is nice  but the colors on the screen do not match the colors of the actual product   the aluminum portion is not what i would call red especially compare
Unextracted:
Extracted:  0



Review:  now i understand that the lighter roasts  such as lavazza  have the most caffeine   i went back to a darker bean  had me jumping
Unextracted:
Extracted:  0



Review:  several issues with these mirrors  first the screw that holds the mirror to the stem month hold so the mirror just spins in circles  blue or red locktite will not hold  s
Unextracted:
Output: low-star
Extracted:  0
```

Few shot:



```
Review:  now i understand that the lighter roasts  such as lavazza  have the most caffeine   i went back to a darker bean  had me jumping
Unextracted:
Extracted:  1



Review:  several issues with these mirrors  first the screw that holds the mirror to the stem month hold so the mirror just spins in circles  blue or red locktite will not hold  s
Unextracted:
Extracted:  0



Review:  it looks good  but it has been falling apart since i first put it up  most of where the horizontal pieces meet the vertical pieces have come unglued with the slightest br
Unextracted:   Output: low-star
Extracted:  0



Review:  actually i ordered this product by mistake   i wanted simply straight bands not knowing that loop bands are a different item entirely  i doubt if i ll use them but the pr
Unextracted:
Extracted:  0



Review:  looks great  poor lifespan buy extras   the lifespan is very poor
Unextracted:   Output: low-star
Extracted:  0



Review:  too small for the phone   it is as if the phone is one eight inch too wide for it   sides did not fully wrap to the front   we went to the verizon store and bought theirs
Unextracted:
Extracted:  1



Review:  once again a great product from rcbs four stars
Unextracted:   Output: high-star
Extracted:  1



Review:  i ve read about half of the reviews of this cd that are posted here and i have listened to the cd at least    times  i don t understand those who dismiss down the road be
personally  i think that this is van s best  most well rounded album since too long in exile  it has all the joyful bounce of street choir and moondance as well as a taste of his
my favorites are talk is cheap  choppin  wood  ok  the background vocals are obnoxious but its still a great song   the maligned all work and no play  its those background vocals
that last merits special mention  originally written as an instrumental by   s soft jazz pioneer acker bilk  van adds lyrics to evening shadows which really fit the music then bri
about the only song i don t care for is van s rendition of georgia on my mind  but then that song has never been a favorite anyhow
i m with the reviewer who said he did not care if van broke no new ground  he has given the world forty years of wonderful music and its hard to see what new ground he could possi
Unextracted:   Output: high-star
Extracted:  1
```

I implemented a function to parse the text output from the model. Initially, the parser identified whether the model's output contained "high-star" or "low-star." If the output was missing or ambiguous, the parser randomly chose between 0 (low-star) and 1 (high-star). This method was intended to avoid class bias, but it introduced significant variability since oftentimes an output was missing. Each run produced different results, which was problematic for consistent analysis. Given that I identified in Part 1 the higher difficulty of classifying low-star reviews, I ultimately decided to default to low-star when the parser could not identify an output. This approach reduced variability but may have introduced a slight bias toward the low-star class.

Tuning the task_instruction was the most critical component of the assignment. The model was sensitive enough that even changing one word would affect the results. My initial strategy involved crafting complex and detailed prompts, hoping to guide the model explicitly. However, I found the results with this strategy to be quite poor, often not outputting anything. The more basic I made the prompt, the more it was able to concretely output ‘high-star’ or ‘low-star’. The model was so sensitive that even using the exact same task_instruction between zero shot and one shot, one shot will not have any outputs. Changing to the Huggingface API helped this issue slightly. It was much more receptive to clear instructions, however it was still critical that the instructions were not overly complex. Oftentimes the model would output way more information than necessary, appearing to ramble, hence the need for the parser, and yet adding “Only output ‘high-star’ or ‘low-star’” would not change much. Overall, the model was very finicky, but performed best when instructed to "classify" and when I clearly defined that "positive" reviews were high-star and "negative" reviews were low-star.

The performance did change by varying the examples given for one and few shot testing. Using a similar logic I had with task_instructions in the beginning, I thought it was better to use long, nuanced reviews so the API had exposure to review sentiments that were not overtly obvious. However, like I found previously, this actually did more harm than good. The model responded better with example reviews that included key sentiment indicators, such as “love” or “bad”, etc. This plays into randomly vs. manually selecting reviews. I experimented with two approaches:

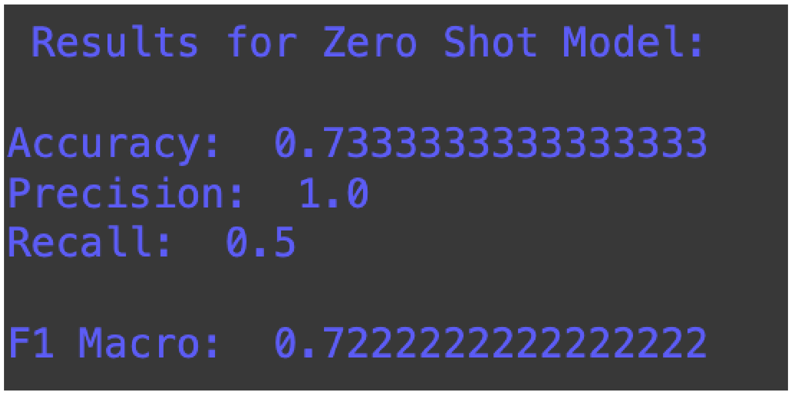
Random Selection: Choosing three examples randomly from the training set

Manual Selection: Carefully selecting three examples that I thought were representative of the task, using reviews with clear sentiments

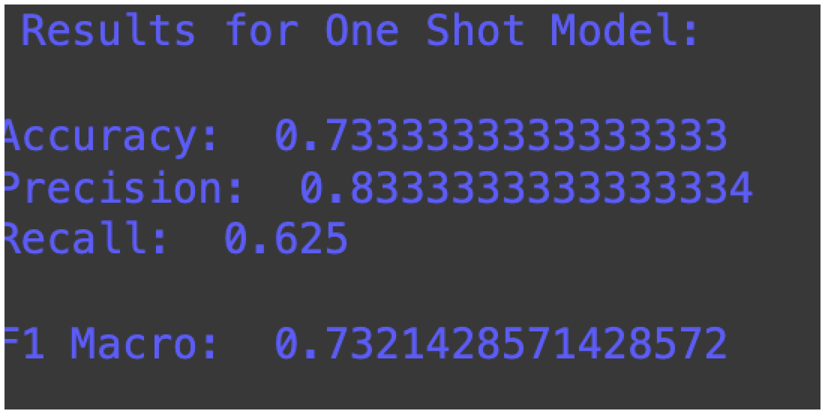
Randomly selecting reviews produced generally worse results, as manually I was able to find reviews with clear and concise language for the model to interpret.

My table of best results I received through numerous tests can be seen below. These results are influenced by the fact that this data utilized my previous parsing function, which randomized the extracted output between 0 and 1 if no output was produced, instead of defaulting to 0.

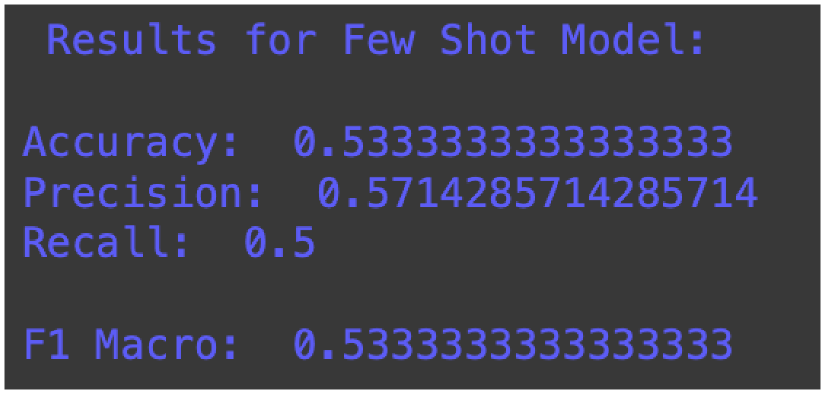
Classifier	Precision (P)	Recall (R)	Macro F1 (F1)
Zero Shot	*1.0*	*0.5*	*0.72222*
One Shot	*0.8333*	*0.625*	*0.732*
Few Shot	*0.571*	*0.5*	*0.533*



```
 Results for Zero Shot Model:

Accuracy:  0.7333333333333333
Precision:  1.0
Recall:  0.5

F1 Macro:  0.7222222222222222
```



```
 Results for One Shot Model:

Accuracy:  0.7333333333333333
Precision:  0.8333333333333334
Recall:  0.625

F1 Macro:  0.7321428571428572
```



```
 Results for Few Shot Model:

Accuracy:  0.5333333333333333
Precision:  0.5714285714285714
Recall:  0.5

F1 Macro:  0.5333333333333333
```