

Model Performance Evaluation Report

Shinhoo Kim. 21900136 21900136@handong.ac.kr
[source code](#)

Introduction

This report evaluates the performance of a rating prediction model developed for the Amazon review dataset. The model aims to predict the rating given by users based on their textual reviews. The evaluation metrics used are Mean Squared Error (MSE) and Mean Absolute Error (MAE).

Data Preprocessing and Model Overview

The dataset was preprocessed using various text preprocessing techniques, including HTML tag removal, punctuation removal, lowercase conversion, lemmatization, tokenization, and stopword removal. These steps were performed to clean and normalize the text data, reducing noise and improving the quality of input for the model.

The model uses a word-to-index dictionary for integer encoding of the preprocessed reviews. Each word in the reviews is encoded as an integer value using the dictionary. Additionally, a word-to-rating dictionary is created, mapping each word to its corresponding rating.

Code Explanation:

[Part 0](#)

1. The code in the above link reads in a CSV file containing Amazon product reviews and assigns it to a Pandas DataFrame called `df`.
2. Next, the column names are changed to more descriptive names using the `columns` attribute of the DataFrame. The new names for the columns are: `title`, `rank`, `date`, `name`, and `review`.

The code then extracts the name of the reviewer from the `name` column using regular expressions. The regular expression

``r'By\s+(.*?)\s+on`` matches the substring "By", followed by one or more whitespace characters, followed by any non-whitespace characters, followed by one or more whitespace characters, followed by the substring "on". The parentheses around `.*?` indicate a capturing group, which allows the matched name to be extracted later.

3. Remove HTML tags (if necessary):

The function `remove_html_tags` uses the BeautifulSoup library to remove any HTML tags from the text. It utilizes the `html.parser` to parse the HTML and extract the text content.

4. Remove punctuation and replace with lowercase:

The function `remove_punctuation` uses regular expressions to remove punctuation marks from the text. The `re.sub` function substitutes any non-word characters (`[^\w\s]`) with an empty string. Additionally, the `lower()` method is applied to convert the text to lowercase.

5. Lemmatization + Tokenization:

The NLTK library is used for lemmatization and tokenization. First, the WordNet lemmatizer (`WordNetLemmatizer()`) is instantiated. The `lemmatize_text` function tokenizes the text using `nltk.word_tokenize`, and then lemmatizes each token using the WordNet lemmatizer. The lemmatized tokens are stored in a list.

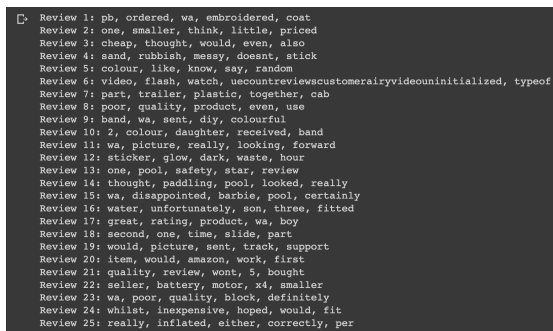
6. Remove stopwords:

The NLTK library's stopwords corpus is downloaded using `nltk.download('stopwords')`. The set of English stopwords is obtained using `stopwords.words('english')`. The function `remove_stopwords` removes stopwords from the tokenized text by filtering out tokens that exist in the stopwords set.

[Part 1](#)

This code iterates over each review in the 'review' column of the dataframe 'df'. For each review, it calculates the frequency distribution of the words in the review using the FreqDist() function from the nltk package. It then extracts the five most frequent words from the frequency distribution using the most_common() method and saves the words in a list. Finally, it prints the words in the list separated by commas.

The function print_most_frequent_words() takes a single argument 'review', which is a list of words representing the lemmatized and filtered tokens of a review. The function calculates the frequency distribution of the words using the FreqDist() function, extracts the five most frequent words using the most_common() method, extracts the words from the list of tuples returned by most_common(), and prints the words separated by commas.



```

Review 1: ph, ordered, wa, embroidered, coat
Review 2: one, smaller, think, little, priced
Review 3: cheap, thought, would, even, also
Review 4: sand, rubbish, messy, doesnt, stick
Review 5: colour, like, know, say, random
Review 6: video, flash, watch, usecountreviewscustomerairyyvideouninitialized, typeof
Review 7: part, trailer, plastic, together, cab
Review 8: poor, quality, product, even, use
Review 9: band, wa, sent, diy, colourful
Review 10: 2, colour, daughter, received, band
Review 11: wa, picture, really, looking, forward
Review 12: sticker, glow, dark, waste, hour
Review 13: one, pool, safety, star, review
Review 14: thought, padding, pool, looked, really
Review 15: wa, disappointed, barbie, pool, certainly
Review 16: water, unfortunately, son, three, fitted
Review 17: great, rating, product, wa, boy
Review 18: second, one, time, slide, part
Review 19: would, picture, sent, track, support
Review 20: item, would, amazon, work, first
Review 21: quality, review, wont, 5, bought
Review 22: seller, battery, motor, x4, smaller
Review 23: wa, poor, quality, block, definitely
Review 24: whilst, inexpensive, hoped, would, fit
Review 25: really, inflated, either, correctly, per
Review 26: not, coding, plastic, buttons, dirty

```

[output screenshot]

Part 2 - v1

The code provided creates an empty dictionary called 'word_to_index'. It then iterates over each review in the 'review' column of the DataFrame 'df'. Within each review, it iterates over each word. For each word, it checks if the word already exists as a key in the 'word_to_index' dictionary. If the word is not present, it adds it as a new key to the dictionary and assigns it the current length of the dictionary as the value.

Finally, it prints the 'word_to_index' dictionary, which contains a mapping of each unique word in the reviews to its corresponding index.

Part 2 - v2

The code provided creates an empty dictionary called 'word_to_rating'. It then iterates over each row in the DataFrame 'df' using the 'iterrows()' method. For each row, it retrieves the review and rating values.

Next, it iterates over each word in the review. For each word, it checks if the word already exists as a key in the 'word_to_rating' dictionary. If the word is present, it appends the rating to the list of ratings associated with that word. If the word is not present, it creates a new entry in the dictionary with the word as the key and the rating as a list.

Finally, it prints the 'word_to_rating' dictionary, which contains a mapping of each unique word in the reviews to a list of ratings associated with that word.

Part 4 - v3

The provided code performs the following steps:

1. Load 'amazon_test_df.csv' data and preprocess it:

The test DataFrame 'df_test' is loaded from the 'amazon_test_df.csv' file. The function 'preprocess_text' is defined to remove HTML tags, remove punctuation, convert the text to lowercase, perform lemmatization and tokenization, and remove stopwords. The column names of 'df_test' are renamed to match the column names used in the training DataFrame ('df'). The 'name' column is extracted using regular expressions, similar to the previous code. The 'review' column is preprocessed using the 'preprocess_text' function.

2. Encode the test review data using the word-to-index dictionary (integer encoding):
The function 'encode_review' is defined to encode a review by replacing each word with its corresponding index from the 'word_to_index' dictionary. The 'encoded_review' column is created by applying the 'encode_review' function to the 'review' column of 'df_test'. An 'index_to_word' dictionary is created to map the indices back to the original words.

3. Predict the rating of the test review:
The function 'predict_rating' is defined to predict the rating of a review based on the

word-to-rating dictionary. For each encoded review in the 'encoded_review' column, the corresponding ratings are retrieved from the 'word_to_rating' dictionary using the 'index_to_word' dictionary. The ratings are flattened into a single list and the average rating is calculated. The predicted ratings are stored in the 'predicted_rating' column of 'df_test'.

index	rank	predicted_rating
0	1.0	2.620893024891328
1	1.0	2.6249969724436743
2	1.0	2.6269946107784433
3	1.0	2.6278884421514
4	1.0	2.6326151422501
5	2.0	3.076324225865009
6	2.0	2.689130434726208
7	2.0	2.670175438984812
8	2.0	2.62983301727592
9	2.0	2.75440148450704
10	3.0	3.030303030303030
11	3.0	2.891620971848253
12	3.0	2.92469820481283
13	3.0	2.857283464588929
14	3.0	2.867286211822244
15	4.0	3.102541125411255
16	4.0	3.1864406779651016
17	4.0	2.7054784520547845
18	4.0	3.0330304447828445
19	4.0	2.87245073999332
20	5.0	3.3003925691698
21	5.0	3.688888888888889
22	5.0	3.048484848484848
23	5.0	2.74253191483617
24	5.0	3.050050005000504

[actual ranking & predicted ranking]

4. Evaluate the predicted result

The provided code uses the mean_squared_error and mean_absolute_error functions from the sklearn.metrics module to evaluate the predicted ratings compared to the actual ratings.

The actual ratings are extracted from the 'rank' column of the test DataFrame df_test.

The predicted ratings are extracted from the 'predicted_rating' column of df_test.

The mean_squared_error function calculates the mean squared error between the actual ratings and the predicted ratings, while the mean_absolute_error function calculates the mean absolute error.

Finally, the code prints the values of the mean squared error (MSE) and mean absolute error (MAE) as evaluation metrics.

```
Mean Squared Error (MSE): 1.6325950534704032
Mean Absolute Error (MAE): 1.0978852023173655
```

[MSE & MAE value]

Model Performance

The model's performance was evaluated using the following metrics:

Mean Squared Error (MSE):
1.6325950534704032
Mean Absolute Error (MAE):
1.0978852023173655

Interpretation and Discussion

The obtained MSE and MAE values reflect the average squared and absolute differences, respectively, between the predicted and actual ratings. Lower values for both metrics indicate better model performance.

Based on the MSE and MAE values obtained, the model demonstrates a reasonable level of performance in rating prediction. However, the interpretation of these metrics depends on the specific problem and context. Therefore, it is crucial to consider additional factors and domain-specific knowledge when assessing the goodness of the model.

Future Improvements (part4.5)

To further improve the model's performance, several areas could be explored:

- Feature Engineering: Consider additional features such as review length, sentiment analysis, or other relevant contextual information to enhance the model's predictive power.
- Advanced Model Techniques: Explore more sophisticated model architectures, such as recurrent neural networks (RNNs) or transformer-based models (e.g., BERT), which are designed to handle sequential and contextual data effectively.
- Hyperparameter Tuning: Optimize the hyperparameters of the model, such as learning rate, regularization techniques, or model architecture parameters, to achieve better performance.
- Data Augmentation: Generate synthetic data or augment the existing dataset to provide more diverse examples for training the model.

Conclusion

In conclusion, the developed rating prediction model demonstrates reasonable performance based on the MSE and MAE evaluation metrics. The model successfully captures the relationship between the textual reviews and the corresponding ratings. However, further improvements and refinements could be explored to enhance its predictive capabilities.