**Algorithm Documentation**

**Introduction**

This document provides detailed documentation for the algorithm implemented in the web application. The algorithm focuses on categorizing reviews based on various criteria, including sentiment analysis, keyword weights, and feature extraction.

**Model Definition**

**Review Model**

The **Review** model is defined using Django's **models.Model** class, encompassing key attributes such as text, author, date, property, address, length, reasonableness, comprehensiveness, relevancy, and tonality.

**Attributes**

* **text**: The main content of the review.
* **author**: The author of the review.
* **date**: The date when the review was submitted.
* **property**: The property being reviewed.
* **address**: The address of the property.
* **length**: The length of the review.
* **reasonableness**: A metric for the reasonableness of the review.
* **comprehensiveness**: A metric for the comprehensiveness of the review.
* **relevancy**: A metric for the relevancy of the review.
* **tonality**: The sentiment tonality of the review.

**Functions**

**calculate\_keyword\_weights(review)**

This function calculates the weights of keywords in a given review using tokenization, stopword removal, and frequency calculation.

def calculate\_keyword\_weights(review):

words = word\_tokenize(review.text)

words = [word for word in words if word not in stopwords.words('english')]

word\_freq = Counter(words)

word\_weights = {word: freq / len(words) for word, freq in word\_freq.items()}

return word\_weights

**extract\_features(review)**

Extracts adjectives and key phrases from the review using tokenization and part-of-speech tagging.

**def extract\_features(review):**

**words = word\_tokenize(review.text)**

**tagged\_words = pos\_tag(words)**

**adjectives = [word for word, pos in tagged\_words if pos == 'JJ']**

**key\_phrases = [**

**' '.join([adjectives[i], adjectives[i + 1]])**

**for i in range(len(adjectives) - 1)**

**]**

**return adjectives, key\_phrases**

**calculate\_tonality(review)**

Calculates the tonality of the review using sentiment analysis.

**def calculate\_tonality(review):**

**blob = TextBlob(review.text)**

**return blob.sentiment.polarity**

**calculate\_reasonableness(review, keyword\_weights, adjectives, key\_phrases)**

Calculates the reasonableness of the review based on keyword weights, adjectives, and key phrases.

**def calculate\_reasonableness(keyword\_weights):**

**return sum(keyword\_weights.values())**

**calculate\_comprehensiveness(review, keyword\_weights, adjectives, key\_phrases)**

Calculates the comprehensiveness of the review based on keyword weights, adjectives, and key phrases.

**def calculate\_comprehensiveness(keyword\_weights):**

**return len(set(keyword\_weights.keys()))**

**calculate\_relevancy(review, keyword\_weights, adjectives, key\_phrases)**

Calculates the relevancy of the review based on keyword weights, adjectives, and key phrases.

**def calculate\_relevancy(keyword\_weights):**

**return sum(keyword\_weights.values())**

**update\_review\_model(review)**

Updates the **Review** model with new features by calling the above functions and saving the updated review.

**def update\_review\_model(review):**

**keyword\_weights = calculate\_keyword\_weights(review)**

**adjectives, key\_phrases = extract\_features(review)**

**review.length = len(review.text)**

**review.reasonableness = calculate\_reasonableness(keyword\_weights)**

**review.comprehensiveness = calculate\_comprehensiveness(keyword\_weights)**

**review.relevancy = calculate\_relevancy(keyword\_weights)**

**review.tonality = calculate\_tonality(review)**

**review.save()**