International Journal of Science and Research (IJSR)
ISSN: 2319-7064
Impact Factor 2025: 7.089

Using Parikh Vectors in Pattern Appreciation: A Practical Approach for Text and Sensitivity Ordering

Dr. M. K. TamilSelvi[1], Vaishnavi T.[2]

[1]Department of Mathematics, Alpha College of Engineering, Chennai, Tamil Nadu, India

[2]Department of Computer Science and Engineering, Alpha College of Engineering, Chennai, Tamil Nadu, India

Abstract: ***Parikh vectors give a compact representation of strings with frequency. It is also useful in pattern recognition tasks. This involves text classification and sensitivity exploration. In this paper we have tried to present a theory of Parikh vectors. Thus, exploring the practical application in sensitivity exploration. Also, we put forward a pre-processing and feature exploration by constructing Parikh vectors. We integrate with machine learning models.***

Keywords: Vector, pre-processing, sensitivity ordering

1. Introduction

In Natural Language processing (NLP) the fundamental aspect is to represent text data into symbols Parikh vector is one such representation. This concept has been originated from formal language theory. Contrast to sequence-sensitive models, Parikh vectors captures the count of symbols and not the position. This is an advantage to find out the distribution of characters or tokens which more important their positions. In this paper we study the application of Parikh vectors in pattern recognition and focus on sensitivity exploration which includes short text messages as inputs

2. Parikh Vectors: Concept and Example

A Parikh vector represents a word or string by counting how many times each symbol from a defined alphabet occurs in it. Suppose we have an alphabet Σ = {A, B, C}. For a word like “ABBAC,” we count:

- A appears 2 times,
- B appears 2 times,
- C appears once.

The Parikh vector is then: [2, 2, 1]. Thus, Parikh vectors offer summary of symbolic sequences based on frequency. Also it enable pattern matching and recognition in a simple manner.

3. Application in Pattern Appreciation

Parikh vectors can helpsequence recognition problems, specifically where the order of symbols is less important than their occurrence. A prominentuse is in text sensitivityordering, where the expressivenature of a message can often be concluded from the frequency of sensitivity - behaviour tokens.

3.1 Use Case: Text-Based sensitivity ordering

We explore a binary classification task distinguishing between *positive* and *negative* sentiment messages:
Dataset Examples:
* Positive: *”I like this item!”*, *”Great experience.”*
* Negative: *”Dissatisfied with the deal.”*, *”Poor quality.”*

3.2 Method

Step 1: Text Preprocessing- Text pre-processing is an important pace in natural language processing (NLP). This includes transforming unstructured text data to organize it for investigation. The following procedure such as tokenization, stemming, lemmatization, stop-word elimination, and part-of-speech classification is carried over

```
from collections import Counter
deffind_anagram_positions(text, pattern):
pattern_length = len(pattern)
   pattern_
vector = Counter(pattern)
window_vector = Counter(text[:pattern_length])
positions = []
ifwindow_vector == pattern_vector:
positions.append(0)
fori in range(1, len(text) - pattern_length + 1):
      # Remove the outgoing character
outgoing_char = text[i - 1]
window_vector[outgoing_char] -= 1
ifwindow_vector[outgoing_char] == 0:
delwindow_vector[outgoing_char]
      # Add the incoming character
incoming_char = text[i + pattern_length - 1]
window_vector[incoming_char] += 1
      # Compare Parikh vectors
ifwindow_vector == pattern_vector:
positions.append(i)
return positions
Example :
text = "cbadebac"
pattern = "abc"

matches = find_anagram_positions(text, pattern)
print("Pattern found at positions:", matches)
Output:
Pattern found at positions: [0, 4, 5]
```

Tokenization for Parikh vectors

Tokenization Function: Here the acquired text is broken down into smaller units, such as words or phrases, called tokens. It also removes punctuation, splitting sentences, and management of various types of formatting.

```
deftokenize_characters(text):
return list(text)
Given:
text = "banana"

Tokenization:
['b', 'a', 'n', 'a', 'n', 'a']
Python Code
from collections import Counter

parikh_vector = Counter(text)
print(parikh_vector)
Output:
Counter({'a': 3, 'n': 2, 'b': 1})
```

Sentiment Analysis: This is a method in NLP wherein the sentiment nature is assessed and analyzed i.e., it articulates a positive, negative or a neutral sentiment.

```
text = "I like the outstanding service but dislike the delay."
# Suppose we define sentiment-labeled word categories
positive_words = {"like", "outstanding", "good", "great"}
negative_words = {"hate", "bad", "poor", "awful",”dislike”,”delay”}
# Tokenize the sentence
tokens = text.lower().split()
# Compute a sentiment-oriented Parikh vector
from collections import Counter
defsentiment_parikh_vector(tokens, pos_words, neg_words):
vec = Counter()
for token in tokens:
if token in pos_words:
vec['positive'] += 1
elif token in neg_words:
vec['negative'] += 1
else:
vec['neutral'] += 1
returnvec
vector = sentiment_parikh_vector(tokens, positive_words, negative_words)
print(vector)
Output :
Counter({'positive': 2, 'negative': 1, 'neutral': 6})
Sentiment Classifier Using Parikh-like Vector
defclassify_sentiment(text):
positive_words = {"love", "excellent", "good", "great", "amazing", "happy",”outstanding”}
negative_words = {"hate", "bad", "poor", "awful", "sad", "terrible",”dislike”,”delay”}
tokens = text.lower().split()
vec = sentiment_parikh_vector(tokens, positive_words, negative_words)
ifvec['positive'] >vec['negative']:
return "Positive"
elifvec['negative'] >vec['positive']:
return "Negative"
else:
return "Neutral"
```

Removing Stop words using NLTK- In order to improve the preciseness Stop words are eliminated often from NLP owing to their Low importance of textual meaning.

```
importnltk
fromnltk.corpus import stopwords
# Download stop words (only once)
nltk.download('stopwords')
defremove_stop_words(text):
stop_words = set(stopwords.words('english'))
tokens = text.lower().split()
filtered = [word for word in tokens if word not in stop_words]
return filtered
Example:
text = "I like the outstanding service but dislike the delay."
print(remove_stop_words(text))
Output :
['like', 'outstanding ', 'service', 'dislike ', 'delay’.']
```

4. Improving Performance over Feature Instigating

Introduce additional techniques to the basic Parikh vectors to improve the performance:

- Word Size: Total number of characters or tokens
- Presence of Feelings or Reference Marks: Solidpointers of feelings
- Keyword Occurrence: Precise positive or negative words

5. Benefits and Drawbacks

Parikh vectors are useful in effective extraction of text data. It aids to find out the information about frequency. Thus, it is suitable for sensitivity exploration. However, the only disadvantage of Parikh vectors is it does not take into account, the position of the symbols. This is a limitation for certain tasks. In spite of this, with the help of NLP embedding, application of Parikh vectors could lead to an hybrid model which facilitate both frequency and contextual semantics.

6. Conclusion

From this study, we understand the usage of Parikh vectors in sequence extraction where symbols are involved. By converting the given text into Parikh vector and applying Text Pre-processing and other techniques we have demonstrated the effectiveness of Parikh vectors in sentiment analysis.

References

[1] Parikh, R. (1966). On context-free languages. *Journal of the ACM, 13*(4), 570-581.
[2] Joachims, T. (1998). Text categorization with Support Vector Machines: Learning with many relevant features. *European Conference on Machine Learning*, 137-142.
[3] Manning, C. D., Raghavan, P., &Schütze, H. (2008). Introduction to Information Retrieval. Cambridge University Press