Experiment 5: Word2Vec Handling Abbreviations and the Boolean Retrieval Model

#Objective:

We explore two distinct aspects of natural language processing (NLP): word embeddings using Word2Vec and the Boolean Retrieval Model. Word2Vec is used to train a model to capture word relationships, while the Boolean Retrieval Model is designed to handle basic search queries using logical operators.

#Code

```
model = gensim.models.Word2Vec(window=10, min_count=2, workers=4)
model.build_vocab(story)
model.train(story, total_examples=model.corpus_count, epochs=model.epochs)

model.wv.most_similar('dragon')
model.wv.most_similar('thrones')
```

#Output:
Most similar to "dragon": `star`, `world`, `god`, `comet`, etc.
Most similar to "thrones": `number`, `fisherfolk`, `nobles`, `septons`, etc.

```
model.wv.most_similar(positive=['prince', 'female'], negative=['female'])
```

#Output:
`martell`, `princess`, `myrcella`,

```
model.wv.doesnt_match(['winter', 'summer', 'spring', 'prince'])
```

#Output:
`prince`

```
from sklearn.decomposition import PCA
import plotly.express as px

pca = PCA(n_components=3)
result = pca.fit_transform(model.wv.get_normed_vectors())
fig = px.scatter_3d(result[:100], x=0, y=1, z=2, color=model.wv.index_to_key[:100])
fig.show()

def translator(user_string):
    user_string = nltk.word_tokenize(user_string)
    translated_string = []
    for word in user_string:
        if word in slangs.keys():
            translated_string.append(slangs[word])
        else:
            translated_string.append(word)
    return " “.join(translated_string)

translator("I am happy AFAIK, BBS")

#Output:
`'I am happy As Far As I Know, Be Back Soon'`

class BooleanRetrievalModel:
    def __init__(self):
        self.inverted_index = defaultdict(set)

    def add_document(self, doc_id, text):
        terms = text.lower().split()
        for term in terms:
            self.inverted_index[term].add(doc_id)

    def _get_postings(self, term):
        return self.inverted_index.get(term, set())

    def boolean_search(self, query):
        tokens = query.lower().split()
        result = set()
        operator = None
        for token in tokens:
            if token == 'and':
                operator = 'and'
            elif token == 'or':
                operator = 'or'
            elif token == 'not':
                operator = 'not'
            else:
                postings = self._get_postings(token)
                if not result:
                    result = postings
                elif operator == 'and':
                    result &= postings
                elif operator == 'or':
                    result |= postings
                elif operator == 'not':
                    result -= postings
                operator = None
        return result

brm = BooleanRetrievalModel()
brm.add_document(1, "the quick brown fox")
brm.add_document(2, "jumped over the lazy dog")
brm.add_document(3, "the fox is quick and the dog is lazy")
brm.add_document(4, "brown fox brown dog")

# Boolean Queries
print(brm.boolean_search("quick AND fox"))
print(brm.boolean_search("fox OR dog"))
print(brm.boolean_search("fox AND NOT lazy"))
```

#Output:

```
Accuracy: 0.8882666666666666
`quick AND fox`: `{1, 3}`
`fox OR dog`: `{1, 2, 3, 4}`
`fox AND NOT lazy`: `{1, 4}`
```

#Conclusion:

This experiment demonstrates how Word2Vec and the Boolean Retrieval Model can be effectively used for different NLP tasks. Word2Vec successfully captures semantic relationships between words, as seen in the similar words for "dragon" and "thrones," and can isolate outliers like "prince" from a seasonal context. The PCA visualization further highlights the clustering of word vectors, revealing semantic proximities in a 3D space. Additionally, the Boolean Retrieval Model shows promising results for simple logical queries, providing accurate document retrieval based on keyword combinations and logical operators. Together, these techniques enhance our ability to process, search, and interpret text data.