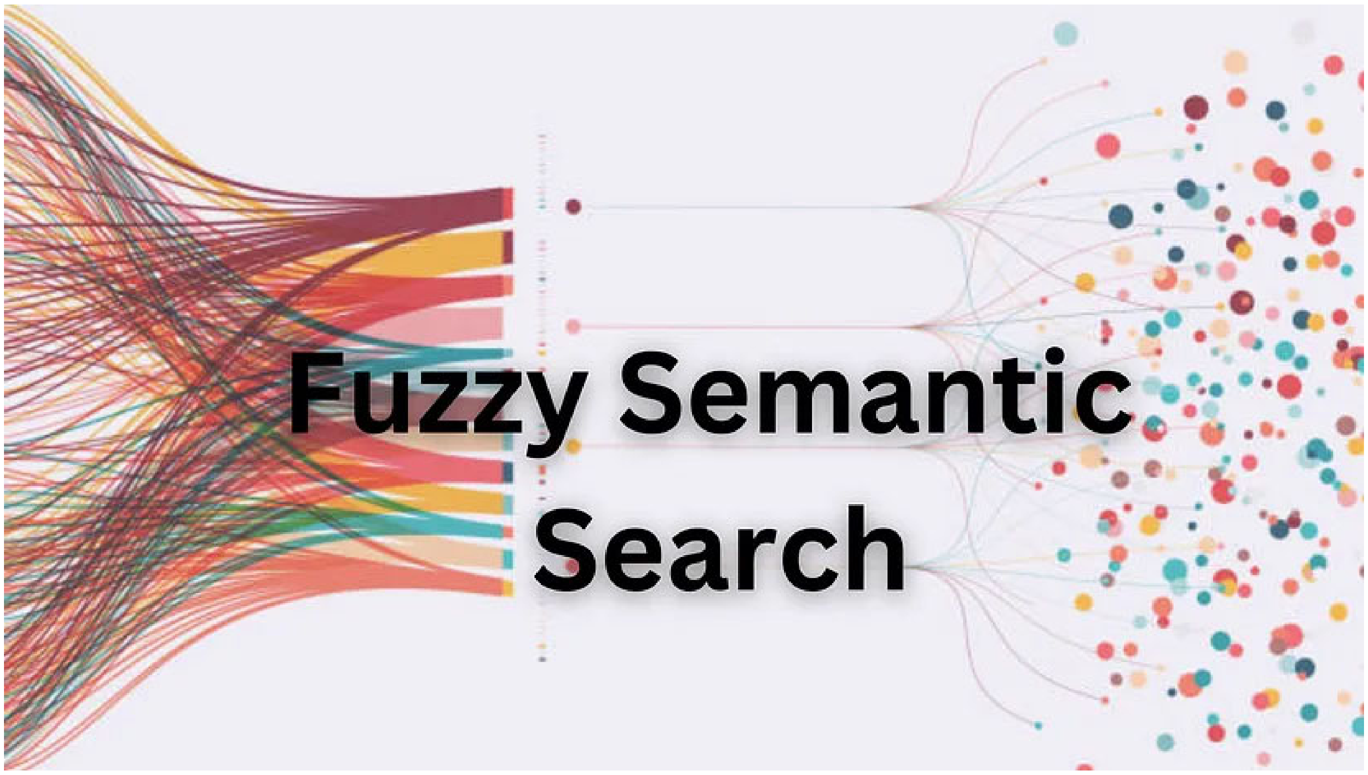
Get unlimited access to the best of Medium for less than $1/week. Become a member

Fuzzy Semantic Search a highly effective search technique

azhar · Follow
Published in azhar labs · 4 min read · Apr 19

2 1



Welcome to the world of Fuzzy Semantic Search! It's a powerful search technique that combines fuzzy matching and semantic search to provide accurate and relevant search results. It's widely used in various fields like e-commerce, healthcare, and finance, to name a few.

Fuzzy Semantic Search can match words or phrases even if they did not spell exactly the same way, and it understands the meaning of the words and the context in which they are used. Combining these two techniques, can more accurate and relevant search results.

In this blog, we gonna see how to implement that from scratch. before that, we gonna see some of the advantages of Fuzzy Semantic Search.

There are many advantages of Fuzzy Semantic Search. It provides more accurate search results compared to traditional search techniques. It improves the user experience by providing more relevant search results, which can help users find what they are looking for more quickly and easily. It also helps organizations save time and money by reducing the number of irrelevant search results, which can improve the efficiency of the search process.

To implement Fuzzy Semantic Search, we can use Faiss, an open-source library for efficient similarity search and clustering of dense vectors. Faiss provides a wide range of algorithms for similarity search, including LSH, PCA, and IVF. It's widely used in various fields, including computer vision, natural language processing, and recommender systems.

The implementation of Fuzzy Semantic Search using Faiss involves several steps.

1. **Data Preprocessing:** The first step in implementing Fuzzy Semantic Search is to preprocess the data. This involves cleaning the data, removing stop words, and converting the text into a vector representation.

```
def preprocess_text(text):
    stop_words = set(stopwords.words('english'))
    tokens = word_tokenize(text)
    tokens = [token.lower() for token in tokens if token.isalpha() and token.low
    return ' '.join(tokens)
```

This can be done using various techniques, including TF-IDF, word2Vec, and Doc2Vec, in this case, I'm using the Universal Sentence encoder.

```
embed = hub.load('https://tfhub.dev/google/universal-sentence-encoder-large/5')

dataset = pd.DataFrame({'text': ['The quick brown fox jumps over the lazy fox',
                                 'A quick brown dog jumps over the lazy fox',
                                 'The quick brown fox jumps over the lazy cat',
                                 'The quick brown cat jumps over the lazy dog',
                                 'A quick brown fox jumps over the lazy dog']})

def encode_text(text):
    embedding = embed([text]).numpy()
    return embedding
```

```
def get_embedding():
    embeddings = []
    for text in dataset['text']:
        preprocessed_text = preprocess_text(text)
        embedding = encode_text(preprocessed_text)
        embeddings.append(embedding)
    embeddings = np.concatenate(embeddings, axis=0)
    return embeddings
```

2. Building and Index: The next step is to build an index using Faiss. This involves creating an index object and adding the vector representations of the data to the index. This can be done using various indexing methods, including LSH, PCA, and IVF.

```
vectors = get_embedding()

index = faiss.IndexFlatIP(vectors.shape[-1])

def build_index(self):
    embeddings = np.array(vectors)
    index.add(embeddings)
```

3. Semantic Search: The next step is to perform a semantic search on the search query. This involves using the vector representation of the search query to find similar vectors in the index. This can be done using various similarity measures, including cosine similarity and Euclidean distance.

4. Fuzzy Matching: The final step is to perform a semantic search on the search query. This can be done using various techniques, including Jaro-Winker distance, Levenshtein distance, and N-Gram similarity.

```
def search(query, k=5, threshold=0):
    preprocessed_query = preprocess_text(query)
    embedding = encode_text(preprocessed_query)
    #Semantic Search
    distances, indices = index.search(embedding.reshape(1, -1), k)
    results = []
    for distance, index in zip(distances[0], indices[0]):
        text = dataset.iloc[index]['text']
        #Fuzzy matching
        score = fuzz.token_sort_ratio(preprocessed_query, self.preprocess_text(t
        if score >= threshold:
            results.append({'text': text, 'score': score, 'distance': distance})
    #Sorting based on semantic distance and fuzzy matching ratio
    results = sorted(results, key=lambda x: (x['distance'], x['score']), reverse
    return results
```

Open in app ↗

Search Write 1

```
 quick brown fox jumps over the lazy cat', 'score': 12, 'distance': 0.087689355}
 quick brown fox jumps over the lazy fox', 'score': 12, 'distance': 0.082839005}
uick brown fox jumps over the lazy dog', 'score': 12, 'distance': 0.053157043}
uick brown dog jumps over the lazy fox', 'score': 12, 'distance': 0.051227365}
 quick brown cat jumps over the lazy dog', 'score': 6, 'distance': 0.037477486}
```

In conclusion,

Fuzzy semantic search is a powerful search technique that can provide more accurate and relevant search results. By implementing it using Faiss, organizations can improve the accuracy of their search results, provide a better user experience, and increase efficiency. So, let's fuzzy match and semantically search our way to success!

complete project notebook link ...

Follow me on Medium and LinkedIn for more data science and deep learning-related article.

Semantic Search · Machine Learning · Fuzzy Matching · Fuzzy Semantic Search · Search Algorithm

Written by azhar

Follow

86 Followers · Editor for azhar labs

Data Scientist | Exploring daily one research paper. LinkedIn : https://www.linkedin.com/in/mohamed-azharudeen/

More from azhar and azhar labs