Implementación y Comparación de Técnicas de Resumen Extractivo

Paul Hans Murillo Dominguez

Introducción

Objetivo del Proyecto:
Implementar y comparar técnicas de resumen extractivo utilizando la métrica ROUGE.

01 **TF-IDF**

Relevancia de términos

02 **TextRank**

Algoritmo basado en grafos

03 **Combinación de TF-IDF y TextRank**

Sinergia de ambos métodos

04 **BERT**

Modelo avanzado preentrenado para análisis semántico

Flujo del Proceso

1. Lectura

Lectura de documentos

2. Preproc.

- Limpieza.
- Tokenización.
- Filtrado de ruido.

3. Modelos

TF-IDF, TextRank, Combinación y BERT

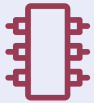
4. ROUGE

Análisis y comparación de resultados

Lectura y Preprocesamiento

Lectura de Documentos

Clase ***DocumentReader*** utiliza PyMuPDF para extraer texto de PDFs.



Mars

Clase ***Preprocessor*** que realiza:

- Eliminación de stopwords.
- Stemming y lematización para reducir ruido.
- Tokenización de oraciones y palabras.

DocumentReader

```python
class DocumentReader:
    def __init__(self, file_path: str):
        self.file_path = file_path

    def read_document(self) -> str:
        text = ""
        try:
            with fitz.open(self.file_path) as doc:
                for page in doc:
                    text += page.get_text()
        except Exception as e:
            raise ValueError(f"Error leyendo el archivo PDF: {e}")
        return text
```

Preprocessor

```python
class Preprocessor:
    def __init__(self, language: str = 'english'):
        self.stopwords = nltk.corpus.stopwords.words(language)
        self.stemmer = SnowballStemmer(language)
        self.lemmatizer = WordNetLemmatizer()

    def preprocess_sentences(self, sentences: List[str]) -> List[str]:
        preprocessed = []
        for sentence in sentences:
            if sum(char.isdigit() for char in sentence) / max(len(sentence), 1) > 0.3:
                continue
            sentence = sentence.lower()
            sentence = re.sub(r'[^a-zA-Z0-9\s]', '', sentence)
            words = word_tokenize(sentence)
            words = [
                self.lemmatizer.lemmatize(self.stemmer.stem(word))
                for word in words if word not in self.stopwords
            ]
            if words:
                preprocessed.append(' '.join(words))
        return preprocessed
```

Modelos Implementados

TF-IDF

Identifica palabras relevantes mediante frecuencia y rareza

Combinación de TF-IDF y TextRank

Selecciona oraciones que contienen palabras clave relevantes en ambos métodos

TextRank

Basado en grafos, mide la similitud entre oraciones para determinar su importancia

BERT

Modelo de lenguaje preentrenado que captura relaciones semánticas profundas

TF-IDF | TextRank

```python
class TFIDFSummarizer:
    @staticmethod
    def summarize(sentences: List[str], preprocessed_sentences: List[str], num_sentences: int = 1) -> str:
        vectorizer = TfidfVectorizer()
        tfidf_matrix = vectorizer.fit_transform(preprocessed_sentences)
        sentence_scores = np.sum(tfidf_matrix.toarray(), axis=1)
        ranked_indices = np.argsort(sentence_scores)[::-1]
        selected = [sentences[i] for i in ranked_indices[:num_sentences]]
        return ' '.join(selected)
```

```python
class TextRankSummarizer:
    @staticmethod
    def summarize(sentences: List[str], preprocessed_sentences: List[str], num_sentences: int = 1) -> str:
        vectorizer = TfidfVectorizer()
        tfidf_matrix = vectorizer.fit_transform(preprocessed_sentences)
        similarity_matrix = cosine_similarity(tfidf_matrix)
        nx_graph = nx.from_numpy_array(similarity_matrix)
        scores = nx.pagerank(nx_graph)
        ranked_indices = sorted(((scores[node], node) for node in nx_graph.nodes), reverse=True)
        selected = [sentences[i] for _, i in ranked_indices[:num_sentences]]
        return ' '.join(selected)
```

BERT

```python
class BERTSummarizer:
    """Genera resúmenes usando un modelo BERT extractivo preentrenado."""

    def __init__(self):
        self.model = Summarizer()

    def summarize(self, text: str, num_sentences: int = 1) -> str:
        return ''.join(self.model(text, num_sentences=num_sentences))
```

Evaluación con ROUGE

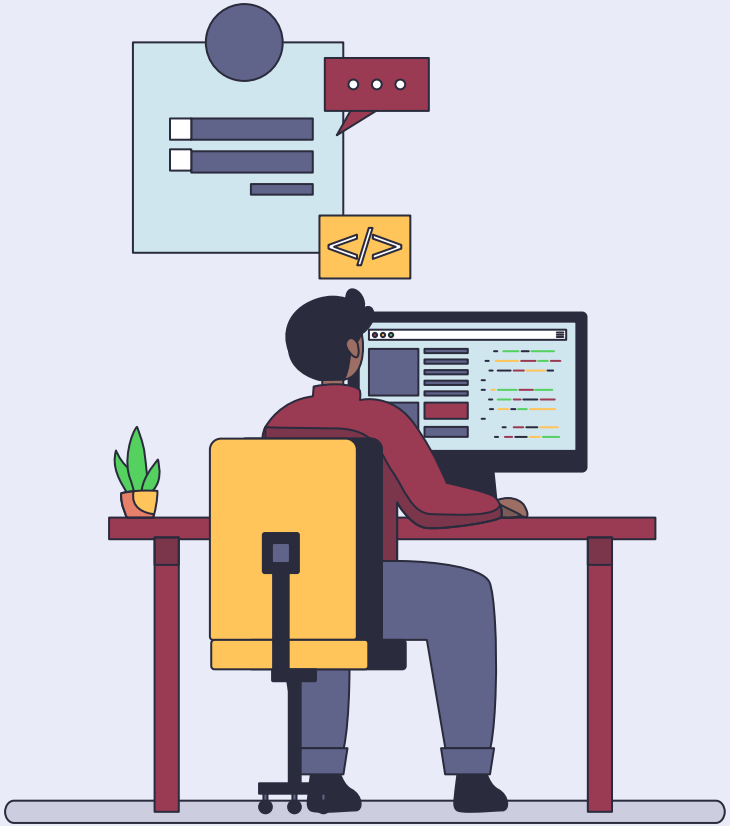
ROUGE (Recall-Oriented Understudy for Gisting Evaluation):
Evalúa la calidad de un resumen comparando con uno de referencia.

Métricas calculadas:

- ROUGE-1: Unigramas coincidentes
- ROUGE-2: Bigramas coincidentes
- ROUGE-L: Longitud de la subsecuencia común más larga

Resultados

Modelo	ROUGE-1	ROUGE-2	ROUGE-L
TF-IDF	0.2817	0.0284	0.1502
TextRank	**0.3314**	0.0809	0.2057
Combinado	0.3218	**0.1279**	**0.2184**
BERT	0.2420	0.0258	0.1529



Gracias!